**Анотація:** в курсовій роботі розроблений сервіс для тренування пам’яті «Memory.pro». Для розробки програмного забезпечення було використано мову програмування Java, бібліотеку JavaFX (з використанням FXML та SASS), JPA фреймворк EclipseLink та СУБД MYSQL.

**Аннотация:** в курсовой работе разработан сервис для тренировки памяти «Memory.pro». Для разработки программного продукта был использован язык программирования Java, библиотека JavaFX (с использование FXML и SASS), JPA фреймворк EclipseLink и СУБД MYSQL.

**Annotation:** a software product for memory training named «Memory.pro» is developed in this coursework. Java programing language, library JavaFX (with FXML and SASS), JPA framework EclipseLink and DBMS MYSQL have been used.

Зміст

[Вступ 4](#_Toc9855994)

[1. Аналіз та опис предметної області 5](#_Toc9855995)

[1.1. Загальний опис предметної області 5](#_Toc9855996)

[1.2. Аналіз вже існуючого програмного забезпечення 5](#_Toc9855997)

[1.3. Дослідження та опис предметної області 6](#_Toc9855998)

[1.4. Вибір засобів розробки 8](#_Toc9855999)

[2. Проектування 9](#_Toc9856000)

[2.1. Ескізний проект 9](#_Toc9856001)

[2.1.1. Схема взаємодії модулів 9](#_Toc9856002)

[Діaграма варіантів використання 10](#_Toc9856003)

[2.2. Технічний проект 11](#_Toc9856004)

[2.2.1. Контекстна діаграми системи 11](#_Toc9856005)

[2.2.2. DFD 1 рівня (для менеджера по роботі з постачальниками) 11](#_Toc9856006)

[2.2.3. DFD 1 рівня (для оператора АЗС) 12](#_Toc9856007)

[2.2.4. Словник даних контекстної діаграми 12](#_Toc9856008)

[2.2.5. Мініспецифікація процесів 14](#_Toc9856009)

[2.2.6. Таблиця відповідності потоків на діаграмах усіх рівнів 17](#_Toc9856010)

[2.2.7. Діаграма сутність-зв’язок (ERD) 19](#_Toc9856011)

[2.2.8. Логічна модель системи 19](#_Toc9856012)

[2.2.9. Діаграма класів 21](#_Toc9856013)

[2.2.10 . Діаграма послідовностей 23](#_Toc9856014)

[Додаток А. Технічне завдання 24](#_Toc9856015)

[Додаток Б. Інструкція користувача 41](#_Toc9856016)

[Додаток В. Код програмних модулів 49](#_Toc9856017)

# **Вступ**

Пам’ять – невід’ємна частина життя людини. Кожного для ми зустрічаємось з незліченною кількістю інформації, і з кожним роком її кількість тільки зростає. А людині, щоб бути успішною, потрібно постійно вчитися, постійно отримувати все нові і нові знання та навички. Але скільки людина дійсно запам’ятовує? А скільки того, що запам’ятовує вона забуває через день, місяць, рік?

Метою роботи є розробка сервісу для тренування пам’яті «Memory.pro», який призначений допомогти користувачам покращити свою пам’ять. А саме – тренування з запам’ятовування різних типів інформації (таких як числа, слова тощо), ведення статистики тренувань користування та рейтингу користувачів сервісу.

Для досягнення мети необхідно вирішити наступні завдання:

* Провести аналіз предметної області та вже існуючих аналогів програмного забезпечення.
* Розробити проект програмного забезпечення, який містить в собі:

1. Розробку, концептуальної, логічної, фізичної моделі бази даних.
2. Розробку специфікації програмних модулів.
3. Опис алгоритмів програми та розробку інтерфейсу користувача.
4. Опис СУБД та засобів програмної реалізації.
5. **Аналіз та опис предметної області**
   1. **Загальний опис предметної області**

Сервіс «Memory.pro» призначених для тренування та розвитку пам’яті. Основна функція сервісу – тренування з запам’ятовування даних різних типів, таких як цифри, слова, картинки, імена, дати, тощо. При виконанні курсової роботи будуть реалізовані два – цифри та слова.

Під час тренування користувачу необхідно запам’ятати максимальну кількість даних вибраного типу за мінімальний час, після чого відновити їх у тому ж порядку.

Після запам’ятовування формується результат тренування – що користувач запам’ятав правильно, що не правильно, скільки часу він запам’ятовував запропоновану інформацію тощо (детальніше у розділі «результати тренування»

На основі результатів користувачів формується статистика – статистика конкретного користувача та загальна статистика.

* 1. **Аналіз вже існуючого програмного забезпечення**
     1. memoryleague.com

Міжнародний сервіс для змагань із запам'ятовування Online Memory League Championship. Ним користуються багато кращих мнемоністів світу. Має всі базові типи тренувань, досить зручний інтерфейс.

Сервіс повністю англомовний. Є платним ($ 25 в рік, або $ 10 на три місяці). Сервісом можна користуватися безкоштовно, але тільки в демонстраційному режимі.

В першу чергу розрахований на швидкісне запам'ятовування невеликої кількості даних (наприклад, 80 цифр, 30 картинок). Має хорошу систему змагань (через цей сервіс проводиться один з чемпіонатів світу із запам'ятовування).

* + 1. memoryman.ru

Відносно новий сервіс по тренуванню пам'яті. Сервіс російськомовний. Містить всі основні типи тренувань, досить гнучке налаштування показу даних, кількості елементів. Має простий і досить зручний інтерфейс, мінімалістичний дизайн.

На даний момент найбільш популярний в російськомовному співтоваристві мнемоністів. Але має велику кількість помилок і недоробок.

* + 1. Powermemory.ru

Один з перших російськомовних сервісів по тренуванню пам'яті (в контексті мнемоніки). Надійний і стабільний, але містить тренування тільки із запам'ятовування чисел, карт і слів. Слабка можливість настройки тренування. З 2017 року сервіс закритий на оновлення.

* 1. **Дослідження та опис предметної області**

Основним користувачем сервісу є «Користувач» (або «User»). Він характеризується наступними атрибутами:

* Id
* Прізвище
* Ім’я
* Username
* Email
* Дата реєстрації

Користувачем може стати будь-який бажаючий (будь-якого віку, громадянства, національності). У кожного користувача повинен бути особистий кабінет (профіль), в якому він може бачити свої дані, а також може змінювати їх. Користувачі можуть бачити профілі інших користувачів.

Тренування

Тренування - основна функція сервісу. Тренування складається з наступних етапів:

1. Вибір тренування. На цьому етапі користувачу необхідно вибрати необхідний вид тренування і вказати кількість елементів інформації у тренуванні.
2. Запам'ятовування випадкових даних відповідно до типу тренування. Користувачеві показується випадкова послідовність даних зазначеного типу відповідно до встановлених на попередньому етапі параметрам. Користувач послідовно передивляється кожен елемент та запам’ятовує його.
3. Згадування. Після запам'ятовування користувачеві пропонується відтворити дані, які вдалося запам’ятати. Відтворити їх необхідно в тому ж порядку в якому вони були показані під час запам'ятовування.
4. Результати. Після того, як користувач закінчив етап згадування йому показуються результати тренування. Саме результати тренування зберігаються у базі даних і враховуються в особистій і загальній статистиці.

Типи тренувань

1. Числа – мета користувача запам’ятати задану послідовність цифр. У рамках курсової роботу користувачу будуть показуватись по 2 цифри. Таким чином набір можливих даних для запам’ятовування складається с чисел від «00» до «99».
2. Слова – мета користувача запам’ятати задану послідовність слів. Користувачу слова показуються по одному.

Набори даних для тренування у систему вводить адміністратор. У рамках курсової роботи фізично адміністратор реалізований не буде.

Результати тренувань

Результати тренування кожного типу зберігаються в базі даних у відповідних таблицях (для кожного типу тренування окрема таблиця). За результатами будується особиста статистика користувача і загальна статистика. Якщо користувач зробив більше 10% помилок – результат оцінюється в 0 балів.

Всі результати тренувань характеризуються такими атрибутами:

* Id користувача
* Дата тренування
* Час запам’ятовування
* Час згадування
* Оцінка

Результат тренування оцінюється за формулою:

Де correct - кількість правильних відповідей,

total - загальна кількість даних,

time - час запам'ятовування в секундах.

Якщо користувач зробив більше 10% помилок, результат оцінюється у 0 балів.

Статистика

Існує два види статистики – статистика користувача та загальна статистика.

У статистиці користувача відображається список усіх його тренувань за певний період та по кожному виду тренувань.

У загальній статистиці відображаються найкращі результати користувачів по кожному виду тренування.

* 1. **Вибір засобів розробки**

Для розробки програмного забезпечення у цій роботі використано мову програмування Java. Для розробки графічного інтерфейсу користувача використано бібліотеку JavaFX з використанням FXML та CSS (SASS) а також Java Persistence API framework EclipseLink та СУБД MYSQL.

1. **Проектування**
   1. **Ескізний проект** 
      1. **Схема взаємодії модулів**

Взаємодію модулів системи можна представити в вигляді наступної діаграми:



**Діаграма варіантів використання**

За допомогою діаграми варіантів використання окреслимо користувачів системи та її функціонал:



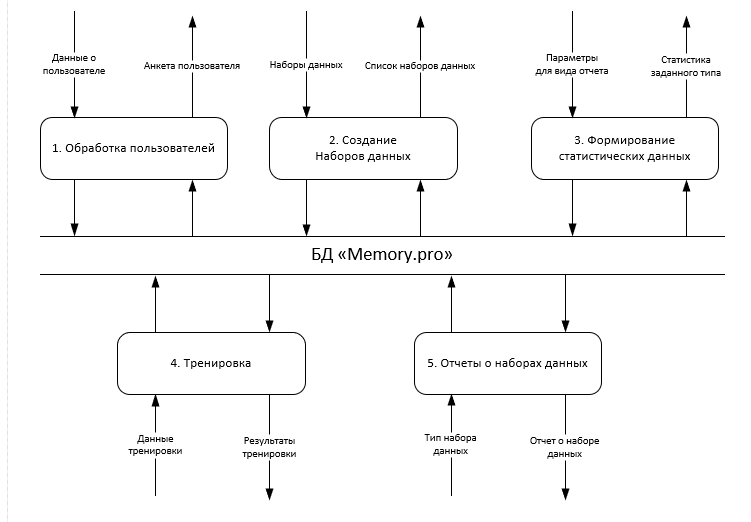
* 1. **Технічний проект**

В результаті опису предметної області та постановки задачі отримано вимоги до БД системи, які можна зобразити такими діаграмами як: DFD, ERD, логічна модель системи.

* + 1. **Контекстна діаграми системи**



* + 1. **DFD 1 рівня**



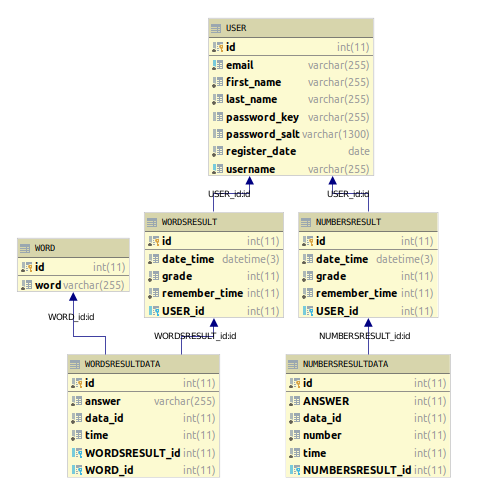
* + 1. **DFD 2 рівня (Статистика)**



* + 1. **Діаграма сутність-зв’язок (ERD)**



* + 1. **Логічна модель системи**



Таблиця User – користувач системи. Усі атрибути зрозумілі з назви. Атрибути password\_key та password\_salt необхідні для авторизації користувача (сам пароль ніде не зберігається, його знає лише користувач). Тобто зберігання паролю користувача основане на механізмі хешування.

Таблиці WordsResult та NumbersResult призначенні для зберігання результатів тренування.

date\_time – час початку тренування

grade – оцінка тренування

remember\_time – час згадування в секундах

Таблиці WordsResultData NumbersResultData зберігають дані тренування. Відповідно слова, та числа. А також час на запам’ятовування кожної одиниці даних у мілісекундах та відповідь користувача (у WordsResultData зберігається як текст, так як, користувач може ввести слово, якого немає в базі даних).

* + 1. **Діаграма класів**

На діаграмі класів зображено взаємодію модулів системи.

У системі усі класи поділяються на такі види:

1. Component – певна візуально та логічно самостійна сутність. Усі компоненті складаються с контролеру (java класу, FXML, та SASS для стилів). Компоненти реалізують лише візуальну складову. Вони не повинні реалізовувати занадто багато логіки, не пов’язаної напряму з відображенням. Назви усіх класів цього виду закінчуються на Component
2. Screen – наслідує від Component. Представляє сутність вікна (сторінки). Screen класи можуть містити у собі компоненти. Один вид компоненту може міститись в різних Screen класах. Назви усіх класів цього виду закінчуються на Screen.
3. Service – класи, що реалізують логічно самостійну сутність. Дані класи реалізують певну задачу (наприклад авторизацію) та надають зручний інтерфейс для роботи з ними. Вони нічого не знають про те, як результати їх роботи будуть представлені користувачу. Дані класи працюють тільки з даними. Назви усіх класів цього виду закінчуються на Service.
4. Entity – описання сутностей для Java Persictence API
5. Dao – класи, що узагальнюють роботу з Entity класами та базою даних в цілому. Тільки в Dao-класах дозволяється працювати з базою даних. Назви усіх класів цього виду закінчуються на Dao.

Кожен вид класів при необхідності може використовувати додаткові класи

Також є два допоміжних класів:

* 1. Common – клас, що дозволяє Screen- та Component-класам спілкуватись між собою, та передавати повідомлення до Bootstrap.
  2. Alerts – надає зручний інтерфейс для виведення повідомлень користувачам (повідомлення про помилки, попередження, інформаційні повідомлення тощо). Даний клас міститься (композиція) у Component і таким чином доступний з усіх Component та Screen класів.

Component класи:

1. HeaderComponent – header, що відображається на усіх сторінках
2. LoginComponent – авторизація користувачів
3. SignupComponent – реєстрація нових користувачів
4. TrainingSetupComponent – вибір параметрів тренування
5. TrainingMemorizeComponent – етап запам’ятовування
6. TrainingRememberComponent – етап згадування
7. ResultComponent – відображає результат тренування за переданими типом тренування на id результату.

Screen класи:

1. Bootstrap – точка входу, головний контролер
2. AuthScreen – сторінка реєстрації та авторизації
3. IndexScreen – головна сторінка
4. ProfileScreen – профіль користувача
5. StatisticScreen – сторінка статистики
6. TrainingScreen – сторінка тренування

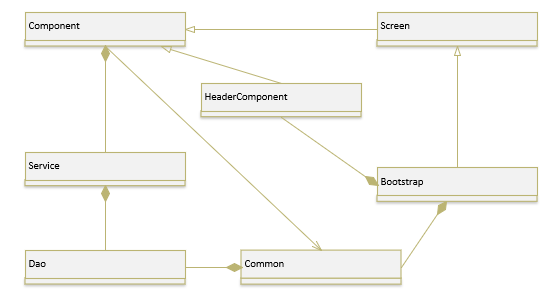
Service класи

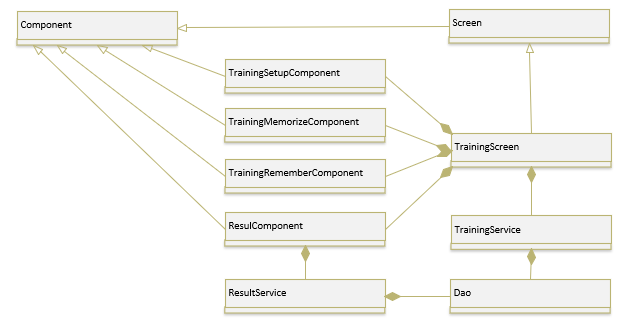
1. AuthService – сервіс авторизації на реєстрації
2. TrainingService – сервіс тренувань (надає дані для тренування, обробляє та зберігає результати.
3. ResultService – сервіс для роботи з результатами тренування (за переданим типом тренування та id результата надає оброблений і готовий для подальшого використання результат тренування (в поверненому результаті все містяться все дані – час запам’ятовування, мінімальний / максимальний час на запам’ятовування одного елементу тощо).

Entity та Dao класи (відповідають сутностям на ER діаграмі

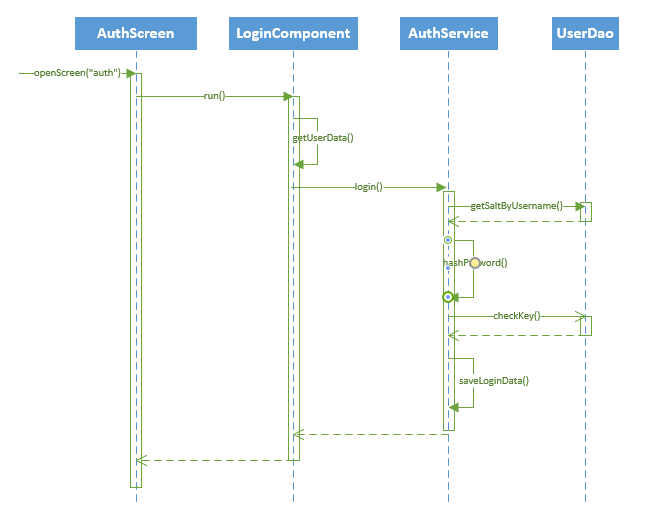
1. User та UserDao
2. NumbersResult та NumbersResultDao
3. NumbersResultData та NumbersResultDataDao
4. WordsResult та WordsResult
5. WordsResultData та WordsResultDataDao
6. Word та WordDao

Так, як сервіс складається з великої кількості класів і при цьому все вони діляться всього на кілька типів, то на діаграмі класів немає сенсу зображати всі класи. Тому буде представлена загальна діаграма класів та діаграма класів відносно TrainingScreen.





* + 1. **. Діаграма послідовностей** 
       1. **Діаграма послідовностей процесу «авторизація»**



* + - 1. **Діаграма послідовностей процесу «тренування»**



# **Додаток А. Технічне завдання**

1. Загальні відомості
   1. Найменування системи
      1. Повна назва системи

Автоматизація роботи автозаправної станції (АЗС).

* + 1. Коротка назва

Управління АЗС, система

* 1. Підстави для розробки

Підставою для розробки програмного забезпечення (ПО) є договір між Беркунським Євгеніем Юрійовичем (далі Замовник) та Кольцовим Андрієм В’ячеславовичем (далі Виконавец) від 08.02.2019.

* 1. Планові строки початку та закінчення робіт

Початком роботи над Пз слід вважати 10.02.2019, закінчення роботи є 26.05.2019.

* 1. Порядок оформлення та пред’явлення замовнику результатів робіт

Роботи зі створення ПЗ здаються Замовнику поєтапно згідно з календрним планом проекта. З закінчення кожного з етапів робіт Виконавец здає замовнику відповідні звітні документи етапа, склад яких зазначено у договорі.

1. Призначення та цілі створення системи
   1. Призначення системи

ПО призначене для автоматизації роботи менджера по роботі з постачальниками мережі АЗС та автоматизації роботи оператора АЗС.

У рамках проекту автоматизується наступна діяльність менеджера по роботі з постачальниками:

1. Внесення, видалення, пошук, перегляд записів про пальне, постачальників до Бази даних (БД).
2. Створення форм звітності на основі даних про пальне та постачальників, що зберігаються у БД.

У рамках проекту автоматизується наступна діяльність операто АЗС:

1. Внесення, видалення, пошук, перегляд записів про клієнтів АЗС.
2. Створення форм звітності на основі даних про клієнтів АЗС, записи про яких зберігаються у БД.
   1. Цілі створення системи

Система створюється з наступними цілями:

1. Перенесення документообігу у електрону форму.
2. Полегшення роботи персоналу АЗС з документообігом.
3. Зберігання даних, що потрібні для функціонування АЗС у електроному вигляді у БД.

У результаті створення системи та перенесення документообігу у електроний формат мають покращітись показники часу обробки даних та підготовки форм звітності.

1. Характеристика об’єктів автоматизації

Дана система призначена для автоматизації роботи менеджера по роботі з постачальниками та оператора мережі стаціонарних автозаправних комплексів.

Менеджер по роботі з постачальниками відповідальний за внесення даних про пальне та постачальників у БД системи та роботу з формами звітності. Вхідні дані він будет отримувати з прихідних накладних та контрактів з постачальниками. Менеджер по роботі з постачальниками буде стоворювати наступні форми звітності:

* Звіт про залишок пального;
* Звіт про оберт (переміщення) пального за певний період;
* Звіт про постачальників;
* Видаткова накладна.

Оператор АЗС відповідальний за створення облікових записів програми лояльності, роботу з формами звітності програми лояльності. Дані для створення облкового запису програми лояльності оператор АЗС будет отримувати безпосередньо від клієнтів АЗС. У випадку успішного створення облікового запису у програмі лояльності АЗС, клієнт отримуватиме картку програми лояльності, що надає право на знижку при покупці пального відповідно до умов програми лояльності. Відповідно оператор АЗС будет створюват наступні форми звітності:

* Звіт про клієнтів, що є учасниками програми лояльності;
* Звіт по карткам програми лояльності.

1. Вимоги до системи
   1. Вимоги до системи цілому
      1. Вимоги до структури та функціонування системи

Система має бути централізованою, тобто усі дані мають знаходитись в одному центральному сховищі.

У системі можна виділити наступні функціональні підсистеми:

1. Підсистема, що відповідає за отримання даних від користувачів.
2. Підсистема, що відповідає за зберігання даних.
3. Підсистема формування та візуалізації звітності.

Джерелами даних для система можуть бути дані від клієнтів АЗС (анкета клієента), дані про пальне (прихідна накладна), дані про постачальника (контракт).

* + 1. Вимоги до чисельності та квалійікації персоналу системи и режиму його роботи
       1. Вимоги до чисельності персоналу

До складу персоналу, що буде працювати з системою буде входити менеджер по роботі з постачальниками та оператори АЗС. Кількість персоналу має регулюватись на кожній з АЗС індивідуально в залежності від ії розміру і т.д.

* + - 1. Вимоги до кваліфікації персоналу

До кваліфікації персоналу, що експлуатує систему виявлені наступні вимоги:

1. Знання відповідної пердметнох області, знання та навички роботи з аналітичними додатками.
2. Ознайомлення з інструкцією користувача.
   * + 1. Вимоги до режиму роботи персоналу

Вимоги до режиму роботи персоналу встановлюються відповідно до штатного розпису АЗС.

* + 1. Вимоги до надійності
       1. Склад показників надійності для системи в цілому

Рівень найдійності повинен досягатись узгодженим застосуванням організаційних, організаційно-технічних заходів та програмно-апаратних засобів. Надійність повина забезпечуватись за рахунок:

1. Застосування технічних засобів, програмного забезпечення, що відповідають класу вирішуваних задач.
2. Своечасне виконання процесів адміністрування та технічного обсуговування програмно-апаратних засобів.
3. Попередне навчання користувачів та обслуговуючого персоналу.
   * + 1. Перелік аварійних ситуацій, за яких регламентуються вимоги до надійности

Під аварійною ситацією слід розуміти аварійне завершеня процесу, що виконується одноєю з підсистем, а також «зависання» процесаю.

При роботі системи можливі наступні аварійні ситуації, що впливають на надійність роботи системи:

1. Сбій у роботі електропостачання серверу БД;
2. Сбій у роботі електропостачання АЗС;
3. Помилки роботи ПЗ, що не були виявлені про відлаці та випробуванні системи.
   * + 1. Вимоги до технічних засобів та програмного забезпечення:

До надійності обладнання висуваються наступні вимоги:

1. У якості апаратних платформ мають використовуватись засоби з підвищеною надійністю.
2. Використання технічних засобів, що відповідають класу задач, що вирішуються.

До надійності електропостачання висуваються наступні вимоги:

1. Для підвищення здатності протистояти збоям в цілому потрібна обовз’язкова можливість роботи системи від джерел безперебійного живленн;
2. Необхідно забезпечувати безперебійне живлення активного мережевого обладнання.

Надійність апаратних та програмних засобів мажє забезпечуватись за рахунок настпних організаційних заходів:

1. Попереднього навчання користувачів та обслуговуючого персоналу;
2. Своєчасного виконання процесів адміністрування;
3. Дотримання правил експлуатації та технічного обслуговування програмно-апаратних засобів;
4. Своечасного виконання процедури резервного копіювання даних;

Надійність прогамного забезпечення повина забезпечуватись за рахунок:

1. Належності розроблюваного ПЗ в цілому;
2. Проведенням комплексу відладки, пошуку та виправлення помилок;
3. Веденням журналу системних повідомлень та помилок по підсистемам для наступного аналізу та виправлення конфігурації.
   * + 1. Вимоги до методів оцінки та контролю показників надійності на різних етапах створення системи у відповідності з діючими нормативно-технічними документами

Перевірку виконання вимог з надійності повинна виконуватись на етапі проектування системи розрахунковим шляхом, а на етапах випробування та експлуатації - за методикою Розробника, погодженою з Замовником.

* + 1. Вимоги до ергономіки до технічної естетики

Підсистема для введення даних про пальне, клієнтів, постачальників, формування звітності має забезпечувати зручний для кінцевого користувача інтерфейс, що відповідає наступним вимогам:

В частіни зовнішнього оформлення:

1. Інтерфейси підсистем мають бути типізованими;
2. Має бути забезпечено підтримку української мови локалізації.

В частині діалогу з користувачем:

1. При виникненні помилок у роботі підсистеми на екрано має виводитись повідомлення з наименуванням помилки та рекомендаціями щодо її усунення.
   * 1. Вимоги до експлуатації, технічного обслуговування, ремонту та зберіганню компаненів системи

Умови експлуатації, а також види і періодичність обслуговування технічних засобів системи повинні відповідати вимогам по експлуатації, технічного обслуговування, ремонту і зберігання, викладеним в документації заводу-виготовлювача (виробника) на них. Технічні засоби системи і персонал повинні розміщуватися в існуючих приміщеннях Замовника, які за кліматичними умовами повинні відповідати ГОСТ 15150-69 «Машини, прилади та інші технічні вироби. Виконання для різних кліматичних районів. Категорії, умови експлуатації, зберігання і транспортування в частині впливу кліматичних факторів зовнішнього середовища» (температура навколишнього повітря від 5 до 40 ° С, відносна вологість від 40 до 80% при Т = 25 ° С, атмосферний тиск від 630 до 800 мм ртутного стовпа ). Розміщення технічних засобів і організація автоматизованих робочих місць повинні бути виконані відповідно до вимог ГОСТ 21958-76 «Система" “Людина-машина”. Зал і кабіни операторів. Взаємне розташування робочих місць. Загальні ергономічні вимоги». Для електроживлення технічних засобів повинна бути передбачена трифазна чотирипровідна мережу з глухо заземленою нейтраллю 380/220 В (+ 10-15)% частотою 50 Гц (+ 1-1) Гц. Кожне технічне засіб живиться однофазним напругою 220 В частотою 50 Гц через мережеві розетки з заземлюючим контактом. Для забезпечення виконання вимог по надійності повинен бути створений комплект запасних виробів і приладів (ЗІП).Склад, місце та умови зберігання ЗІП визначаються на етапі технічного проектування.

* + 1. Вимоги до захисту інформації від несанкціонованого доступу
       1. Вимоги до інформаційної безпеки

Забезпечення інформаційне безпеки Системи має відповідати таким вимогам:

1. Захист Системи повинен забезпечуватися комплексом програмно-технічних засобів і організаційних заходів, що іх підтримують
2. Захист Системи повинен забезпечуватися на всіх технологічних етапах обробки інформації і в усіх режимах функціонування, в тому числі при проведенні ремонтних та регламентних робіт.
3. Програмно-технічні засоби захисту не повинні істотно погіршувати основні функціональні характеристики Системи (надійність, швидкодію, можливість зміни конфігурації).
4. Розмежування прав доступу користувачів і адміністраторів Системи повинно будуватися за принципом "що не дозволено, то заборонено".
   * + 1. Вимоги до антивірусного захисту

Засоби антивірусного захисту повинні бути встановлені на всіх робочих місцях користувачів і адміністраторів Системи. Засоби антивірусного захисту робочих місцях користувачів і адміністраторів повинні забезпечувати:

1. централізоване управління скануванням, видаленням вірусів і протоколированием вірусної активності на робочих місцях користувачів;
2. централізовану автоматичну інсталяцію клієнтського ПЗ на робочих місцях користувачів і адміністраторів;
3. централізоване автоматичне оновлення вірусних сигнатур на робочих місцях користувачів і адміністраторів;
4. ведення журналів вірусної активності;
5. адміністрування всіх антивірусних продуктів.
   * 1. Вимоги щодо збереження інформації при аваріях

Ймовірними аварійними ситуаціями слід вважата збоії електропостачання на сервере БД, сбої електропостачання на АЗС. Для подібних випадків система має забезпечувати резервне копіювання.

* + 1. Вимиоги до захисту від впливу зовнішніх чинників

Стосовно до програмно-апаратного оточення Системи ставляться такі вимоги до захисту від впливу зовнішніх впливів. Вимоги до радіоелектронної захисту:

1. електромагнітне випромінювання радіодіапазону, що виникає при роботі електропобутових приладів, електричних машин і установок, пріёмопередающіх пристроїв, що експлуатуються на місці розміщення АПК Системи, не повинні призводити до порушень працездатності підсистем.Вимоги по стійкості, стійкості і міцності до зовнішніх впливів:- Система повинна мати можливість функціонування при коливаннях напруги електроживлення в межах від 155 до 265 В (220 ± 20% - 30%);
2. Система повинна мати можливість функціонування в діапазоні допустимих температур навколишнього середовища, встановлених виробником апаратних засобів;
3. Система повинна мати можливість функціонування в діапазоні допустимих значень вологості навколишнього середовища, встановлених виробником апаратних засобів;
4. Система повинна мати можливість функціонування в діапазоні допустимих значень вібрації, встановлених виробником апартних засобів.
   * 1. Вимоги до стандартизації та унифікації

У вимоги до стандартизації та уніфікації включають: показники, що встановлюють необхідний ступінь використання стандартних, уніфікованих методів реалізації функцій (завдань) системи, що поставляються програмних засобів, типових математичних методів і моделей, типових проектних рішень, уніфікованих форм управлінських документів, встановлених ДСТУ 6.10.1, загальносоюзних класифікаторів техніко-економічної інформації та класифікаторів інших категорій відповідно до галузі їх застосування, вимоги до використання типових а втоматізірованних робочих місць, компонентів і комплексів.

* + 1. Додаткові вимоги

Система має розроблятися і експлуатуватися на вже наявному у Замовника апаратно-технічному комплексі. Необхідно створити окремі самостійні зони розробки та тестування Системи. Для зони розробки і тестування повинні використовуватися ті ж програмні засоби, що і для зони промислової експлуатації

* + 1. Вимоги до безпеки

При впровадженні, експлуатації та обслуговуванні технічних засобів системи повинні виконуватися заходи електробезпеки відповідно до «Правил улаштування електроустановок» і «Правилами техніки безпеки при експлуатації електроустановок споживачів».Апаратне забезпечення системи повинно відповідати вимогам пожежної безпеки в виробничих приміщеннях по ДСТУ 12.1.004-91. «ССБТ. Пожежна безпека. Загальні вимоги".Повинно бути забезпечено дотримання загальних вимог безпеки відповідно до ДСТУ 12.2.003-91. «ССБТ. Оборудование производственное. Загальні вимоги безпеки »при обслуговуванні системи в процесі експлуатації.Апаратна частина системи повинна бути заземлена відповідно до вимог ДСТУ Р 50571.22-2000. «Електроустановки будівель. Частина 7. Вимоги до спеціальних електроустановок. Розділ 707. Заземлення обладнання обробки інформації ».Значення еквівалентного рівня акустичного шуму, створюваного апаратурою системи, має відповідати ДСТУ 21552-84 «Засоби обчислювальної техніки. Загальні технічні вимоги, приймання, методи випробувань, маркування, упаковка, транспортування і зберігання », але не перевищувати наступних величин:

50 дБ - при роботі технологічного обладнання та засобів обчислювальної техніки без друкувального пристрою;

- 60 дБ - при роботі технологічного обладнання та засобів обчислювальної техніки з друкуючим пристроєм.

* 1. Вимоги до видів забезпечення
     1. Вимоги до математичного забезпечення

Не виявляються

* + 1. Вимоги до інформаційного забезпечення
       1. Вимоги до складу, стуктурі та способам організації даних у системі

Структура зберігання даних в КХД повинна складатися з наступних основних областей:

1. область тимчасового зберігання даних;
2. постійного зберігання даних;
   * + 1. Вимоги до інформаційного обміну між компонентами системи

Інформаційний обмін між компонентами системи має бути реалізований наступним чином:



* + - 1. Вимоги до інформаційної сумісності зі суміжними системами

Система не повинна бути закритою для суміжних систем і повинна підтримувати можливість експорту даних в суміжні системи через інтерфейсні таблиці або файли даних.Система повинна забезпечити можливість завантаження даних, одержуваних від суміжної системи.

* + - 1. Вимоги щодо використання класифікаторів та уніфікованих документів

Система, за можливості, повинна використовувати класифікатори і довідники, які ведуться в системах-джерелах даних. Основні класифікатори і довідники в системі повинні бути єдиними. Значення класифікаторів і довідників, відсутні в системах-джерелах, але необхідні для аналізу даних, необхідно підтримувати в спеціально розроблених файлах або репозиторії бази даних.

* + - 1. Вимоги щодо застосування систем управління баз даних (СУБД)

Для реалізації підсистеми зберігання даних необхідно використовувати СУБД MariDB.

* + - 1. Вимоги до структури процеса збору, обробки, передачі даних у системі та предсставленню даних

Процес збору, обробки і передачі даних в системі визначається регламентом процесів збору, перетворення і завантаження даних, що розробляється на етапі «Проектування. Розробка ескізного проекту. Розробка технічного проекту ».

* + - 1. Вимоги до захисту даних від руйнування при аваріях та сбоях електропостачання системи

Інформація в базі даних системи повинна зберігатися при виникненні аварійних ситуацій, пов'язаних зі збоями електроживлення.

Система повинна мати безперебійне електроживлення, що забезпечує її нормальне функціонування протягом 15 хвилин у разі відсутності зовнішнього енергопостачання, і 5 хвилин додатково для коректного завершення всіх процесів.

Резервне копіювання даних повинне здійснюватися на регулярній основі, в обсягах, достатніх для відновлення інформації в підсистемі зберігання даних.

* + - 1. Вимоги до контролю, зберігання, оновлення та відновлення даних

До контролю даних висуваються такі вимоги:

1. система повинна протоколювати всі події, пов'язані зі зміною свого інформаційного наповнення, і мати можливість в разі збою в роботі відновлювати свій стан, використовуючи раніше запротокольовані зміни даних.

До зберігання даних висуваються такі вимоги:

1. зберігання історичних даних в системі повинно проводитися не більше ніж за 5 (п'ять) попередніх років. По закінченню цього терміну дані повинні переходити в архів;
2. історичні дані, що перевищують п'ятирічний поріг, повинні зберігатися на стрічковому масиві з можливістю їх відновлення.

До оновлення і відновлення даних висуваються такі вимоги:

1. для сервера збору, обробки і завантаження даних необхідно забезпечити резервне копіювання його бінарних файлів (Home) раз в 2 тижні і зберігання копії протягом 2-х місяців;
2. для сервера бази даних необхідно забезпечити резервне копіювання його бінарних файлів раз в 2 тижні і зберігання копії протягом 2-х місяців;- для даних сховища даних необхідно забезпечити резервне копіювання і архівацію на стрічковий масив з наступним розподілом часу:

* холодна копія - щоквартально;
* логічна копія - щомісяця (кінець місяця);
* інкрементальное резервне копіювання - щотижня (неділя);
* архівірованіе - щоквартально;
  + - 1. Вимоги до процедури надання юридичної сили документам, продукуються технічними засобами системи

Вимоги не пред'являються.

* + 1. Вимоги до лінгвістичного забезпечення

При реалізації системи повинні застосовуватися такі мови високого рівня: SQL, Java.

Для організації ділогу системи з користувачем необхідно використовувати графічний віконний інтерфейс користувача.

* + 1. Вимоги до ПЗ

До забезпечення якості ПЗ ставляться такі вимоги:-

1. функціональність повинна забезпечуватися виконанням підсистемами всіх їх функцій.
2. надійність повинна забезпечуватися за рахунок попередження помилок - недопущення помилок в готових ПЗ;
3. легкість застосування повинна забезпечуватися за рахунок застосування покупних програмних засобів
4. - ефективність повинна забезпечуватися за рахунок прийняття відповідних, вірних рішень на різних етапах розробки ПЗ і системи в цілому;
5. супроводжуваність повинна забезпечуватися за рахунок високої якості документації по супроводу, а також за рахунок використання в програмному тексті опису об'єктів і коментарів; використанням осмислених (мнемонічних) і стійко помітних імен об'єктів; розміщенням не більш одного оператора в рядку тексту програми; униканням створення фрагментів текстів програм з неочевидним або прихованим змістом.
6. також на кожному етапі в розробці ПЗ повинна проводиться перевірка правильності прийнятих рішень щодо розробки та застосування готових ПЗ.
   * 1. Вимоги до організаційного забезпечення

Основними користувачами системи є працівнки функціонального підрозділу Замовника. Забезпечує експлуатацію Системи підрозділ інформаційних технологій Замовника.Склад співробітників кожного з підрозділів визначається штатним розписом Замовника, який, в разі необхідності, може змінюватися.

До захисту від помилкових дій персоналу ставляться такі вимоги:

1. повинна бути передбачена система підтвердження легітимності користувача при перегляді даних;
2. для всіх користувачів повинна бути заборонена можливість видалення переднастроєних об'єктів і звітності;
3. для зниження помилкових дій користувачів має бути розроблено повне і доступне керівництво користувача.
   * 1. Вимоги до методичного забезпечення

Порядок роботи з з системою викладено у окремому документі – «Інструкції користувача».

* + 1. Вимоги до патентної чистоти

Не висуваються.

1. Склад робіт по створенню системи

Роботи зі створення системи виконуються в три етапи:

Проектування. Розробка ескізного проекту. Розробка технічного проекту (тривалість – 08.02.2019 – 20.03.2019).

Розробка робочої документації. Адаптація програм (тривалість – 20.03.2019 – 02.05.2019).

Введення в дію (тривалість – 02.05.2019 – 25.05.2019).

Конкретні терміни виконання стадій і етапів розробки і створення Системи визначаються Планом виконання робіт, який є невід'ємною частиною Договору на виконання робіт по справжньому Приватному технічним завданням.

Перелік організацій - виконавців робіт, визначення відповідальних за проведення цих робіт організацій визначаються Договором.

1. Порядок котролю та прийому системи
   1. Об’єм та види випробувань системи

Система піддається випробуванням наступних видів:

1. Попередні випробування.
2. Дослідна експлуатація.
3. Приймальні випробування.

Склад, обсяг та методи випробувань системи визначаються документом «Програма і методика випробувань», який розроблюється на стадії «Робоча документація».

Склад, обсяг та методи дослідної експлуатації системи визначаються документом «Програма дослідної експлуатації», який розробляють на стадії «Введення в дію».

Склад, обсяг та методи приймальних випробувань системи визначаються документом «Програма і методика випробувань», який розробляють на стадії «Введення в дію» з урахуванням результатів проведення попередніх випробувань і дослідної експлуатації.

1. Вимоги до складу та змісту робіт з підготовки об'єкта автоматизації до введення системи в дію
   1. Технічні заходи

Силами Замовника в термін до початку етапу «Розробка робочої документації. Адаптація програм» повинні бути виконані наступні роботи:

1. здійснена підготовка приміщення для розміщення АТК системи відповідно до вимог, наведених у цьому технічному завданні;
2. здійснено закупівлю та встановлення необхідного АТК;
3. организавать необхідне мережеве взаємодія.
   1. Організаційні заходи

Силами Замовника в термін до початку етапу робіт «Розробка робочої документації. Адаптація програм» повинні бути вирішені організаційні питання щодо взаємодії з системами-джерелами даних. До даних організаційних питань відносяться:

1. організація доступу до баз даних джерел;
2. визначення регламенту інформування про зміни структур систем-джерел;
3. виділення відповідальних фахівців з боку Замовника для взаємодії з проектною командою з питань взаємодії з системами-джерелами даних.
   1. Зміни в інформаційному забезпеченні

Для організації інформаційного забезпечення системи повинен бути розроблений і затверджений регламент підготовки та публікації даних з систем-джерел. Перелік регламентів може бути змінений на стадії «Розробка робочої документації. Адаптація програм».

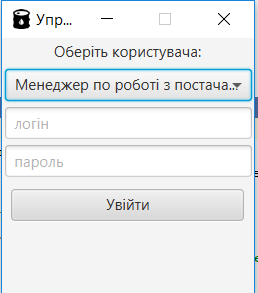
1. Вимоги до документування

|  |  |
| --- | --- |
| Проектування. Розробка ескізного проекта. Розробка технічного проекта. | Відомість ескізного проекта |
| Пояснювальна записка ескізного проекта |
| Відомість ескізного проекта |
| Пояснювальна записка до технічного проекта |
| Схема функціональної структури |
| Відомість експлуатаціоних документів |
| Паспорт |
| Загальне призначення системи |
| Технологічна інструкція |
| Інструкція користувача |
| Розробка технічної документації. Адаптація програм | Опис технологічного процесу обробки даних |
| Інструкція по формуванню та введеню бази даних |
| Склад вхідних даних |
| Каталог бази даних |
| Програма та методика випробувань |
| Специфікація |
|  | Опис програм |
| Текст програм |
| Введення в дію | Акт прийому в дослідну експлуатацію |
| Протокол випробувань |
| Акт прийому системи в експлуатацію |
| Акт закінчення робіт |
|  |

# **Додаток Б. Інструкція користувача**

Дана інструкція призничена для ознайомлення з функціями ПЗ та полегшення роботи з ним.

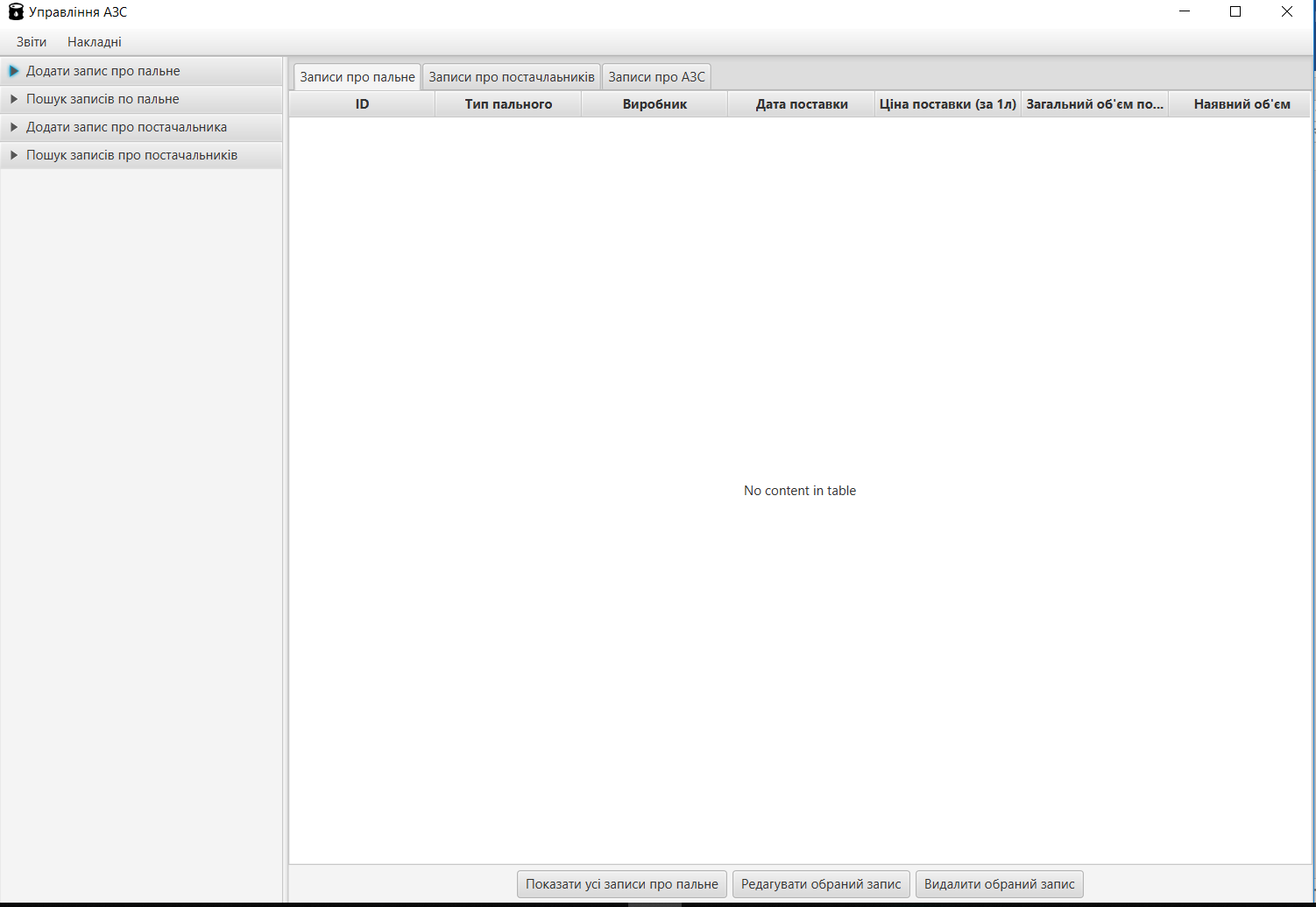
1. Для запуску програми потрібно натиснути на якрлик з написом «Управління АЗС» або відкрити його будь-яким іншим чином.
2. Після запуску з’явиться вікно в якому потрібно буде вибрати користувача (менеджера по роботі з постачальниками або оператора АЗС) та ввести свії логін та пароль. У випадку введення правильного логіну та паролю буде відкрите відповідне АРМ, інакше користувач побачить повідомлення про помилку.



1. Детальна інструкція для менеджера по робота з постачальниками

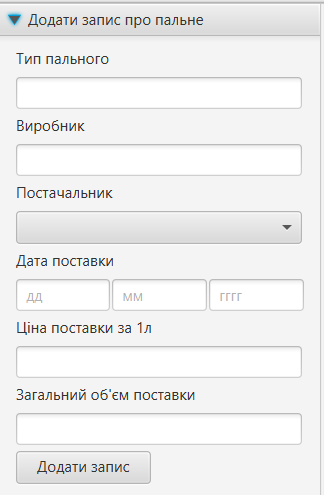
Після відкриття АРМ будуть доступні наступні функції:

* Додати запис про пальне;
* Знайти запис пального;
* Додати запис про постачальника;
* Знайти запис про постачальника;
* Переглянути усі записи про пальне;
* Видалити обраний запис (про пальне);
* Редагувати обраний запис (про пальне);
* Переглянути усі записи про постачальників;
* Видалити обраний запис (про постачальника);
* Редагувати обраний запис (про постачальника);
* Переглянути усі записи про АЗС;
* Видалити обраний запи (про АЗС);
* Редагувати обраний запис про АЗС;
* Створити звіт про постачальників;
* Створити звіт про пальне;
* Створити звіт про оберт (переміщення) пального;
* Створити видаткову накладну.



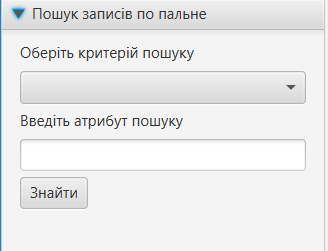
* 1. Додати запис про пальне

Для того, аби додати запис про пальне необхідно натиснути на напис «Додати запис про пальне». Після цього з’являться поля для введення даних про тип пального, виробника пального, постачальника (яко потрібно вибрати зі списку), дату поставки, ціну за 1л, загальний об’єм поставки. Після того, як усі поля будут заповнені необхідно натиснути на кнопку «Додати запис». У випадку заповнення усіх полів правильними даними запис про пальне буде додано до БД системи, інакше буду виведено повідомлення про помилку з поясненям помилки.



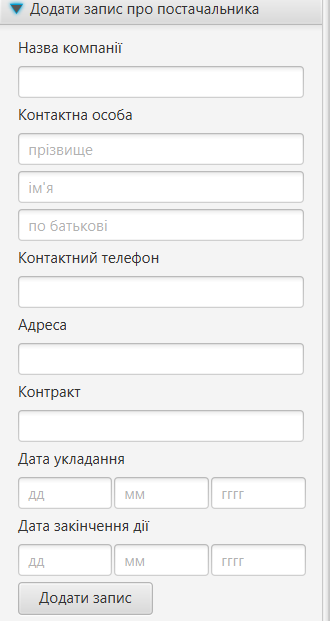
* 1. Знайти запис пального

Для того, аби додати знайти про пальне необхідно натиснути на напис «Пошук записів про пальне». Після цього з’являться випадаюче меню для вибору критерія пошуку поля для введення даних для введеня цього критерію. Після того, як буде обрано критерій пошуку та введено атрибут пошуку у поле необхідно натиснути на кнопку «Знайти». У випадку вибору критерія пошуку з меню та заповнення поля відповідними даними запис про зайндено (або виведено повідомлення, що такого запису немає у БД), інакше буду виведено повідомлення про помилку з поясненям помилки.



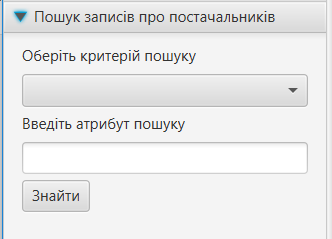
* 1. Додати запис про постачальника

Для того, аби додати запис про пальне необхідно натиснути на напис «Додати запис про постачальника». Після цього з’являться поля для введення даних про назву компанії, ПІБ контактної особи, контактний телефон, адресу, контракт, дата укладання та дату закінчення дії контракту. Після того, як усі поля будут заповнені необхідно натиснути на кнопку «Додати запис». У випадку заповнення усіх полів правильними даними запис про пальне буде додано до БД системи, інакше буду виведено повідомлення про помилку з поясненям помилки.



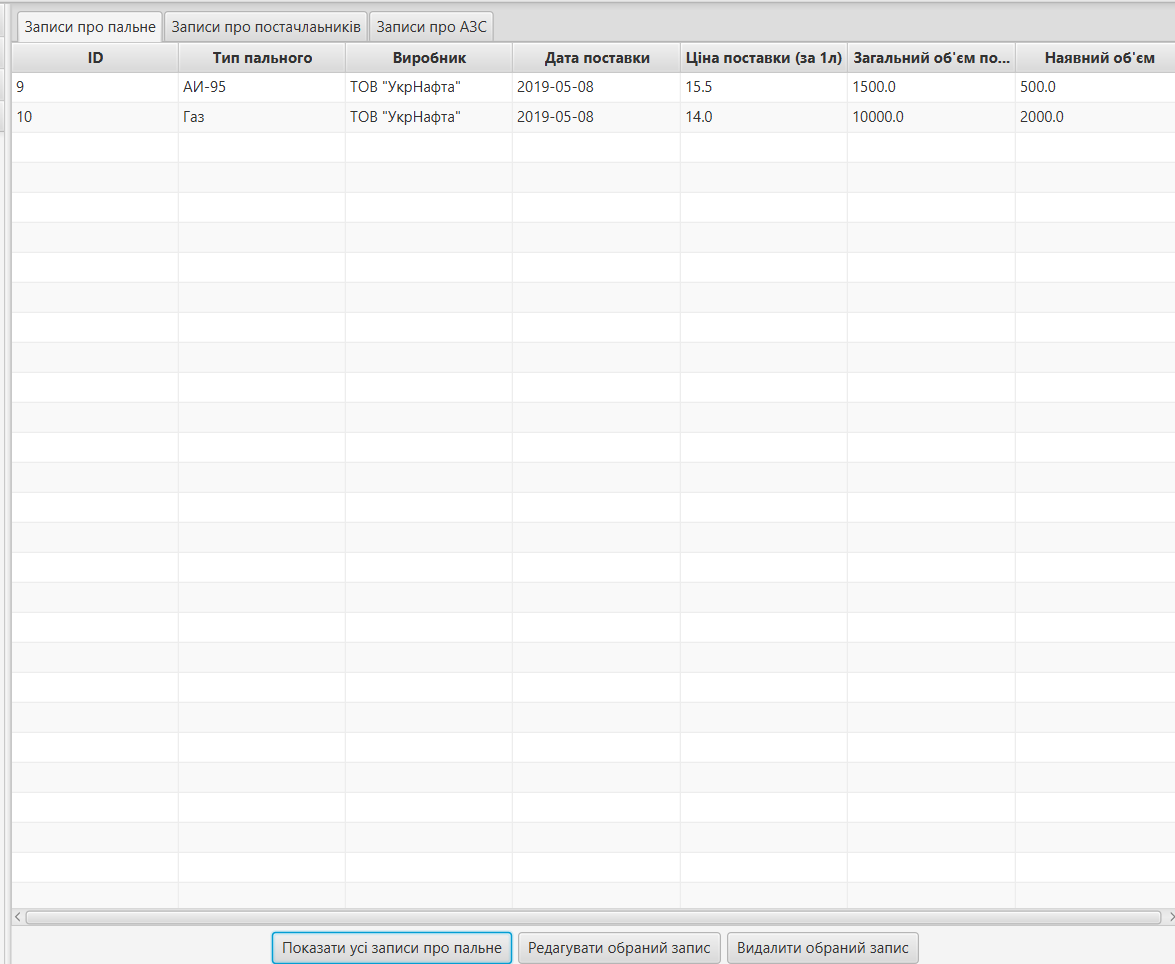
* 1. Пошук записів про постачальника

Для того, аби додати знайти про пальне необхідно натиснути на напис «Пошук записів про постачальників». Після цього з’являться випадаюче меню для вибору критерія пошуку поля для введення даних для введеня цього критерію. Після того, як буде обрано критерій пошуку та введено атрибут пошуку у поле необхідно натиснути на кнопку «Знайти». У випадку вибору критерія пошуку з меню та заповнення поля відповідними даними запис про знайдено(або виведено повідомлення, що такого запису немає у БД), інакше буду виведено повідомлення про помилку з поясненям помилки.



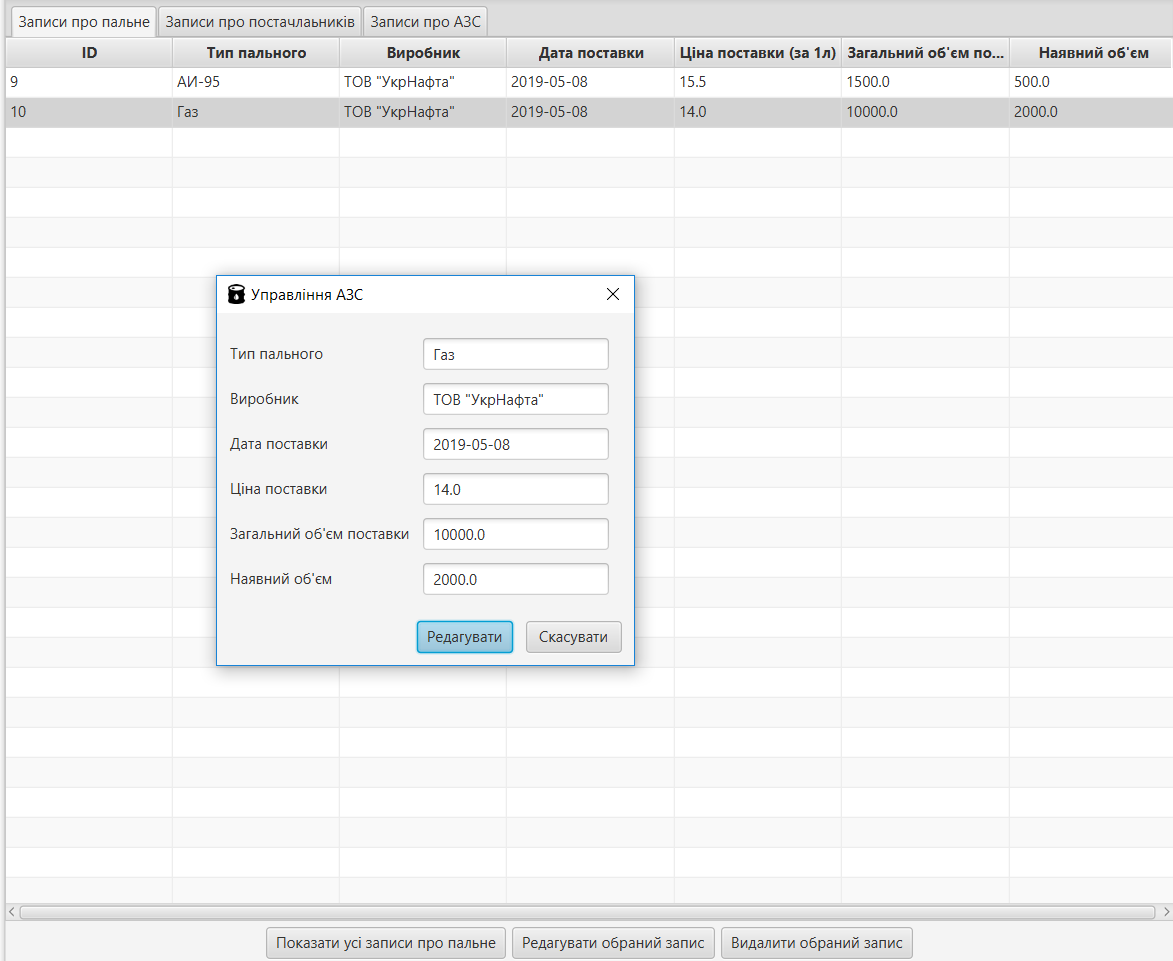
* 1. Переглянути усі записи про пальне

Для того, аби переглянути усі записи про пальне необхідно натиснути на кнопку «Показати усі записи про пальне». У випадку наявності записів у таблицю буде відображено всі записи про пальне, інашке будет виведено повідомлення, що записів у БД не має.



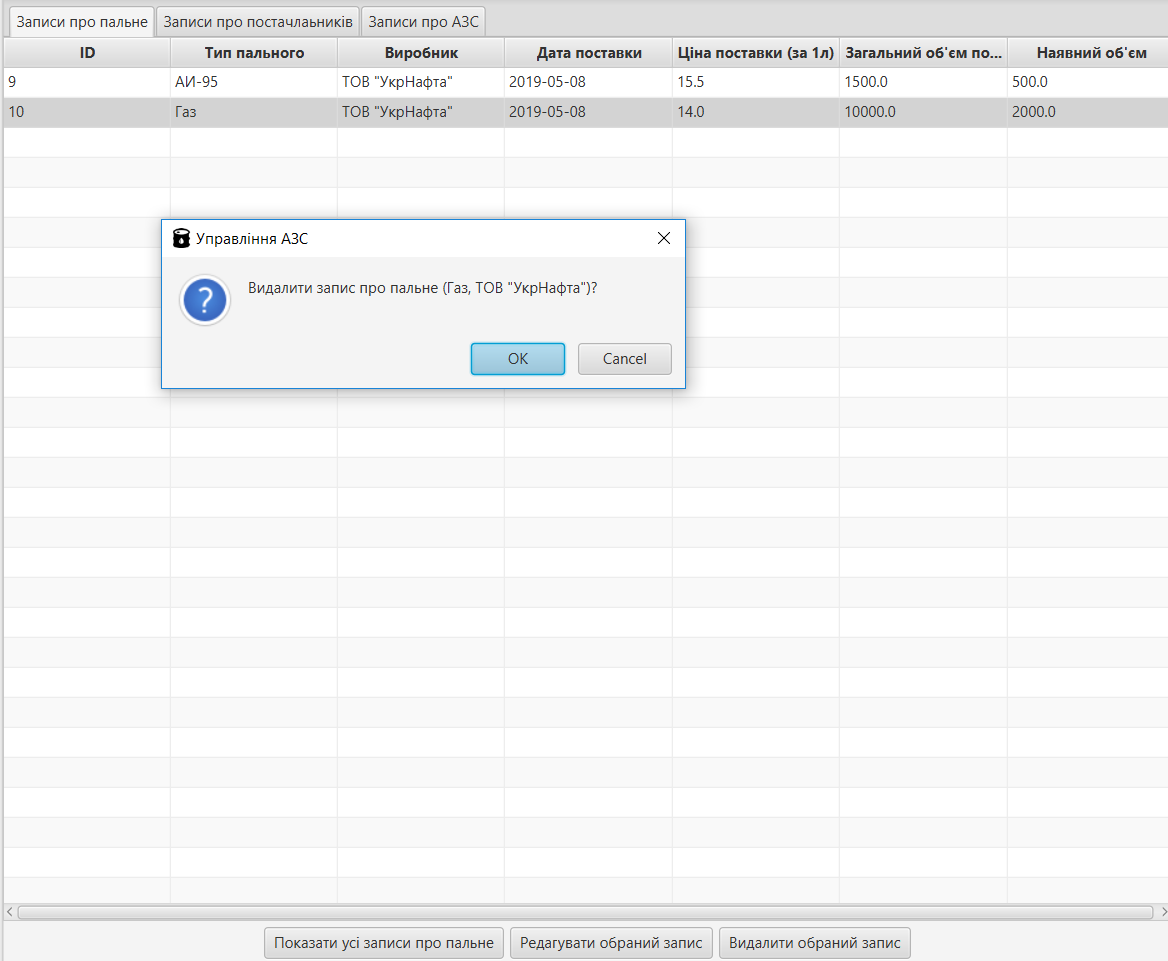
* 1. Редагувати запис про пальне

Для того, аби редагувати запис про пальне треба спочатку знати запис(и), або переглянути усі записи про пальне. Після цього необхідно вибрати (натиснути на запис у таблиці) та натиснути на кнопку «Редагувати обраний запис». Після цього відкриється вікно, у якому будуть доступні для редагування типа пального, виробник, дата поставки, ціна поставки, загальний об’єм пального, наявний об’єм пального. Після того, як буде зкореговано усі необхідні поля потрібно натиснути на кнопку «Редагувати». У випадку заповнення полів даними правильного формату запис про пальне будет зкореговано і це відобразиться у таблиці «Записи про пальне», інашке ви отримаете повідомлення про помилку та вікно редагування закріється. При натисканні кнопки «Скасувати» операція редагування запису буде перевана й жодні зміни до запису внесено не буде.



* 1. Видалити запис про пальне

Для того, аби видалити запис про пальне треба спочатку знати запис(и), або переглянути усі записи про пальне. Після цього необхідно вибрати (натиснути на запис у таблиці) та натиснути на кнопку «Видалити обраний запис». Після цього відкриється вікно, яке попросить підтвердити запит на видалення запису. У випадку натискання кнопки «OK» обраний запис буде видалено, у випадку натискання кнопки «Cancel» запис не буде видалено.

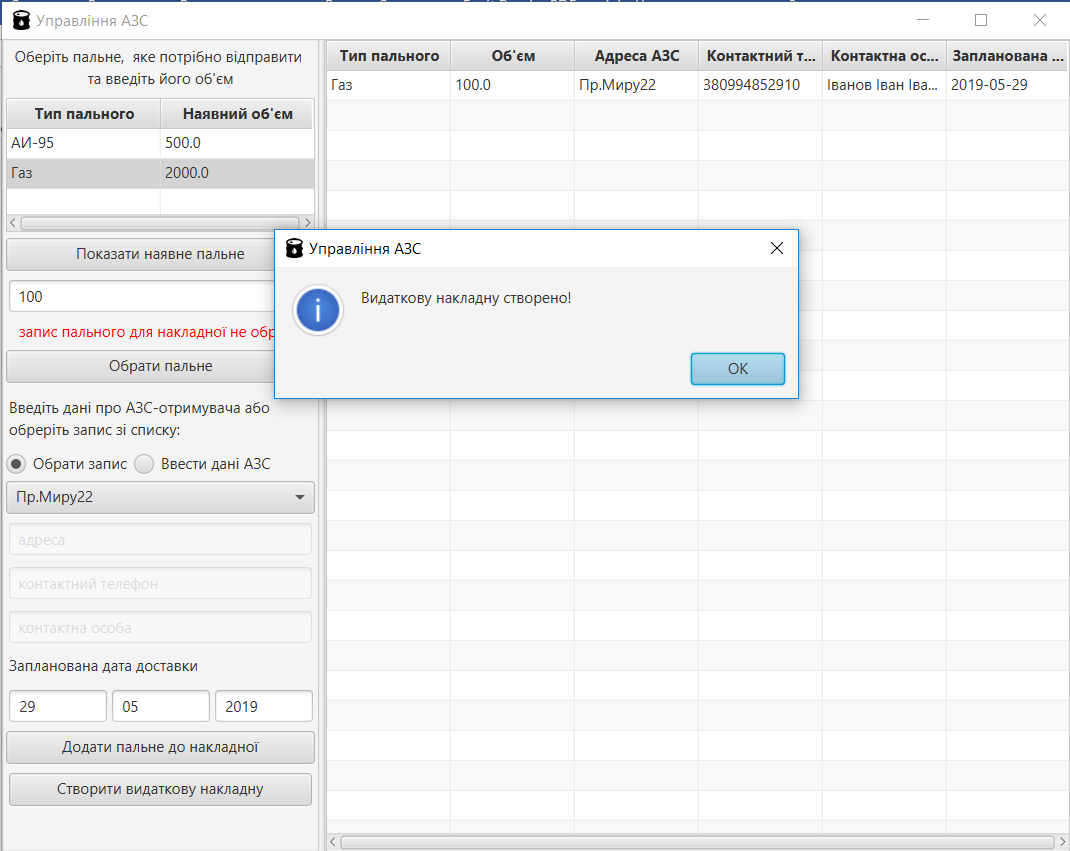


* 1. Створити видаткову накладну

Для того, аби створити видаткову накладну необхідно у верхній панелі натиснути на пункт «Накладні» та вибрати зі списку пункт «Створити видаткову накладну». Після цього відкриється ще одне вікно, у якому будет проходити створення видаткової накладної. Для того, аби створити видаткову накладну спочатку необхідно вибрати одне пальне з доступних. Для цього необхідно натиснути на кнопку «Показати наявне пальне». Ви побачите список усіх типів пального, що є, так їх об’єм. Після цього необхідно ввести в поле для введення об’єм той об’єм, який необхідно відправити та натиснути кнопку «Обрати пальне». У тому випадку, якщо буде обрано об’єм більший, ніж наявний об’єм пального, буде виведено повідомлення про помилку.

Після того, як пальне буде обрано, необхідно обрати АЗС, куда потрібно відправити пальне. Це можна зробити двома шляхами: обрати АЗС з тих, що вже є у БД, або ввести дані про АЗС (адресу, контактний телефон, контактну особу). Обов’язковим є введення запланованої дати доставки пального. Після того, як буде обрано АЗС тим, чи іншим шляхом та введено заплановану дату поставки можна створити запис у видатковії накладній натиснувши кнопку «Додати пальне до накладної». Якщо було введено дані неправильного формату, буде виведено повідомлення про помилку.

Крок з додаванням пального до накладної можна повторювати необхідну кількість разів. Після того, як усе необхідне пальне буде додано до накладної, тобто вона буде сформована, для її створення необхідно натиснути на кнопку «Створити видтакову накладку». У випадку, якщо у накадній не меньше 1 запису, вона будет створена.



# **Додаток В. Код програмних модулів**

Main.java

**package** sample;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.image.Image;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(**"loginWindow.fxml"**));  
 Parent root = loader. load();  
 primaryStage.setTitle(**"Управління АЗС"**);  
 primaryStage.setScene(**new** Scene(root));  
 primaryStage.getIcons().add(**new** Image(**"img/barrel.png"**));  
 primaryStage.setResizable(**false**);  
 LoginWindow loginWindow = loader.getController();  
 loginWindow.setStage(primaryStage);  
 *//root.requestFocus();* primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

LoginWindow. Java

**package** sample;  
  
**import** javafx.application.Application;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.image.Image;  
**import** javafx.stage.Stage;  
  
**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(**"loginWindow.fxml"**));  
 Parent root = loader. load();  
 primaryStage.setTitle(**"Управління АЗС"**);  
 primaryStage.setScene(**new** Scene(root));  
 primaryStage.getIcons().add(**new** Image(**"img/barrel.png"**));  
 primaryStage.setResizable(**false**);  
 LoginWindow loginWindow = loader.getController();  
 loginWindow.setStage(primaryStage);  
 *//root.requestFocus();* primaryStage.show();  
 }  
  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

OperatorWorkbench.java

**package** sample;  
  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeParseException;  
**import** java.util.List;  
**import** java.util.Optional;  
**import** java.util.ResourceBundle;  
  
**import** dao.ClientDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.geometry.Insets;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** javafx.scene.image.Image;  
**import** javafx.scene.layout.GridPane;  
**import** javafx.scene.paint.Color;  
**import** javafx.stage.Stage;  
**import** tables.Client;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
**import** javax.persistence.Persistence;  
  
**public class** OperatorWorkbench {  
  
 @FXML **private** ResourceBundle resources;  
 @FXML **private** URL location;  
 @FXML **private** TextField secondNameTextFieldAdd;  
 @FXML **private** TextField firstNameTextFieldAdd;  
 @FXML **private** TextField middleNameTextFieldAdd;  
 @FXML **private** TextField ddTextField;  
 @FXML **private** TextField mmTextField;  
 @FXML **private** TextField yyyyTextField;  
 @FXML **private** TextField phoneTextField;  
 @FXML **private** TextField emailTextField;  
 @FXML **private** TextField carModelTextField;  
 @FXML **private** TextField fuelTypeTextField;  
 @FXML **private** TextField loyaltyCardNumTextField;  
 @FXML **private** Button addRecordButton;  
 @FXML **private** Label addClientResultLabel;  
 @FXML **private** ChoiceBox<String> choiceBoxSearch;  
 @FXML **private** TextField searchByTextField;  
 @FXML **private** Button findRecordButton;  
 @FXML **private** MenuItem createClientReport;  
 @FXML **private** MenuItem createLoyaltyCardReport;  
 @FXML **private** Button showAllRecordButton;  
 @FXML **private** Button deleteSelectedRecordButton;  
 @FXML **private** Button editRecordButton;  
 @FXML **private** TableView<Client> clientTable;  
 @FXML **private** TableColumn<Client, Integer> idColumn;  
 @FXML **private** TableColumn<Client, String> secondNameColumn;  
 @FXML **private** TableColumn<Client, String> firstNameColumn;  
 @FXML **private** TableColumn<Client, String> middleNameColumn;  
 @FXML **private** TableColumn<Client, LocalDate> birthdayColumn;  
 @FXML **private** TableColumn<Client, String> phoneColumn;  
 @FXML **private** TableColumn<Client, String> emailColumn;  
 @FXML **private** TableColumn<Client, String> **carModelColumn**;  
 @FXML **private** TableColumn<Client, String> **fuelTypeColumn**;  
 @FXML **private** TableColumn<Client, Integer> **loyaltyCardNumColumn**;  
 @FXML **private** TableColumn<Client, LocalDate> **issuanceDateColumn**;  
 @FXML **private** TableColumn<Client, Double> **discountPercentColumn**;  
 @FXML **private** TableColumn<Client, Double> **boughtVolumeColumn**;  
  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** ObservableList<String> **choiceBoxItems**;  
 **private** List<Client> **clients**;  
  
 @FXML  
 **void** initialize() {  
 **choiceBoxItems** = FXCollections.*observableArrayList*(**"ID"**, **"Прізвище"**, **"Ім'я"**, **"По батькові"**,  
 **"Дата народження (у форматі дд-мм-рррр)"**, **"Контактний телефон"**, **"e-mail"**, **"Марка автомобіля"**, **"Тип Пального"**, **"Номер картки програми лояльності"**,  
 **"Дата отримання картки (у форматі дд-мм-рррр)"**, **"Відсоток знижки"**);  
 **entityManagerFactory** = Persistence.*createEntityManagerFactory*(**"MyPU"**);  
 setCellValueFactories();  
 **choiceBoxSearch** = **new** ChoiceBox<>(**choiceBoxItems**);  
  
 **addRecordButton**.setOnAction(event -> addRecord());  
 **findRecordButton**.setOnAction(event -> search());  
 **deleteSelectedRecordButton**.setOnAction(event -> deleteSelectedRecord());  
 **showAllRecordButton**.setOnAction(event -> show());  
 **editRecordButton**.setOnAction(event -> editRecord());  
 **createClientReport**.setOnAction(event -> makeClientReport());  
 createLoyaltyCardReport.setOnAction(event -> makeCardReport());}  
  
 **private void** makeCardReport() {  
 loadReportWindow(**"cardReport.fxml"**, **"loyaltyCardReport"**);  
 }  
  
 **private void** makeClientReport() {  
 loadReportWindow(**"clientReport.fxml"**, **"clientReport"**);  
 }  
  
 **private void** editRecord() {  
 Client client = clientTable.getSelectionModel().getSelectedItem();  
 **if** (client == **null**)  
 **return**;  
  
 Dialog<Client> dialog = **new** Dialog<>();  
 dialog.setTitle(**"Управління АЗС"**);  
 **var** window = showAllRecordButton.getScene().getWindow();  
 dialog.initOwner(window);  
 ButtonType okButtonType = **new** ButtonType(**"Редагувати"**, ButtonType.OK.getButtonData());  
 ButtonType cancelButtonType = **new** ButtonType(**"Скасувати"**, ButtonType.CANCEL.getButtonData());  
 dialog.getDialogPane().getButtonTypes().addAll(okButtonType, cancelButtonType);  
  
 GridPane grid = **new** GridPane();  
 grid.setHgap(10);  
 grid.setVgap(10);  
 grid.setPadding(**new** Insets(20, 20, 10, 10));  
  
 TextField secondName = **new** TextField(client.getSecondName());  
 secondName.setPromptText(**"Прізвище"**);  
 TextField firstName = **new** TextField(client.getFirstName());  
 firstName.setPromptText(**"Ім'я"**);  
 TextField middleName = **new** TextField(client.getMiddleName());  
 middleName.setPromptText(**"по батькові"**);  
 TextField birthday = **new** TextField(client.getBirthday().toString());  
 birthday.setPromptText(**"Дата народження"**);  
 TextField phone = **new** TextField(client.getPhone());  
 phone.setPromptText(**"Номер телефону"**);  
 TextField email = **new** TextField(client.getEmail());  
 email.setPromptText(**"email"**);  
 TextField carModel = **new** TextField(client.getCarModel());  
 carModel.setPromptText(**"Марка автомобіля"**);  
 TextField fuelType = **new** TextField(client.getFuelType());  
 fuelType.setPromptText(**"Тип пального"**);  
  
 grid.add(**new** Label(**"Прізвище:"**), 0, 0);  
 grid.add(secondName, 1, 0);  
 grid.add(**new** Label(**"Ім'я:"**), 0, 1);  
 grid.add(firstName, 1, 1);  
 grid.add(**new** Label(**"По батькові:"**), 0, 2);  
 grid.add(middleName, 1, 2);  
 grid.add(**new** Label(**"Дата народження:"**), 0, 3);  
 grid.add(birthday, 1, 3);  
 grid.add(**new** Label(**"Номер телефону"**), 0, 4);  
 grid.add(phone, 1, 4);  
 grid.add(**new** Label(**"email:"**), 0, 5);  
 grid.add(email, 1, 5);  
 grid.add(**new** Label(**"Марка автомобіля"**), 0, 6);  
 grid.add(carModel, 1, 6);  
 grid.add(**new** Label(**"Тип пального:"**), 0, 7);  
 grid.add(fuelType, 1, 7);  
  
 dialog.getDialogPane().setContent(grid);  
  
 dialog.setResultConverter(dialogButton -> {  
 **if** (dialogButton == okButtonType) {  
 **try** {  
 **int** indexSelected = clients.indexOf(client);  
 client.setSecondName(secondName.getText().trim());  
 client.setFirstName(firstName.getText().trim());  
 client.setMiddleName(middleName.getText().trim());  
 client.setBirthday(LocalDate.parse(birthday.getText().trim()));  
 client.setPhone(phone.getText().trim());  
 client.setEmail(email.getText().trim());  
 client.setCarModel(carModel.getText().trim());  
 client.setFuelType(fuelType.getText().trim());  
  
  
 EntityManager em = entityManagerFactory.createEntityManager();  
 ClientDAO clientDAO = **new** ClientDAO(em);  
 clients.set(indexSelected, client);  
 clientDAO.editRecord(client);  
 clientTable.setItems(FXCollections.observableArrayList(clients));  
 clientTable.refresh();  
 } **catch** (NullPointerException e) {  
 alertWindow.errorAlert(**"Помилка! Переверте введені дані."**);  
 *//return null;* } **catch** (NumberFormatException e) {  
 alertWindow.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 *//return null;* } **catch** (DateTimeParseException e) {  
 alertWindow.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 *// return null;* }  
  
 }  
 **return null**;  
 });  
  
 Optional<Client> result = dialog.showAndWait();  
 }  
  
 **private void** deleteSelectedRecord() {  
 Client client = clientTable.getSelectionModel().getSelectedItem();  
 **if** (client != **null** && alertWindow.confirmationAlert(**"клієнта"**, client.briefInfo())) {  
 EntityManager em = entityManagerFactory.createEntityManager();  
 ClientDAO clientDAO = **new** ClientDAO(em);  
 clientDAO.delete(client);  
 clientTable.setItems(FXCollections.observableList(clientDAO.findAll()));  
 }  
 }  
  
 **private void** search() {  
 **if** (choiceBoxSearch.getValue() == **null**) {  
 alertWindow.errorAlert(**"Помилка! Оберіть критерій пошуку"**);  
 **return**;  
 }  
 **try** {  
 EntityManager em = entityManagerFactory.createEntityManager();  
 ClientDAO clientDAO = **new** ClientDAO(em);  
  
 String choice = choiceBoxSearch.getValue();  
 **if** (choice.equals(**"ID"**)) {  
 **int** id = Integer.parseInt(searchByTextField.getText().trim());  
 clients = clientDAO.searchById(id);  
 } **else if** (choice.equals(**"Прізвище"**)) {  
 String secondName = searchByTextField.getText().trim();  
 clients = clientDAO.searchBySecondName(secondName);  
 } **else if** (choice.equals(**"Ім'я"**)) {  
 String firstName = searchByTextField.getText().trim();  
 clients = clientDAO.searchByFirstName(firstName);  
 } **else if** (choice.equals(**"По батькові"**)) {  
 String middleName = searchByTextField.getText().trim();  
 clients = clientDAO.searchByMiddleName(middleName);  
 } **else if** (choice.equals(**"Дата народження (у форматі дд-мм-рррр)"**)) {  
 LocalDate birthday = LocalDate.parse(searchByTextField.getText().trim());  
 clients = clientDAO.searchByBirthday(birthday);  
 } **else if** (choice.equals(**"Контактний телефон"**)) {  
 String phone = searchByTextField.getText().trim();  
 clients = clientDAO.searchByPhone(phone);  
 } **else if** (choice.equals(**"e-mail"**)) {  
 String email = searchByTextField.getText().trim();  
 clients = clientDAO.searchByEmail(email);  
 } **else if** (choice.equals(**"Марка автомобіля"**)) {  
 String carModel = searchByTextField.getText().trim();  
 clients = clientDAO.searchByCarModel(carModel);  
 } **else if** (choice.equals(**"Тип пального"**)) {  
 String fuelType = searchByTextField.getText().trim();  
 clients = clientDAO.searchByFuelType(fuelType);  
 } **else if** (choice.equals(**"Номер картки програми лояльності"**)) {  
 **int** loyaltyCardNum = Integer.parseInt(searchByTextField.getText().trim());  
 clients = clientDAO.searchByLoyaltyCardNum(loyaltyCardNum);  
 } **else if** (choice.equals(**"Дата отримання картки (у форматі дд-мм-рррр)"**)) {  
 LocalDate issuanceDate = LocalDate.parse(searchByTextField.getText().trim());  
 clients = clientDAO.searchByIssuanceDate(issuanceDate);  
 } **else if** (choice.equals(**"Відсоток знижки"**)) {  
 **double** discountPercent = Double.parseDouble(searchByTextField.getText().trim());  
 clients = clientDAO.searchByDiscountPercent(discountPercent);  
 }  
 searchByTextField.clear();  
 clientTable.getItems().clear();  
 clientTable.setItems(FXCollections.observableArrayList(clients));  
 } **catch** (NullPointerException e) {  
 alertWindow.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 alertWindow.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 alertWindow.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 }  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory managerFactory) {  
 entityManagerFactory = managerFactory;  
 }  
  
 **private void** addRecord() {  
  
 **if**(firstNameTextFieldAdd.getText().isEmpty() || secondNameTextFieldAdd.getText().isEmpty() || middleNameTextFieldAdd.getText().isEmpty() ||  
 yyyyTextField.getText().isEmpty() || mmTextField.getText().isEmpty() || ddTextField.getText().isEmpty() ||  
 emailTextField.getText().isEmpty() || phoneTextField.getText().isEmpty() || carModelTextField.getText().isEmpty() ||  
 fuelTypeTextField.getText().isEmpty()|| loyaltyCardNumTextField.getText().isEmpty()) {  
 alertWindow.errorAlert(**"Помилка! Одне або декілька полів пусті!"**);  
 **return**;  
 }  
  
 **try** {  
 Client client = **new** Client();  
 client.setFirstName(firstNameTextFieldAdd.getText().trim());  
 client.setSecondName(secondNameTextFieldAdd.getText().trim());  
 client.setMiddleName(middleNameTextFieldAdd.getText().trim());  
 client.setBirthday(LocalDate.parse(yyyyTextField.getText().trim() + **"-"** + mmTextField.getText().trim() + **"-"** + ddTextField.getText().trim()));  
 client.setEmail(emailTextField.getText().trim());  
 client.setPhone(phoneTextField.getText().trim());  
 client.setCarModel(carModelTextField.getText().trim());  
 client.setFuelType(fuelTypeTextField.getText().trim());  
 client.setLoyaltyCardNum(Integer.parseInt(loyaltyCardNumTextField.getText().trim()));  
 client.setIssuanceDate(LocalDate.now());  
 client.setBoughtVolume(0);  
 client.setDiscountPercent(0);  
  
 EntityManager em = entityManagerFactory.createEntityManager();  
 ClientDAO clientDAO = **new** ClientDAO(em);  
 clientDAO.add(client);  
 addClientResultLabel.setText(**"Запис про постачальника успішно додано!"**);  
 addClientResultLabel.setTextFill(Color.web(**"#00FF00"**));  
  
 **if** (!clientTable.getItems().isEmpty())  
 clientTable.setItems(FXCollections.observableList(clientDAO.findAll()));  
  
 firstNameTextFieldAdd.clear();  
 secondNameTextFieldAdd.clear();  
 middleNameTextFieldAdd.clear();  
 ddTextField.clear();  
 mmTextField.clear();  
 yyyyTextField.clear();  
 emailTextField.clear();  
 phoneTextField.clear();  
 carModelTextField.clear();  
 fuelTypeTextField.clear();  
 loyaltyCardNumTextField.clear();  
  
 } **catch** (NullPointerException e) {  
 alertWindow.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 alertWindow.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 alertWindow.errorAlert(**"Помилка при введені дати!"**);  
 }  
  
  
 }  
  
 **private void** setCellValueFactories() {  
 idColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 secondNameColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"secondName"**));  
 firstNameColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"firstName"**));  
 middleNameColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"middleName"**));  
 birthdayColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"birthday"**));  
 phoneColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"phone"**));  
 emailColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"email"**));  
 carModelColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"carModel"**));  
 fuelTypeColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 loyaltyCardNumColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"loyaltyCardNum"**));  
 issuanceDateColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"issuanceDate"**));  
 discountPercentColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"discountPercent"**));  
 boughtVolumeColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"boughtVolume"**));  
 }  
  
 **private void** show() {  
 **try** {  
 EntityManager em = entityManagerFactory.createEntityManager();  
 ClientDAO clientDAO = **new** ClientDAO(em);  
 clients = clientDAO.findAll();  
 **if** (clients.isEmpty()) {  
 alertWindow.errorAlert(**"Не знайдено жодного запису!"**);  
 **return**;  
 }  
 clientTable.getItems().clear();  
 clientTable.setItems(FXCollections.observableList(clients));  
 }  
 **catch** (Exception ex) {  
 alertWindow.errorAlert(ex.getMessage());  
 }  
 }  
  
 **private void** loadReportWindow(String url, String window) {  
 Stage st = **new** Stage();  
 Parent root = **null**;  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(url));  
  
 **try** {  
 root = loader.load();  
 } **catch** (IOException e) {  
 alertWindow.errorAlert(e.getMessage());  
 e.printStackTrace();  
 }  
  
 **if** (window.equals(**"clientReport"**)) {  
 ClientReportWindow clientReportWindow = loader.getController();  
 clientReportWindow.setEntityManagerFactory(entityManagerFactory);  
 }  
 **else if** (window.equals(**"loyaltyCardReport"**)) {  
 CardReportWindow cardReportWindow = loader.getController();  
 cardReportWindow.setEntityManagerFactory(entityManagerFactory);  
 }  
  
 st.setTitle(**"Управління АЗС"**);  
 Scene scene = **new** Scene(root);  
 st.setScene(scene);  
 st.getIcons().add(**new** Image(**"img/barrel.png"**));  
 st.showAndWait();  
 }  
}

ManagerWorkbench.java

**package** sample;  
  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeParseException;  
**import** java.util.List;  
**import** java.util.Optional;  
**import** java.util.ResourceBundle;  
  
**import** dao.FuelDAO;  
**import** dao.FuelVendorDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.geometry.Insets;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.Dialog;  
**import** javafx.scene.control.Label;  
**import** javafx.scene.control.MenuItem;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** javafx.scene.image.Image;  
**import** javafx.scene.layout.GridPane;  
**import** javafx.scene.paint.Color;  
**import** javafx.stage.Stage;  
**import** reports.FuelBalanceReport;  
**import** tables.Fuel;  
**import** tables.FuelVendor;  
**import** tables.PetrolStation;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** ManagerWorkbench {  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** TextField **fuelTypeTextField**;  
 @FXML **private** TextField **manufacturerTextField**;  
 @FXML **private** ChoiceBox<String> **selectVendorChoiceBox**;  
 @FXML **private** TextField **ddTextField**;  
 @FXML **private** TextField **mmTextField**;  
 @FXML **private** TextField **yyyyTextField**;  
 @FXML **private** TextField **priceTextField**;  
 @FXML **private** TextField **totalVolumeTextField**;  
 @FXML **private** Button **addFuelRecordButton**;  
 @FXML **private** Label **addFuelResultLabel**;  
 @FXML **private** TextField **companyTitleTextField**;  
 @FXML **private** TextField **secondNameTextField**;  
 @FXML **private** TextField **firstNameTextField**;  
 @FXML **private** TextField **middleNameTextField**;  
 @FXML **private** TextField **phoneTextField**;  
 @FXML **private** TextField **addressTextField**;  
 @FXML **private** TextField **contractTextField**;  
 @FXML **private** TextField **ddContractTextField1**;  
 @FXML **private** TextField **mmContractTextField1**;  
 @FXML **private** TextField **yyyyContractTextField1**;  
 @FXML **private** TextField **ddContractTextField2**;  
 @FXML **private** TextField **mmContractTextField2**;  
 @FXML **private** TextField **yyyyContractTextField2**;  
 @FXML **private** Button **addVendorRecordButton**;  
 @FXML **private** Label **addVendorResultLabel**;  
 @FXML **private** ChoiceBox<String> **choiceBoxSearchVendor**;  
 @FXML **private** TextField **searchVendorTextField**;  
 @FXML **private** Button **findVendorRecordButton**;  
 @FXML **private** ChoiceBox<String> **choiceBoxSearchFuel**;  
 @FXML **private** TextField **searchFuelTextField**;  
 @FXML **private** Button **findFuelRecordButton**;  
 @FXML **private** TableView<Fuel> **fuelTable**;  
 @FXML **private** TableColumn<Fuel, Integer> **idFuelColumn**;  
 @FXML **private** TableColumn<Fuel, String> **fuelTypeColumn**;  
 @FXML **private** TableColumn<Fuel, String> **manufacturerColumn**;  
 @FXML **private** TableColumn<Fuel, LocalDate> **dateOfSupplyColumn**;  
 @FXML **private** TableColumn<Fuel, Double> **priceOfSupplyColumn**;  
 @FXML **private** TableColumn<Fuel, Double> **totalVolumeColumn**;  
 @FXML **private** TableColumn<Fuel, Double> **availableVolumeColumn**;  
 @FXML **private** Button **showAllFuelRecordButton**;  
 @FXML **private** Button **correctSelectedFuelRecordButton**;  
 @FXML **private** Button **deleteSelectedFuelRecordButton**;  
 @FXML **private** TableView<FuelVendor> **vendorTable**;  
 @FXML **private** TableColumn<FuelVendor, Integer> **idVendorColumn**;  
 @FXML **private** TableColumn<FuelVendor, String> **companyTitleColumn**;  
 @FXML **private** TableColumn<FuelVendor, String> **contactPersonColumn**;  
 @FXML **private** TableColumn<FuelVendor, String> **contactPhoneColumn**;  
 @FXML **private** TableColumn<FuelVendor, String> **addressColumn**;  
 @FXML **private** TableColumn<FuelVendor, String> **contractColumn**;  
 @FXML **private** TableColumn<FuelVendor, LocalDate> **dateOfNegotiationColumn**;  
 @FXML **private** TableColumn<FuelVendor, LocalDate> **endDateColumn**;  
 @FXML **private** Button **showAllVendorRecordButton**;  
 @FXML **private** Button **correctSelectedVendorRecordButton**;  
 @FXML **private** Button **deleteSelectedVendorRecordButton**;  
 @FXML **private** TableView<PetrolStation> **tablePetrolStation**;  
 @FXML **private** TableColumn<PetrolStation, Integer> **idPSColumn**;  
 @FXML **private** TableColumn<PetrolStation, String> **addressPSColumn**;  
 @FXML **private** TableColumn<PetrolStation, String> **contactPhonePSColumn**;  
 @FXML **private** TableColumn<PetrolStation, String> **contactPersonPSColumn**;  
 @FXML **private** TableColumn<PetrolStation, String> **fuelTypePSColumn**;  
 @FXML **private** TableColumn<PetrolStation, Double> **totalFuelVolumeColumn**;  
 @FXML **private** TableColumn<PetrolStation, Double> **soldFuelVolumeColumn**;  
 @FXML **private** TableColumn<PetrolStation, LocalDate> **supplyDateColumn**;  
 @FXML **private** TableColumn<PetrolStation, Double> **salePriceColumn**;  
 @FXML **private** Button **showAllPetrolStationRecordButton**;  
 @FXML **private** Button **correctSelectedPSRecordButton**;  
 @FXML **private** Button **deleteSelectedPSRecordButton**;  
 @FXML **private** MenuItem **createFuelBalanceReportItem**;  
 @FXML **private** MenuItem **createFuelTransferReportItem**;  
 @FXML **private** MenuItem **createVendorReportItem**;  
 @FXML **private** MenuItem **createWaybillItem**;  
  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** ObservableList<String> **choiceBoxSearchFuelItems** = FXCollections.*observableArrayList*(**"ID"**, **"Тип пального"**,  
 **"Виробник"**, **"Дата поставки (у форматі рррр-мм-дд)"**);  
 **private** ObservableList<String> **choiceBoxSearchVendorItems** = FXCollections.*observableArrayList*(**"ID"**, **"Назва компанії"**, **"Контактний телефон"**,  
 **"Адреса"**, **"Контракт"**);  
  
 **private** List<Fuel> **fuels**;  
 **private** List<FuelVendor> **vendors**;  
 **private** List<PetrolStation> **petrolStations**;  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **choiceBoxSearchFuel**.setItems(**choiceBoxSearchFuelItems**);  
 **choiceBoxSearchVendor**.setItems(**choiceBoxSearchVendorItems**);  
  
 **addFuelRecordButton**.setOnAction(event -> addFuelRecord());  
 **addVendorRecordButton**.setOnAction(event -> addVendorRecord());  
 **showAllFuelRecordButton**.setOnAction(event -> showFuelRecords());  
 **showAllVendorRecordButton**.setOnAction(event -> showFuelVendorRecords());  
 **correctSelectedFuelRecordButton**.setOnAction(event -> editFuelRecord());  
 **correctSelectedVendorRecordButton**.setOnAction(event -> editVendorRecord());  
 **findFuelRecordButton**.setOnAction(event -> findFuelRecords());  
 **findVendorRecordButton**.setOnAction(event -> findVendorRecords());  
 **deleteSelectedFuelRecordButton**.setOnAction(event -> deleteSelectedFuelRecord());  
 **deleteSelectedVendorRecordButton**.setOnAction(event -> deleteSelectedVendorRecord());  
 **createFuelBalanceReportItem**.setOnAction(event -> createFuelBalanceReport());  
 **createFuelTransferReportItem**.setOnAction(event -> createFuelTransferReport());  
 **createVendorReportItem**.setOnAction(event -> createVendorReport());  
 **createWaybillItem**.setOnAction(event -> createWaybill());  
  
 **selectVendorChoiceBox**.setOnMousePressed(event -> setVendorSelectChoiceBoxItems());  
  
 }  
  
 **private void** createWaybill() {  
 loadReportWindow(**"waybill.fxml"**, **"Waybill"**);  
 }  
  
 **private void** createVendorReport() {  
 loadReportWindow(**"vendorReport.fxml"**, **"vendorReport"**);  
 }  
  
 **private void** createFuelTransferReport() {  
 loadReportWindow(**"fuelTransferReport.fxml"**, **"fuelTransferReport"**);  
 }  
  
 **private void** createFuelBalanceReport() {  
 loadReportWindow(**"balanceReport.fxml"**, **"fuelBalanceReport"**);  
 }  
  
 **private void** deleteSelectedVendorRecord() {  
 FuelVendor fuelVendor = **vendorTable**.getSelectionModel().getSelectedItem();  
 **if** (fuelVendor != **null** &&  
 **alertWindow**.confirmationAlert(**"запис про постачальника"**, fuelVendor.briefInfo())) {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 fuelVendorDAO.delete(fuelVendor);  
 **vendorTable**.setItems(FXCollections.*observableList*(fuelVendorDAO.findAll()));  
 }  
 }  
  
 **private void** deleteSelectedFuelRecord() {  
 Fuel fuel = **fuelTable**.getSelectionModel().getSelectedItem();  
 **if** (fuel != **null** &&  
 **alertWindow**.confirmationAlert(**"запис про пальне"**, fuel.briefInfo())) {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 fuelDAO.delete(fuel);  
 **fuelTable**.setItems(FXCollections.*observableList*(fuelDAO.findAll()));  
 }  
 }  
  
 **private void** findVendorRecords() {  
 **if** (**choiceBoxSearchVendor**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть критерій пошуку"**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 **if** (**choiceBoxSearchVendor**.getValue().equals(**"ID"**)) {  
 **int** id = Integer.*parseInt*(**searchVendorTextField**.getText().trim());  
 **vendors** = fuelVendorDAO.findByID(id);  
 } **else if** (**choiceBoxSearchVendor**.getValue().equals(**"Назва компанії"**)) {  
 String companyTitle = **searchVendorTextField**.getText().trim();  
 **vendors** = fuelVendorDAO.findByCompanyTitle(companyTitle);  
 } **else if** (**choiceBoxSearchVendor**.getValue().equals(**"Контактний телефон"**)) {  
 String phone = **searchVendorTextField**.getText().trim();  
 **vendors** = fuelVendorDAO.findByContactPhone(phone);  
 } **else if** (**choiceBoxSearchVendor**.getValue().equals(**"Адреса"**)) {  
 String address = **searchVendorTextField**.getText().trim();  
 **vendors** = fuelVendorDAO.findByAddress(address);  
 } **else if** (**choiceBoxSearchVendor**.getValue().equals(**"Контракт"**)) {  
 String contract = **searchVendorTextField**.getText().trim();  
 **vendors** = fuelVendorDAO.findByContract(contract);  
 }  
  
 **searchVendorTextField**.clear();  
 **vendorTable**.getItems().clear();  
 **vendorTable**.setItems(FXCollections.*observableArrayList*(**vendors**));  
  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 }  
  
 }  
  
 **private void** findFuelRecords() {  
 **if** (**choiceBoxSearchFuel**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть критерій пошуку"**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 **if** (**choiceBoxSearchFuel**.getValue().equals(**"ID"**)) {  
 **int** id = Integer.*parseInt*(**searchFuelTextField**.getText().trim());  
 **fuels** = fuelDAO.findByID(id);  
 } **else if** (**choiceBoxSearchFuel**.getValue().equals(**"Тип пального"**)) {  
 String fuelType = **searchFuelTextField**.getText().trim();  
 **fuels** = fuelDAO.findByFuelType(fuelType);  
 } **else if** (**choiceBoxSearchFuel**.getValue().equals(**"Виробник"**)) {  
 String manufacturer = **searchFuelTextField**.getText().trim();  
 **fuels** = fuelDAO.findByManufacturer(manufacturer);  
 } **else if** (**choiceBoxSearchFuel**.getValue().equals(**"Дата поставки (у форматі рррр-мм-дд)"**)) {  
 LocalDate supplyDate = LocalDate.*parse*(**searchFuelTextField**.getText().trim());  
 **fuels** = fuelDAO.findBySupplyDate(supplyDate);  
  
 **searchFuelTextField**.clear();  
 **fuelTable**.getItems().clear();  
 **fuelTable**.setItems(FXCollections.*observableArrayList*(**fuels**));  
 }  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 }  
 }  
  
 **private void** setVendorSelectChoiceBoxItems() {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 ObservableList<String> companies = FXCollections.*observableArrayList*(fuelVendorDAO.getVendorsCompany());  
 **selectVendorChoiceBox**.setItems(companies);  
 }  
  
 **private void** editVendorRecord() {  
 FuelVendor fuelVendor = **vendorTable**.getSelectionModel().getSelectedItem();  
 **if** (fuelVendor == **null**)  
 **return**;  
  
 Dialog<Fuel> dialog = **new** Dialog<>();  
 **var** window = **showAllFuelRecordButton**.getScene().getWindow();  
 dialog.initOwner(window);  
 dialog.setTitle(**"Управління АЗС"**);  
  
 ButtonType okButtonType = **new** ButtonType(**"Редагувати"**, ButtonType.***OK***.getButtonData());  
 ButtonType cancelButtonType = **new** ButtonType(**"Скасувати"**, ButtonType.***CANCEL***.getButtonData());  
 dialog.getDialogPane().getButtonTypes().addAll(okButtonType, cancelButtonType);  
 GridPane grid = **new** GridPane();  
 grid.setHgap(10);  
 grid.setVgap(10);  
 grid.setPadding(**new** Insets(20, 20, 10, 10));  
  
  
 TextField companyTitle = **new** TextField(fuelVendor.getCompanyTitle() );  
 companyTitle.setPromptText(**"назва компанії"**);  
 TextField contactPerson = **new** TextField(fuelVendor.getContactPerson());  
 contactPerson.setPromptText(**"контактна особа"**);  
 TextField address = **new** TextField(fuelVendor.getAddress());  
 address.setPromptText(**"адреса"**);  
 TextField phone = **new** TextField(fuelVendor.getPhone());  
 phone.setPromptText(**"контактний телефон"**);  
 TextField contract = **new** TextField(fuelVendor.getContract());  
 contract.setPromptText(**"контракт"**);  
 TextField dateOfNegotiation = **new** TextField(fuelVendor.getDateOfNegotiation().toString());  
 dateOfNegotiation.setPromptText(**"дата укладення контракту"**);  
 TextField endDate = **new** TextField(fuelVendor.getEndDate().toString());  
 endDate.setPromptText(**"дата закінчення дії контракту"**);  
  
 grid.add(**new** Label(**"Назва компанії"**), 0, 0);  
 grid.add(companyTitle, 1, 0);  
 grid.add(**new** Label(**"Контактна особа"**), 0, 1);  
 grid.add(contactPerson, 1, 1);  
 grid.add(**new** Label(**"Контактний телефон"**), 0, 2);  
 grid.add(phone, 1, 2);  
 grid.add(**new** Label(**"Адреса"**), 0, 3);  
 grid.add(address, 1, 3);  
 grid.add(**new** Label(**"Контракт"**), 0, 4);  
 grid.add(contract, 1, 4);  
 grid.add(**new** Label(**"Дата укладення контракту"**), 0, 5);  
 grid.add(dateOfNegotiation, 1, 5);  
 grid.add(**new** Label(**"Дата закінчення дії контракту"**), 0, 6);  
 grid.add(endDate, 1, 6);  
  
 dialog.getDialogPane().setContent(grid);  
  
 dialog.setResultConverter(dialogButton -> {  
  
 **if** (dialogButton == okButtonType) {  
 **try** {  
 **int** indexSelected = **vendors**.indexOf(fuelVendor);  
 fuelVendor.setCompanyTitle(companyTitle.getText().trim());  
 fuelVendor.setContactPerson(contactPerson.getText().trim());  
 fuelVendor.setPhone(phone.getText().trim());  
 fuelVendor.setAddress(address.getText().trim());  
 fuelVendor.setContract(contract.getText().trim());  
 fuelVendor.setDateOfNegotiation(LocalDate.*parse*(dateOfNegotiation.getText().trim()));  
 fuelVendor.setEndDate(LocalDate.*parse*(endDate.getText().trim()));  
  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 **vendors**.set(indexSelected, fuelVendor);  
 fuelVendorDAO.editRecord(fuelVendor);  
 **vendorTable**.setItems(FXCollections.*observableArrayList*(**vendors**));  
 **vendorTable**.refresh();  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 *//return null;* } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 *//return null;* } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 *// return null;* }  
 }  
 **return null**;  
 });  
  
 Optional<Fuel> result = dialog.showAndWait();  
 }  
  
 **private void** editFuelRecord() {  
 Fuel fuel = **fuelTable**.getSelectionModel().getSelectedItem();  
 **if** (fuel == **null**)  
 **return**;  
 Dialog<Fuel> dialog = **new** Dialog<>();  
 **var** window = **showAllFuelRecordButton**.getScene().getWindow();  
 dialog.initOwner(window);  
 dialog.setTitle(**"Управління АЗС"**);  
  
 ButtonType okButtonType = **new** ButtonType(**"Редагувати"**, ButtonType.***OK***.getButtonData());  
 ButtonType cancelButtonType = **new** ButtonType(**"Скасувати"**, ButtonType.***CANCEL***.getButtonData());  
 dialog.getDialogPane().getButtonTypes().addAll(okButtonType, cancelButtonType);  
 GridPane grid = **new** GridPane();  
 grid.setHgap(10);  
 grid.setVgap(10);  
 grid.setPadding(**new** Insets(20, 20, 10, 10));  
  
  
 TextField fuelType = **new** TextField(fuel.getType());  
 fuelType.setPromptText(**"тип пального"**);  
 TextField manufacturer = **new** TextField(fuel.getManufacturer());  
 manufacturer.setPromptText(**"виробник"**);  
 TextField dateOfSupply = **new** TextField(fuel.getDateOfSupply().toString());  
 dateOfSupply.setPromptText(**"дата поставки"**);  
 TextField priceOfSupply = **new** TextField(Double.*toString*(fuel.getPriceOfSupply()));  
 priceOfSupply.setPromptText(**"ціна поставки"**);  
 TextField totalVolume = **new** TextField(Double.*toString*(fuel.getTotalVolume()));  
 totalVolume.setPromptText(**"загальний об'єм поставки"**);  
 TextField availableVolume = **new** TextField(Double.*toString*(fuel.getAvailableVolume()));  
 availableVolume.setPromptText(**"наявний об'єм"**);  
  
 grid.add(**new** Label(**"Тип пального"**), 0, 0);  
 grid.add(fuelType, 1, 0);  
 grid.add(**new** Label(**"Виробник"**), 0, 1);  
 grid.add(manufacturer, 1, 1);  
 grid.add(**new** Label(**"Дата поставки"**), 0, 2);  
 grid.add(dateOfSupply, 1, 2);  
 grid.add(**new** Label(**"Ціна поставки"**), 0, 3);  
 grid.add(priceOfSupply, 1, 3);  
 grid.add(**new** Label(**"Загальний об'єм поставки"**), 0, 4);  
 grid.add(totalVolume, 1, 4);  
 grid.add(**new** Label(**"Наявний об'єм"**), 0, 5);  
 grid.add(availableVolume, 1, 5);  
  
 dialog.getDialogPane().setContent(grid);  
  
 *// Enable/Disable login button depending on whether a username was entered.  
 //Node loginButton = dialog.getDialogPane().lookupButton(okButtonType);  
 //loginButton.setDisable(true);  
  
 // Do some validation (using the Java 8 lambda syntax).  
 /\* manufacturer.textProperty().addListener((observable, oldValue, newValue) -> {  
 loginButton.setDisable(newValue.trim().isEmpty());  
 });\*/* dialog.setResultConverter(dialogButton -> {  
  
 **if** (dialogButton == okButtonType) {  
 **int** indexSelected = **fuels**.indexOf(fuel);  
 **try** {  
 fuel.setType(fuelType.getText().trim());  
 fuel.setManufacturer(manufacturer.getText().trim());  
 fuel.setDateOfSupply(LocalDate.*parse*(dateOfSupply.getText().trim()));  
 fuel.setPriceOfSupply(Double.*parseDouble*(priceOfSupply.getText().trim()));  
 fuel.setTotalVolume(Double.*parseDouble*(totalVolume.getText().trim()));  
 fuel.setAvailableVolume(Double.*parseDouble*(availableVolume.getText().trim()));  
  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 **fuels**.set(indexSelected, fuel);  
 fuelDAO.editRecord(fuel);  
 **fuelTable**.setItems(FXCollections.*observableArrayList*(**fuels**));  
 **fuelTable**.refresh();  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 *//return null;* } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 *//return null;* } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 *// return null;* }  
 }  
 **return null**;  
 });  
  
 Optional<Fuel> result = dialog.showAndWait();  
 }  
  
 **private void** showFuelVendorRecords() {  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 **vendors** = fuelVendorDAO.findAll();  
 **if** (**vendors**.isEmpty()) {  
 **alertWindow**.errorAlert(**"Не знайдено жодного запису!"**);  
 **return**;  
 }  
 **vendorTable**.getItems().clear();  
 **vendorTable**.setItems(FXCollections.*observableList*(**vendors**));  
 }  
 **catch** (Exception ex) {  
 **alertWindow**.errorAlert(ex.getMessage());  
 }  
 }  
  
 **private void** addVendorRecord() {  
 **if**(**companyTitleTextField**.getText().isEmpty() || **secondNameTextField**.getText().isEmpty() || **firstNameTextField**.getText().isEmpty() ||  
 **middleNameTextField**.getText().isEmpty() || **phoneTextField**.getText().isEmpty() || **addressTextField**.getText().isEmpty() ||  
 **contractTextField**.getText().isEmpty() || **yyyyContractTextField1**.getText().isEmpty() || **mmContractTextField1**.getText().isEmpty() ||  
 **ddContractTextField1**.getText().isEmpty()|| **yyyyContractTextField2**.getText().isEmpty() || **mmContractTextField1**.getText().isEmpty()  
 || **ddContractTextField2**.getText().isEmpty()) {  
 **alertWindow**.errorAlert(**"Помилка! Одне або декілька полів пусті!"**);  
 **return**;  
 }  
  
 **try** {  
 **addVendorResultLabel**.setText(**""**);  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
  
 FuelVendor fuelVendor = **new** FuelVendor();  
 fuelVendor.setCompanyTitle(**companyTitleTextField**.getText().trim());  
 fuelVendor.setContactPerson(**secondNameTextField**.getText().trim() + **" "** + **firstNameTextField**.getText().trim() + **" "** + **middleNameTextField**.getText().trim());  
 fuelVendor.setPhone(**phoneTextField**.getText().trim());  
 fuelVendor.setAddress(**addressTextField**.getText().trim());  
 fuelVendor.setContract(**contractTextField**.getText().trim());  
 fuelVendor.setDateOfNegotiation(LocalDate.*parse*(**yyyyContractTextField1**.getText().trim() + **"-"** + **mmContractTextField1**.getText().trim() + **"-"** +  
 **ddContractTextField1**.getText().trim()));  
 fuelVendor.setEndDate(LocalDate.*parse*(**yyyyContractTextField2**.getText().trim() + **"-"** + **mmContractTextField2**.getText().trim() + **"-"** +  
 **ddContractTextField2**.getText().trim()));  
  
 fuelVendorDAO.add(fuelVendor);  
 **addVendorResultLabel**.setText(**"Запис про постачальника успішно додано!"**);  
 **addVendorResultLabel**.setTextFill(Color.*web*(**"#00FF00"**));  
  
 **if** (!**vendorTable**.getItems().isEmpty())  
 **vendorTable**.setItems(FXCollections.*observableList*(fuelVendorDAO.findAll()));  
  
 **companyTitleTextField**.clear();  
 **secondNameTextField**.clear();  
 **firstNameTextField**.clear();  
 **middleNameTextField**.clear();  
 **priceTextField**.clear();  
 **addressTextField**.clear();  
 **contractTextField**.clear();  
 **yyyyContractTextField1**.clear();  
 **mmContractTextField1**.clear();  
 **ddContractTextField1**.clear();  
 **yyyyContractTextField2**.clear();  
 **mmContractTextField2**.clear();  
 **ddContractTextField1**.clear();  
  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка при введені дати!"**);  
 }  
 }  
  
 **private void** showFuelRecords() {  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 **fuels** = fuelDAO.findAll();  
 **if** (**fuels**.isEmpty()) {  
 **alertWindow**.errorAlert(**"Не знайдено жодного запису!"**);  
 **return**;  
 }  
 **fuelTable**.getItems().clear();  
 **fuelTable**.setItems(FXCollections.*observableList*(**fuels**));  
 }  
 **catch** (Exception ex) {  
 **alertWindow**.errorAlert(ex.getMessage());  
 }  
 }  
  
 **private void** addFuelRecord() {  
  
 **if**(**fuelTypeTextField**.getText().isEmpty() || **manufacturerTextField**.getText().isEmpty() || **yyyyTextField**.getText().isEmpty() ||  
 **mmTextField**.getText().isEmpty() || **ddTextField**.getText().isEmpty() || **priceTextField**.getText().isEmpty() ||  
 **totalVolumeTextField**.getText().isEmpty()) {  
 **alertWindow**.errorAlert(**"Помилка! Одне або декілька полів пусті!"**);  
 **return**;  
 }  
  
 **if**(**selectVendorChoiceBox**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть постачальника!"**);  
 **return**;  
 }  
  
 **try** {  
 String vendorCompany = **selectVendorChoiceBox**.getValue();  
 **addFuelResultLabel**.setText(**""**);  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
  
 Fuel fuel = **new** Fuel();  
 fuel.setType(**fuelTypeTextField**.getText().trim());  
 fuel.setManufacturer(**manufacturerTextField**.getText().trim());  
 fuel.setDateOfSupply(LocalDate.*parse*(**yyyyTextField**.getText().trim() + **"-"** + **mmTextField**.getText().trim() + **"-"** + **ddTextField**.getText().trim()));  
 fuel.setPriceOfSupply(Double.*parseDouble*(**priceTextField**.getText().trim()));  
 fuel.setTotalVolume(Double.*parseDouble*(**totalVolumeTextField**.getText().trim()));  
 *//fuel.setAvailableVolume(Double.parseDouble(availableVolumeTextField.getText().trim()));* fuel.setAvailableVolume(Double.*parseDouble*(**totalVolumeTextField**.getText().trim()));  
 fuelDAO.add(fuel);  
 List<FuelVendor> fuelVendors = fuelVendorDAO.findByCompanyTitle(vendorCompany);  
 fuel.getVendors().add(fuelVendors.get(0));  
 fuelDAO.add(fuel);  
  
 **addFuelResultLabel**.setText(**"Запис про пальне успішно додано!"**);  
 **addFuelResultLabel**.setTextFill(Color.*web*(**"#00FF00"**));  
 **if** (!**fuelTable**.getItems().isEmpty())  
 **fuelTable**.setItems(FXCollections.*observableList*(fuelDAO.findAll()));  
  
 **fuelTypeTextField**.clear();  
 **manufacturerTextField**.clear();  
 **yyyyTextField**.clear(); **mmTextField**.clear(); **ddTextField**.clear();  
 **priceTextField**.clear();  
 **totalVolumeTextField**.clear();  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Переверте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі, правильний формат (рррр-мм-дд)."**);  
 }  
  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
  
 **private void** setCellValueFactories() {  
 **idFuelColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **fuelTypeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"type"**));  
 **manufacturerColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"manufacturer"**));  
 **dateOfSupplyColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"dateOfSupply"**));  
 **priceOfSupplyColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"priceOfSupply"**));  
 **totalVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"totalVolume"**));  
 **availableVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"availableVolume"**));  
  
 **idVendorColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **companyTitleColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"companyTitle"**));  
 **contactPersonColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"contactPerson"**));  
 **contactPhoneColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"phone"**));  
 **addressColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"address"**));  
 **contractColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"contract"**));  
 **dateOfNegotiationColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"dateOfNegotiation"**));  
 **endDateColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"endDate"**));  
  
 **idPSColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **addressPSColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"address"**));  
 **contactPhonePSColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"phone"**));  
 **contactPersonPSColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"contactPerson"**));  
 **fuelTypePSColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 **totalFuelVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"totalFuelVolume"**));  
 **soldFuelVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"soldVolume"**));  
 **supplyDateColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"dateOfSupply"**));  
 **salePriceColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"salePrice"**));  
 }  
  
 **private void** loadReportWindow(String url, String window) {  
 Stage st = **new** Stage();  
 Parent root = **null**;  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(url));  
  
 **try** {  
 root = loader.load();  
 } **catch** (IOException e) {  
 **alertWindow**.errorAlert(e.getMessage());  
 e.printStackTrace();  
 }  
  
 **if** (window.equals(**"fuelBalanceReport"**)) {  
 BalanceReportWindow balanceReportWindow = loader.getController();  
 balanceReportWindow.setEntityManagerFactory(**entityManagerFactory**);  
 } **else if** (window.equals(**"fuelTransferReport"**)) {  
 FuelTransferReportWindow fuelTransferReportWindow = loader.getController();  
 fuelTransferReportWindow.setEntityManagerFactory(**entityManagerFactory**);  
 } **else if** (window.equals(**"vendorReport"**)) {  
 VendorReportWindow vendorReportWindow = loader.getController();  
 vendorReportWindow.setEntityManagerFactory(**entityManagerFactory**);  
 } **else if** (window.equals(**"Waybill"**)) {  
 WaybillWindow waybillWindow = loader.getController();  
 waybillWindow.setEntityManagerFactory(**entityManagerFactory**);  
 }  
  
 st.setTitle(**"Управління АЗС"**);  
 Scene scene = **new** Scene(root);  
 st.setScene(scene);  
 st.getIcons().add(**new** Image(**"img/barrel.png"**));  
 st.showAndWait();  
 }  
}

WaybillWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeParseException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.FuelDAO;  
**import** dao.PetrolStationDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** javafx.scene.paint.Color;  
**import** reports.CardReport;  
**import** reports.Waybill;  
**import** tables.Fuel;  
**import** tables.PetrolStation;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**import static** java.lang.Math.*abs*;  
  
**public class** WaybillWindow {  
  
 **public class** FuelVolumeTable {  
 **private** String **fuelType**;  
 **private double availableVolume**;  
  
 **public** FuelVolumeTable(String fuelType, **double** availableVolume) {  
 **this**.**fuelType** = fuelType;  
 **this**.**availableVolume** = availableVolume;  
 }  
  
 **public** String getFuelType() {  
 **return fuelType**;  
 }  
  
 **public void** setFuelType(String fuelType) {  
 **this**.**fuelType** = fuelType;  
 }  
  
 **public double** getAvailableVolume() {  
 **return availableVolume**;  
 }  
  
 **public void** setAvailableVolume(**double** availableVolume) {  
 **this**.**availableVolume** = availableVolume;  
 }  
 }  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** TextField **phoneTextField**;  
 @FXML **private** TableView<FuelVolumeTable> **fuelsTable**;  
 @FXML **private** TableColumn<FuelVolumeTable, String> **fuelTypeFuelsColumn**;  
 @FXML **private** TableColumn<FuelVolumeTable, Double> **availableVolumeColumn**;  
 @FXML **private** Button **showAvailableFuelButton**;  
 @FXML **private** TextField **inputFuelVolumeTextField**;  
 @FXML **private** Label **fuelSelectionStatusLabel**;  
 @FXML **private** Button **selectFuelButton**;  
 @FXML **private** RadioButton **choseRecordRadioButton**;  
 @FXML **private** RadioButton **inputDataRadioButton**;  
 @FXML **private** ChoiceBox<String> **selectPetrolStationChoiceBox**;  
 @FXML **private** TextField **addressTextField**;  
 @FXML **private** TextField **contactPersonTextField**;  
 @FXML **private** TextField **ddSupplyDateTextField**;  
 @FXML **private** TextField **mmSupplyDateTextField**;  
 @FXML **private** TextField **yyyySupplyDateTextField**;  
 @FXML **private** Button **addFuelToWaybillButton**;  
 @FXML **private** Button **createWaybillButton**;  
 @FXML **private** TableView<Waybill> **waybillTable**;  
 @FXML **private** TableColumn<Waybill, String> **fuelTypeWaybillColumn**;  
 @FXML **private** TableColumn<Waybill, Double> **fuelVolumeWaybillColumn**;  
 @FXML **private** TableColumn<Waybill, String> **phoneWaybillColumn**;  
 @FXML **private** TableColumn<Waybill, String> **contactPersonWaybillColumn**;  
 @FXML **private** TableColumn<Waybill, LocalDate> **supplyDateWaybillColumn**;  
 @FXML **private** TableColumn<Waybill, String> **addressWaybillColumn**;  
  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private boolean fuelIsChosen** = **false**;  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
 **private** ToggleGroup **group**;  
 **private** List<Waybill> **waybillItems** = **new** ArrayList<>();  
 **private** String **fuelType**;  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
  
 **selectPetrolStationChoiceBox**.setDisable(**true**);  
 **addressTextField**.setDisable(**true**);  
 **phoneTextField**.setDisable(**true**);  
 **contactPersonTextField**.setDisable(**true**);  
*// ddSupplyDateTextField.setDisable(true);  
// mmSupplyDateTextField.setDisable(true);  
// yyyySupplyDateTextField.setDisable(true);* **group** = **new** ToggleGroup();  
 **choseRecordRadioButton**.setToggleGroup(**group**);  
 **inputDataRadioButton**.setToggleGroup(**group**);  
  
 **showAvailableFuelButton**.setOnAction(event -> showAvailableFuel());  
 **selectFuelButton**.setOnAction(event -> selectFuel());  
 **addFuelToWaybillButton**.setOnAction(event -> addFuelToWaybill());  
 **createWaybillButton**.setOnAction(event -> createWaybill());  
  
 **selectPetrolStationChoiceBox**.setOnMousePressed(event -> setPetrolStationChoiceBoxItems());  
  
 **group**.selectedToggleProperty().addListener((observable, oldValue, newValue) -> {  
 RadioButton selectedButton = (RadioButton) newValue;  
 **if** (selectedButton.getText().equals(**"Обрати запис"**)) {  
 **selectPetrolStationChoiceBox**.setDisable(**false**);  
 **addressTextField**.setDisable(**true**);  
 **phoneTextField**.setDisable(**true**);  
 **contactPersonTextField**.setDisable(**true**);  
 }  
 **if** (selectedButton.getText().equals(**"Ввести дані АЗС"**)) {  
 **selectPetrolStationChoiceBox**.setDisable(**true**);  
 **addressTextField**.setDisable(**false**);  
 **phoneTextField**.setDisable(**false**);  
 **contactPersonTextField**.setDisable(**false**);  
 }  
 });  
 }  
  
 **private void** createWaybill() {  
 **if** (**waybillItems**.isEmpty()) {  
 **alertWindow**.errorAlert(**"Помилка! У накладній немає записів!"**);  
 **return**;  
 }  
  
 **try** {  
 **alertWindow**.infoAlert(**"Видаткову накладну створено!"**);  
 showAvailableFuel();  
 **fuelSelectionStatusLabel**.setText(**"запис пального для накладної не обрано"**);  
 **fuelSelectionStatusLabel**.setTextFill(Color.*web*(**"#ff0000"**));  
  
 } **catch** (Exception e) {  
 **alertWindow**.errorAlert(e.getMessage());  
 }  
  
 }  
  
 **private void** setPetrolStationChoiceBoxItems() {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 ObservableList<String> companies = FXCollections.*observableArrayList*(petrolStationDAO.getPetrolStationAddresses());  
 **selectPetrolStationChoiceBox**.setItems(companies);  
 }  
  
 **private void** addFuelToWaybill() {  
 **if** (!**fuelIsChosen**) {  
 **alertWindow**.errorAlert(**"Помилка! Щоб додати пальне до накладної його треба спочатку обрати."**);  
 **return**;  
 }  
  
 **if** (**choseRecordRadioButton** == **null** && **inputDataRadioButton** == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Необхідно вказати АЗС для відправки!"**);  
 **return**;  
 }  
  
 RadioButton selectedButton = (RadioButton) **group**.getSelectedToggle();  
 **if** (selectedButton.getText().equals(**"Обрати запис"**) && **selectPetrolStationChoiceBox**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть зі спису АЗС для відправки!"**);  
 **return**;  
 }  
  
 **else if** ( selectedButton.getText().equals(**"Ввести дані АЗС"**)  
 && (**phoneTextField**.getText().isEmpty() || **addressTextField**.getText().isEmpty()  
 || **contactPersonTextField**.getText().isEmpty())) {  
 **alertWindow**.errorAlert(**"Помилка! Одне або декілька полів даних про АЗС-отримувача пусті!"**);  
 **return**;  
 }  
  
 **if** (**yyyySupplyDateTextField**.getText().isEmpty() || **mmSupplyDateTextField**.getText().isEmpty()  
 || **ddSupplyDateTextField**.getText().isEmpty()) {  
 **alertWindow**.errorAlert(**"Помилка! Введіть дату поставки"**);  
 **return**;  
 }  
  
 **try** {  
 **if** (selectedButton.getText().equals(**"Обрати запис"**)) {  
  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 String choiceBoxValue = **selectPetrolStationChoiceBox**.getValue();  
 PetrolStation petrolStation = petrolStationDAO.findPetrolStationTargetAddress(choiceBoxValue).get(0);  
  
 **waybillItems**.add(**new** Waybill(  
 LocalDate.*parse*(**yyyySupplyDateTextField**.getText().trim() + **"-"** + **mmSupplyDateTextField**.getText().trim() +  
 **"-"** + **ddSupplyDateTextField**.getText().trim()),  
 **fuelType**,  
 Double.*parseDouble*(**inputFuelVolumeTextField**.getText().trim()),  
 petrolStation.getAddress(),  
 petrolStation.getPhone(),  
 petrolStation.getContactPerson()));  
 }  
  
 **else** {  
 **waybillItems**.add(**new** Waybill(  
 LocalDate.*parse*(**yyyySupplyDateTextField**.getText().trim() + **"-"** + **mmSupplyDateTextField**.getText().trim() +  
 **"-"** + **ddSupplyDateTextField**.getText().trim()),  
 **fuelType**,  
 Double.*parseDouble*(**inputFuelVolumeTextField**.getText().trim()),  
 **addressTextField**.getText().trim(),  
 **phoneTextField**.getText().trim(),  
 **contactPersonTextField**.getText().trim()));  
 }  
 **waybillTable**.setItems(FXCollections.*observableArrayList*(**waybillItems**));  
 **waybillTable**.refresh();  
 **fuelIsChosen** = **false**;  
 **fuelType** = **null**;  
 **fuelSelectionStatusLabel**.setText(**"запис пального для накладної не обрано"**);  
 **fuelSelectionStatusLabel**.setTextFill(Color.*web*(**"#ff0000"**));  
  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Перевірте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введену у неправильному форматі."**);  
 }  
 }  
  
 **private void** selectFuel() {  
 **try** {  
 **fuelType** = **fuelsTable**.getSelectionModel().getSelectedItem().getFuelType();  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Ви повинні обрати запис з таблиці!"**);  
 **return**;  
 }  
 **if**(**inputFuelVolumeTextField**.getText().isEmpty()) {  
 **alertWindow**.errorAlert(**"Помилка! Потрібно ввести об'єм пального для відправки."**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 **double** volume = Double.*parseDouble*(**inputFuelVolumeTextField**.getText().trim());  
 List<Fuel> fuelList = **new** ArrayList<>(fuelDAO.getAvailableFuelByType(**fuelType**));  
 **for** (Fuel fuel : fuelList) {  
 **if** (*abs*(volume) < 0.001) {  
 **alertWindow**.infoAlert(**"Невистачає пального!"**);  
 **return**;  
 }  
  
 **if**(*abs*(fuel.getAvailableVolume() - volume) < 0.1) {  
 fuel.setAvailableVolume(0);  
 fuelDAO.editRecord(fuel);  
 }  
 **else** {  
 volume = fuel.getAvailableVolume() - volume;  
 fuel.setAvailableVolume(volume);  
 fuelDAO.editRecord(fuel);  
 }  
 }  
 **fuelIsChosen** = **true**;  
 **fuelSelectionStatusLabel**.setText(**"Обрано "** + **fuelType**);  
 **fuelSelectionStatusLabel**.setTextFill(Color.*web*(**"00FF00"**));  
 *//fuelType = null;* } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(**"Помилка! Перевірте введені дані."**);  
 } **catch** (NumberFormatException e) {  
 **alertWindow**.errorAlert(**"Помилка! Введено текст замість числа!"**);  
 }  
 }  
  
 **private void** showAvailableFuel() {  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelDAO fuelDAO = **new** FuelDAO(em);  
 List<String> fuelTypes = fuelDAO.getAllAvailableFuelTypes();  
 **if** (fuelTypes.isEmpty()) {  
 **alertWindow**.errorAlert(**"Записи про пальне відсутні! Спочатку треба додати хоча б один запис!"**);  
 **return**;  
 }  
 List<Double> totalVolume = **new** ArrayList<>();  
 List<FuelVolumeTable> tableList = **new** ArrayList<>();  
 **for** (**int** i = 0; i < fuelTypes.size(); i++) {  
 totalVolume.add(fuelDAO.getTotalAvailableFuelVolume(fuelTypes.get(i)));  
 tableList.add(**new** FuelVolumeTable(fuelTypes.get(i), totalVolume.get(i)));  
 }  
  
 **fuelsTable**.setItems(FXCollections.*observableArrayList*(tableList));  
  
 } **catch** (Exception e) {  
 **alertWindow**.errorAlert(e.getMessage());  
 }  
 }  
  
 **private void** setCellValueFactories() {  
 **fuelTypeFuelsColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 **availableVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"availableVolume"**));  
  
 **fuelTypeWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 **contactPersonWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"contactPerson"**));  
 **fuelVolumeWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelVolume"**));  
 **addressWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"petrolStationAddress"**));  
 **phoneWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"phone"**));  
 **supplyDateWaybillColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"date"**));  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
}

VendorReportWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeParseException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.FuelVendorDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.ChoiceBox;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** reports.VendorReport;  
**import** tables.Fuel;  
**import** tables.FuelVendor;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** VendorReportWindow {  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** ChoiceBox<String> **fuelVendorChoiceBox**;  
 @FXML **private** TextField **ddTextField1**;  
 @FXML **private** TextField **mmTextField1**;  
 @FXML **private** TextField **yyyyTextField1**;  
 @FXML **private** TextField **ddTextField2**;  
 @FXML **private** TextField **mmTextField2**;  
 @FXML **private** TextField **yyyyTextField2**;  
 @FXML **private** Button **createFuelVendorsReportButton**;  
 @FXML **private** TableView<VendorReport> **fuelVendorReportTable**;  
 @FXML **private** TableColumn<VendorReport, String> **companyTitleColumn**;  
 @FXML **private** TableColumn<VendorReport, String> **fuelTypeColumn**;  
 @FXML **private** TableColumn<VendorReport, LocalDate> **supplyDateColumn**;  
 @FXML **private** TableColumn<VendorReport, Double> **supplyPriceColumn**;  
 @FXML **private** TableColumn<VendorReport, Double> **supplyVolumeColumn**;  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **createFuelVendorsReportButton**.setOnAction(event -> createFuelVendorsReport());  
 **fuelVendorChoiceBox**.setOnMousePressed(event -> setChoiceBoxItems());  
 }  
  
 **private void** setChoiceBoxItems() {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 ObservableList<String> companies = FXCollections.*observableArrayList*(fuelVendorDAO.getVendorsCompany());  
 companies.add(0, **"Усі постачальники"**);  
 **fuelVendorChoiceBox**.setItems(companies);  
 }  
  
 **private void** createFuelVendorsReport() {  
 **if** (**fuelVendorChoiceBox**.getValue() == **null** ) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть постачальника зі списку списку!"**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 FuelVendorDAO fuelVendorDAO = **new** FuelVendorDAO(em);  
 List<FuelVendor> fuelVendors;  
  
 LocalDate from = LocalDate.*parse*(**yyyyTextField1**.getText().trim() + **"-"** + **mmTextField1**.getText().trim()  
 + **"-"** + **ddTextField1**.getText().trim());  
 LocalDate to = LocalDate.*parse*(**yyyyTextField2**.getText().trim() + **"-"** + **mmTextField2**.getText()  
 + **"-"** + **ddTextField2**.getText().trim());  
  
 **if** (to.isBefore(from)) {  
 **alertWindow**.errorAlert(**"Помилка при введені дати! Вказана кінцева дата раніше початккової дати!"**);  
 }  
  
 **if** (**fuelVendorChoiceBox**.getValue().equals(**"Усі постачальники"**))  
 fuelVendors = fuelVendorDAO.findAllVendorsWithSupplies();  
 **else** {  
 fuelVendors = fuelVendorDAO.findByCompanyTitle(**fuelVendorChoiceBox**.getValue());  
 **if** (fuelVendors.isEmpty()) {  
 **alertWindow**.infoAlert(**"Цей постачальник ще нічего не постачає!"**);  
 **return**;  
 }  
 }  
  
 **if**(fuelVendors.isEmpty()) {  
 **alertWindow**.infoAlert(**"Не зайдено жодного постачальника з поставками!"**);  
 **return**;  
 }  
  
 List<VendorReport> report = **new** ArrayList<>();  
 **for** (FuelVendor fuelVendor : fuelVendors) {  
 **for** (Fuel fuel : fuelVendor.getFuels()) {  
 **boolean** p1 = fuel.getDateOfSupply().isAfter(from.minusDays(1));  
 **boolean** p2 = fuel.getDateOfSupply().isBefore(to.plusDays(1));  
 **if** ( p1 && p2)  
 report.add(**new** VendorReport(fuelVendor.getCompanyTitle(),  
 fuel.getType(), fuel.getPriceOfSupply(), fuel.getDateOfSupply(), fuel.getTotalVolume()));  
 }  
 }  
 **fuelVendorReportTable**.setItems(FXCollections.*observableArrayList*(report));  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(e.getMessage());  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введено неправильно, перевірте введення!"**);  
 }  
 }  
  
 **private void** setCellValueFactories() {  
 **companyTitleColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"companyTitle"**));  
 **supplyDateColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"supplyDate"**));  
 **fuelTypeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 **supplyPriceColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"supplyPrice"**));  
 **supplyVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"supplyVolume"**));  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
}

ClientReportWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.ClientDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** reports.ClientReport;  
**import** tables.Client;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** ClientReportWindow {  
  
 @FXML  
 **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** TableView<ClientReport> **clientReportTable**;  
 @FXML **private** TableColumn<ClientReport, Integer> **idColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **secondNameColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **firstNameColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **middleNameColumn**;  
 @FXML **private** TableColumn<ClientReport, LocalDate> **birthdayColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **phoneColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **emailColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **carModelColumn**;  
 @FXML **private** TableColumn<ClientReport, String> **fuelTypeColumn**;  
 @FXML **private** Button **clientReportButton**;  
  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** EntityManager **entityManager**;  
 **private** ClientDAO **clientDAO**;  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **clientReportButton**.setOnAction(event -> createClientReport());  
 }  
  
 **private void** createClientReport() {  
 **entityManager** = **entityManagerFactory**.createEntityManager();  
 **clientDAO** = **new** ClientDAO(**entityManager**);  
 List<Client> clients = **clientDAO**.findAll();  
 List<ClientReport> reportList = **new** ArrayList<>();  
 **for** (Client client : clients) {  
 reportList.add(**new** ClientReport(client.getId(), client.getSecondName(), client.getFirstName(), client.getMiddleName(),  
 client.getBirthday(), client.getPhone(), client.getEmail(), client.getCarModel(), client.getFuelType()));  
 }  
 **clientReportTable**.setItems(FXCollections.*observableList*(reportList));  
 }  
  
 **private void** setCellValueFactories() {  
 **idColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **secondNameColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"secondName"**));  
 **firstNameColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"firstName"**));  
 **middleNameColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"middleName"**));  
 **birthdayColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"birthday"**));  
 **phoneColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"phone"**));  
 **emailColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"email"**));  
 **carModelColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"carModel"**));  
 **fuelTypeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
}

FuelTransferReportWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.time.format.DateTimeParseException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.PetrolStationDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.ChoiceBox;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** reports.FuelTransferReport;  
**import** tables.PetrolStation;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** FuelTransferReportWindow {  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** ChoiceBox<String> **petrolStationChoiceBox**;  
 @FXML **private** TextField **ddTextField1**;  
 @FXML **private** TextField **mmTextField1**;  
 @FXML **private** TextField **yyyyTextField1**;  
 @FXML **private** TextField **ddTextField2**;  
 @FXML **private** TextField **mmTextField2**;  
 @FXML **private** TextField **yyyyTextField2**;  
 @FXML **private** Button **createTransferReportButton**;  
 @FXML **private** TableView<FuelTransferReport> **fuelVendorReportTable**;  
 @FXML **private** TableColumn<FuelTransferReport, Integer> **idPetrolStationColumn**;  
 @FXML **private** TableColumn<FuelTransferReport, String> **petrolStationAddress**;  
 @FXML **private** TableColumn<FuelTransferReport, Double> **supplyVolumeColumn**;  
 @FXML **private** TableColumn<FuelTransferReport, LocalDate> **supplyDateColumn**;  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **petrolStationChoiceBox**.setOnMousePressed(event -> setChoiceBoxItems());  
 **createTransferReportButton**.setOnAction(event -> createTransferReport());  
 }  
  
 **private void** createTransferReport() {  
 **if** (**petrolStationChoiceBox**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Пимилка! Оберіть АЗС зі списку!"**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 List<PetrolStation> petrolStations;  
  
 LocalDate from = LocalDate.*parse*(**yyyyTextField1**.getText().trim() + **"-"** + **mmTextField1**.getText().trim()  
 + **"-"** + **ddTextField1**.getText().trim());  
 LocalDate to = LocalDate.*parse*(**yyyyTextField2**.getText().trim() + **"-"** + **mmTextField2**.getText()  
 + **"-"** + **ddTextField2**.getText().trim());  
  
 **if** (to.isBefore(from)) {  
 **alertWindow**.errorAlert(**"Помилка при введені дати! Вказана кінцева дата раніше початккової дати!"**);  
 }  
  
 **if** (**petrolStationChoiceBox**.getValue().equals(**"Усі АЗС"**))  
 petrolStations = petrolStationDAO.findAll();  
 **else** {  
 petrolStations = petrolStationDAO.findPetrolStationTargetAddress(**petrolStationChoiceBox**.getValue());  
 **if** (petrolStations.isEmpty()) {  
 **alertWindow**.infoAlert(**"Не знайдено жодного запису про поставки на цю АЗС"**);  
 **return**;  
 }  
 }  
  
 **if**(petrolStations.isEmpty()) {  
 **alertWindow**.infoAlert(**"Не зайдено жодної АЗС!"**);  
 **return**;  
 }  
  
 List<FuelTransferReport> fuelTransferReport = **new** ArrayList<>();  
 **for** (PetrolStation petrolStation : petrolStations) {  
 **boolean** p1 = petrolStation.getDateOfSupply().isAfter(from.minusDays(1));  
 **boolean** p2 = petrolStation.getDateOfSupply().isBefore(to.plusDays(1));  
 **if** ( p1 && p2)  
 fuelTransferReport.add(**new** FuelTransferReport(petrolStation.getId(), petrolStation.getAddress(),  
 petrolStation.getTotalFuelVolume(), petrolStation.getDateOfSupply()));  
 }  
 **fuelVendorReportTable**.setItems(FXCollections.*observableArrayList*(fuelTransferReport));  
 } **catch** (NullPointerException e) {  
 **alertWindow**.errorAlert(e.getMessage());  
 } **catch** (DateTimeParseException e) {  
 **alertWindow**.errorAlert(**"Помилка! Дату введено неправильно, перевірте введення!"**);  
 }  
 }  
  
 **private void** setChoiceBoxItems() {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 ObservableList<String> companies = FXCollections.*observableArrayList*(petrolStationDAO.getPetrolStationAddresses());  
 companies.add(0, **"Усі АЗС"**);  
 **petrolStationChoiceBox**.setItems(companies);  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
  
 **private void** setCellValueFactories() {  
 **idPetrolStationColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **supplyDateColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"supplyDate"**));  
 **petrolStationAddress**.setCellValueFactory(**new** PropertyValueFactory<>(**"address"**));  
 **supplyVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"supplyVolume"**));  
 }  
}

CardReportWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.time.LocalDate;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.ClientDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** reports.CardReport;  
**import** tables.Client;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** CardReportWindow {  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** TableView<CardReport> **clientReportTable**;  
 @FXML **private** TableColumn<CardReport, Integer> **idColumn**;  
 @FXML **private** TableColumn<CardReport,String> **fullNameColumn**;  
 @FXML **private** TableColumn<CardReport, Integer> **cardNumColumn**;  
 @FXML **private** TableColumn<CardReport, LocalDate> **issuanceDateColumn**;  
 @FXML **private** TableColumn<CardReport, Double> **discountPercentColumn**;  
 @FXML **private** TableColumn<CardReport, Double> **boughtVolumeColumn**;  
 @FXML **private** Button **cardReportButton**;  
  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** EntityManager **entityManager**;  
 **private** ClientDAO **clientDAO**;  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **cardReportButton**.setOnAction(event -> createCardReport());  
 }  
  
 **private void** createCardReport() {  
 **entityManager** = **entityManagerFactory**.createEntityManager();  
 **clientDAO** = **new** ClientDAO(**entityManager**);  
 List<Client> clients = **clientDAO**.findAll();  
 List<CardReport> reportList = **new** ArrayList<>();  
 **for** (Client client : clients) {  
 reportList.add(**new** CardReport(client.getId(), client.getSecondName() + **" "** + client.getFirstName() + **" "** + client.getMiddleName(),  
 client.getLoyaltyCardNum(), client.getIssuanceDate(), client.getDiscountPercent(), client.getBoughtVolume()));  
 }  
 **clientReportTable**.setItems(FXCollections.*observableList*(reportList));  
 }  
  
 **private void** setCellValueFactories() {  
 **idColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 **fullNameColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"fullName"**));  
 **cardNumColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"cardNum"**));  
 **issuanceDateColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"issuanceDate"**));  
 **discountPercentColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"discountPercent"**));  
 **boughtVolumeColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"boughtVolume"**));  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.**entityManagerFactory** = entityManagerFactory;  
 }  
}

BalanceReportWindow.java

**package** sample;  
  
**import** java.net.URL;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.ResourceBundle;  
  
**import** dao.PetrolStationDAO;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.control.ChoiceBox;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** reports.FuelBalanceReport;  
**import** tables.PetrolStation;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.EntityManagerFactory;  
  
**public class** BalanceReportWindow {  
  
 @FXML **private** ResourceBundle **resources**;  
 @FXML **private** URL **location**;  
 @FXML **private** ChoiceBox<String> **petrolStationAddressChoiceBox**;  
 @FXML **private** Button **createFuelBalanceReportButton**;  
 @FXML **private** TableView<FuelBalanceReport> **fuelBalanceReportTable**;  
 @FXML **private** TableColumn<FuelBalanceReport, Integer> **petrolStationIDColumn**;  
 @FXML **private** TableColumn<FuelBalanceReport, String> **petrolStationAddressColumn**;  
 @FXML **private** TableColumn<FuelBalanceReport, String> **fuelTypeColumn**;  
 @FXML **private** TableColumn<FuelBalanceReport, Double> **availableVolumeColumn**;  
  
 **private** EntityManagerFactory **entityManagerFactory**;  
 **private** AlertWindow **alertWindow** = **new** AlertWindow();  
  
 @FXML  
 **void** initialize() {  
 setCellValueFactories();  
 **createFuelBalanceReportButton**.setOnAction(event -> createFuelBalanceReport());  
 **petrolStationAddressChoiceBox**.setOnMousePressed(event -> setChoiceBoxItems());  
 }  
  
 **private void** setChoiceBoxItems() {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 ObservableList<String> addresses = FXCollections.*observableArrayList*(petrolStationDAO.getAddresses());  
 addresses.add(0, **"Усі АЗС"**);  
 **petrolStationAddressChoiceBox**.setItems(addresses);  
 }  
  
 **private void** createFuelBalanceReport() {  
 **if** (**petrolStationAddressChoiceBox**.getValue() == **null**) {  
 **alertWindow**.errorAlert(**"Помилка! Оберіть АЗС зі списку!"**);  
 **return**;  
 }  
  
 **try** {  
 EntityManager em = **entityManagerFactory**.createEntityManager();  
 PetrolStationDAO petrolStationDAO = **new** PetrolStationDAO(em);  
 List<PetrolStation> petrolStations;  
  
 **if** (petrolStationAddressChoiceBox.getValue().equals(**"Усі АЗС"**))  
 petrolStations = petrolStationDAO.balanceReportAll();  
 **else** {  
 petrolStations = petrolStationDAO.balanceReportSingleStation(petrolStationAddressChoiceBox.getValue());  
 }  
  
 **if**(petrolStations.isEmpty()) {  
 alertWindow.infoAlert(**"Залишки пального на АЗС відсутні!"**);  
 **return**;  
 }  
  
 List<FuelBalanceReport> report = **new** ArrayList<>();  
 **for** (PetrolStation petrolStation : petrolStations) {  
 report.add(**new** FuelBalanceReport(petrolStation.getId(), petrolStation.getAddress(),  
 petrolStation.getFuelType(), petrolStation.getAvailableVolume()));  
 }  
 fuelBalanceReportTable.setItems(FXCollections.observableArrayList(report));  
 } **catch** (Exception e) {  
 alertWindow.errorAlert(e.getMessage());  
 }  
 }  
  
 **private void** setCellValueFactories() {  
 petrolStationIDColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"petrolStationID"**));  
 petrolStationAddressColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"petrolStationAddress"**));  
 fuelTypeColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"fuelType"**));  
 availableVolumeColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"availableVolume"**));  
 }  
  
 **public void** setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
 **this**.entityManagerFactory = entityManagerFactory;  
 }  
}

AlertWindow.java

**package** sample;  
  
**import** javafx.scene.control.Alert;  
**import** javafx.scene.control.ButtonType;  
**import** javafx.scene.image.Image;  
**import** javafx.stage.Stage;  
  
**import** java.util.Optional;  
  
**public class** AlertWindow {  
  
 **public void** errorAlert(String message){  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Управління АЗС"**);  
 alert.setContentText(message);  
 Stage stage = (Stage)alert.getDialogPane().getScene().getWindow();  
 stage.getIcons().add(**new** Image(**"img/barrel.png"**));  
 alert.setHeaderText(**null**);  
 alert.showAndWait();  
 }  
  
 **public void** infoAlert(String message) {  
 Alert alert = **new** Alert(Alert.AlertType.INFORMATION);  
 alert.setTitle(**"Управління АЗС"**);  
 alert.setContentText(message);  
 Stage stage = (Stage)alert.getDialogPane().getScene().getWindow();  
 stage.getIcons().add(**new** Image(**"img/barrel.png"**));  
 alert.setHeaderText(**null**);  
 alert.showAndWait();  
 }  
  
 **public boolean** confirmationAlert(String objNameToDelete, String objBriefInfo) {  
 Alert alert = **new** Alert(Alert.AlertType.CONFIRMATION);  
 alert.setTitle(**"Управління АЗС"**);  
 Stage stage = (Stage)alert.getDialogPane().getScene().getWindow();  
 stage.getIcons().add(**new** Image(**"img/barrel.png"**));  
 alert.setContentText(String.format(**"Видалити "** + objNameToDelete + **" (%s)?"**, objBriefInfo));  
 alert.setHeaderText(**null**);  
 Optional<ButtonType> buttonType = alert.showAndWait();  
 **if** (ButtonType.OK.equals(buttonType.get()))  
 **return true**;  
 **return false**;  
 }  
}