МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

« » червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Диспетчер систем керування базами даних»
здобувача групи ІН-01 Піскурьова Іллі Віталійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ілля ПІСКУРЬОВ
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук, к.т.н,
доцент Борис КУЗІКОВ _____
(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Піскурьова Іллі Віталійовича

1. Тема роботи: «Диспетчер систем керування базами даних»
   затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «__» _____ 20__ р.

Завдання прийняв на виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Виконати аналіз предметної області та визначити актуальність*		
2	*Провести дослідження існуючих аналогів та виявити їх недоліки*		
3	*Обрати технології для розробки інформаційної системи*		
4	*Виконати розробку інформаційної системи*		
5	*Виконати розробку зручного графічного інтерфейсу*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 59 стр., 10 рис., 1 додаток, 28 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі спрощення та автоматизації використання різноманіття систем керування базами даних шляхом розробки відповідних методів, моделей та програмного забезпечення.

Об'єкт дослідження — процес керування базами даних.

Мета роботи — розробка програми-диспетчера для керування різноманітними системами керування базами даних з використанням контейнеризації Docker для спрощення встановлення, налаштування, оновлення та керування базами даних.

Методи дослідження — аналіз існуючих програм-диспетчерів, проектування архітектури програмного забезпечення, контейнеризація за допомогою Docker, розробка графічного інтерфейсу користувача.

Результати — розроблено програму-диспетчер, яка дозволяє користувачам зручно управляти різними системами керування базами даних, включаючи встановлення, налаштування, запуск та зупинка. Програма використовує контейнеризацію Docker для автоматизації процесів і забезпечення безперервного доступу до баз даних. Проведено тестування розробки на реальних сценаріях використання.

ІНФОРМАЦІЙНА СИСТЕМА, ДИСПЕТЧЕР СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ, СИСТЕМИ КЕРУВАННЯ БАЗАМИ ДАННИХ, JAVA, JAVAFX, DOCKER

ЗМІСТ

ВСТУП

Актуальність. Сучасний світ інформаційних технологій неможливо уявити без систем керування базами даних (далі – СКБД [1]). Бази даних є основним інструментом для зберігання, організації та доступу до великої кількості даних, і вони відіграють критичну роль у багатьох сферах людської діяльності, включаючи розробку програмного забезпечення, управління бізнесом, наукові дослідження, освіту, медицину та інші.

Однією з ключових проблем, з якими стикаються користувачі баз даних різних професій та зайнятості, є необхідність використання різноманіття систем керування базами даних для різних проектів або завдань. Наприклад, як студент, я мав необхідність використовувати чотири й більше різних СКБД. Також при розгортанні вже готових проектів, наприклад з джерел з відкритим програмним кодом, вони можуть залежати від різних СКБД, що робить необхідним їх встановлення. Кожна СКБД може мати свої вимоги до встановлення, конфігурації, оновлення та керування, і це може призвести до втрати часу, поганого досвіду розробника, неочевидних та заплутаних сценаріїв використання.

Таким чином, розробка інформаційної системи, що зробить підхід до управління СКБД більше ефективним та автоматизованим, позитивно вплине на світ інформаційних технологій.

Об’єкт дослідження. Процес керування базами даних.

Предмет дослідження. Методологія автоматизації управління різноманітними СКБД з використанням програми-диспетчера.

Гіпотеза. Автоматизація процесів встановлення, налаштування, оновлення та керування СКБД може бути досягнута шляхом розробки програми-диспетчера, яка використовує контейнеризацію Docker [2] для спрощення цих процесів і підвищення ефективності роботи користувачів.

Новизна. На відміну від існуючих аналогів, запропонована у даній роботі програма-диспетчер дозволить користувачам автоматизувати процеси керування різноманітними СКБД, використовуючи контейнеризацію Docker.

Це забезпечить швидкий доступ до необхідних баз даних, спростить встановлення та налаштування, а також запропонує зручний графічний інтерфейс для взаємодії з користувачами. Розробка забезпечить безперервний доступ до баз даних і покращить загальний досвід користувачів.

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан диспетчерів або систем запуску та управління програмами

Сучасна система запуску та управління програмами або «диспетчер» - це інструмент, що надає користувачам можливість зручно відкривати та керувати різними програмами, додатками або службами, які встановлені на їх пристрої. Диспетчери призначені для спрощення та організації робочого процесу користувача, допомагаючи швидко знайти та запустити необхідний програмний продукт, а також можуть забезпечувати інші функції, такі як пошук, сортування та групування програм. Таким чином, вони роблять використання комп'ютера більш зручним та продуктивним, допомагаючи користувачам зосереджуватися на важливих завданнях та досягати кращих результатів.

1.2 Аналіз аналогічних проектів

JetBrains Toolbox (див Рисунок 1.1) - це інструмент для керування та оновлення інструментів розробки, які розробляється компанією JetBrains [3]. Він дозволяє розробникам легко встановлювати, оновлювати та керувати різними інтегрованими середовищами розробки (IDE [4]) та іншими інструментами для програмування. Програма пропонує зручний інтерфейс для управління всіма цими інструментами та надає можливість встановлення та оновлення їх з мінімальними зусиллями з боку розробників.

Цей інструмент наведений до прикладу, оскільки поточне завдання він не вирішує, але гарно демонструє як може виглядати програма-диспетчер, і як такі програми покращують взаємодію з іншими програмними продуктами.

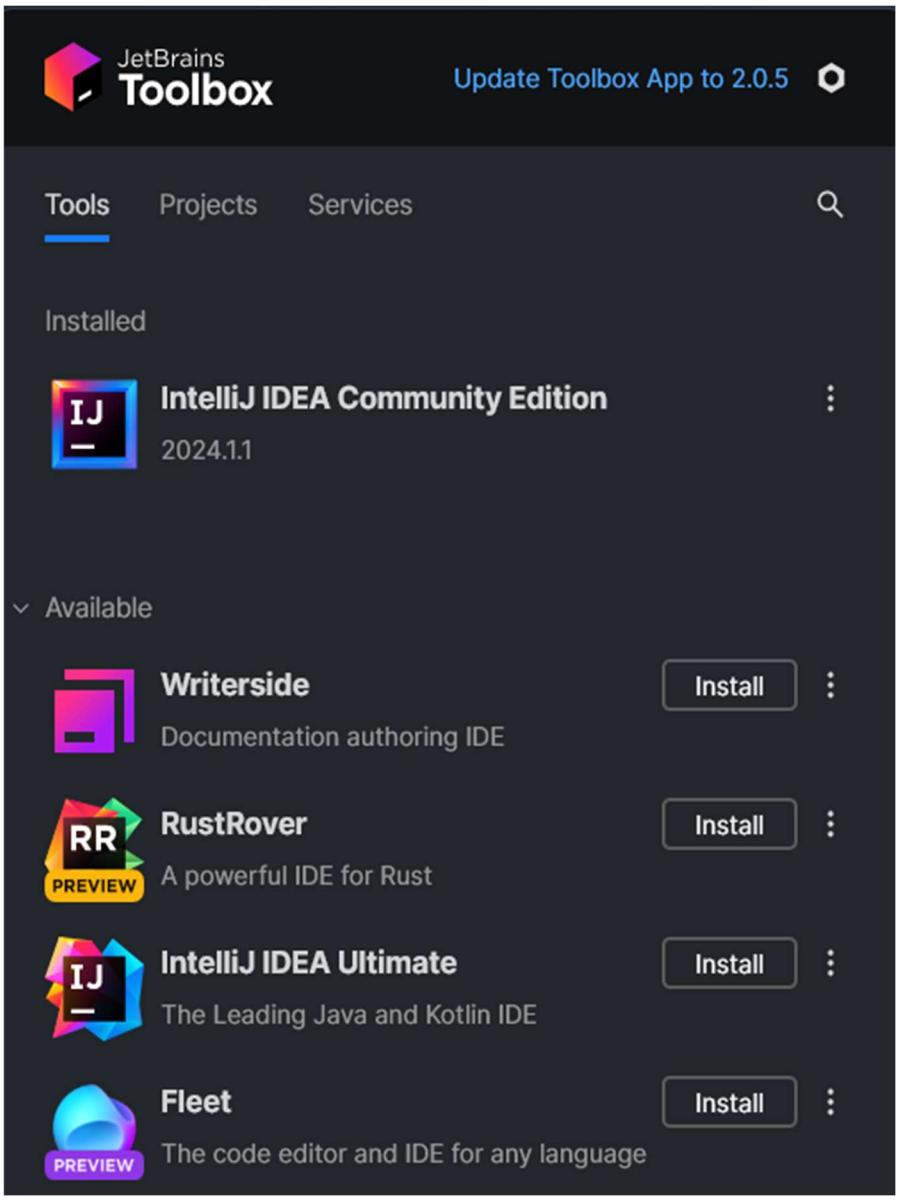

Рисунок 1.1 – JetBrains Toolbox

DBngin (див Рисунок 1.2) є інструментом для запуску та керування СКБД. Він дозволяє легко встановлювати, налаштовувати та запускати різні СКБД, такі як MySQL, PostgreSQL, SQLite, Reddis та багато інших за допомогою графічного інтерфейсу. Також дозволяє перемикатися між різними версіями СКБД та конфігураціями, що допомагає забезпечити сумісність і стабільність програмного забезпечення.

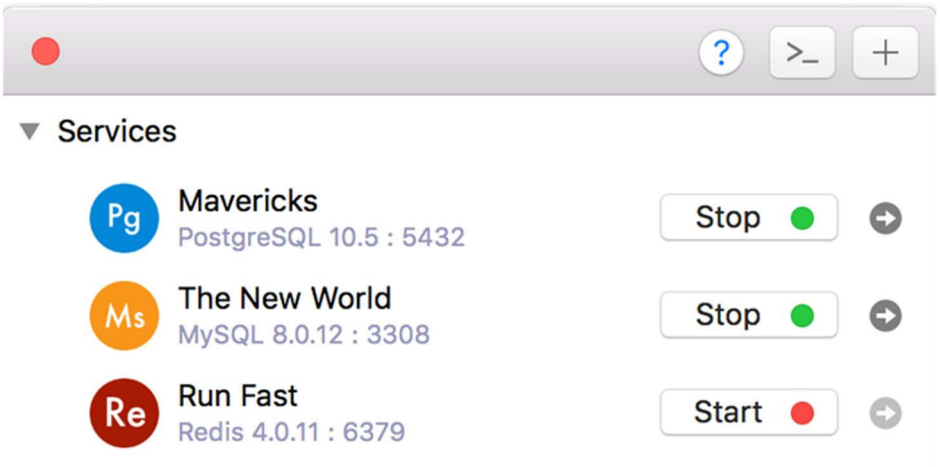

Рисунок 1.2 – DBngin

Недоліком цього інструменту є підтримка лише однієї операційної системи – macOS [5].

1.3 Постановка задачі

Метою бакалаврської роботи, стала розробка програмного забезпечення, яке зробить підхід до управління СКБД більш ефективним та автоматизованим. Воно дозволить користувачам швидко вибирати та запускати необхідні СКБД, автоматично встановлювати налаштування та забезпечувати безперервний доступ до баз даних.

Програмна реалізація програми-диспетчера повинна мати наступний функціонал:

1. Менеджмент СКБД:
    - Програма повинна дозволяти обирати СКБД зі списку підтримуваних.
    - Користувачі повинні мати змогу визначити, чи буде СКБД встановлюватися локально або в контейнері Docker.
2. Запуск СКБД та управління ними:
    - Програма повинна надавати можливість користувачам запускати обрані СКБД.

- Користувачам повинна бути доступна функціональність управління запущеними СКБД (запуск, зупинка, перезавантаження).

3. Підтримка операційної системи Windows [6].
4. Читання конфігурації у форматі JSON [7].

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Виконати аналіз предметної області та визначити актуальність
2. Провести дослідження існуючих аналогів та виявити їх недоліки
3. Обрати технології для розробки інформаційної системи
4. Виконати розробку інформаційної системи
5. Виконати розробку зручного графічного інтерфейсу

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель

З метою вирішення проблеми є доцільним створення desktop-додатку [8]. Desktop-додаток (іноді також називають настільним додатком або програмою для робочого столу) - це програмне забезпечення, призначене для виконання на комп'ютерах або ноутбуках, які працюють під управлінням операційних систем, таких як Windows, macOS або Linux. Основна відмінність desktop-додатків від веб-додатків полягає в тому, що вони встановлюються на локальному пристрої користувача і запускаються незалежно від браузера та доступу в інтернет. Для цього треба обрати інструментарій графічного інтерфейсу користувача.

Основним способом встановлення СКБД буде використання Docker-образів, що забезпечить стандартизацію, ізоляцію середовищ, легкість розгортання та зменшення конфліктів між різними СКБД. Використання контейнеризації Docker дозволить автоматизувати процеси встановлення та конфігурації СКБД, зменшити ризики помилок та підвищити ефективність управління базами даних.

2.2 Інструментарій графічного інтерфейсу користувача

Інструменти графічного інтерфейсу користувача - це набір програмних інструментів, бібліотек і компонентів, призначених для розробки графічних користувацьких інтерфейсів (GUI [9]) в програмах і додатках. Графічний інтерфейс користувача включає в себе елементи, такі як вікна, кнопки, тексти, полоси прокрутки, меню, віджети та інші компоненти, які взаємодіють з користувачем через мишу, клавіатуру та інші способи введення.

Такі інструменти надають розробникам можливість створювати і налаштовувати вигляд та поведінку графічних інтерфейсів для їхніх програм. Вони також допомагають організувати взаємодією з операційною системою, обробкою подій, створення віджетів і керування графічними об'єктами.

Наведемо перелік популярних інструментів для створення користувацького інтфейсу, що підтримують Windows:

Qt [10]:

- Мова програмування: C++ [11] (але також є обгортки для інших мов, наприклад PyQt для Python).
- Qt - це кросплатформова бібліотека та фреймворк для розробки графічних користувацьких інтерфейсів (GUI) та додатків. Він підтримує розробку як настільних, так і мобільних додатків і надає можливість створювати красивий та функціональний інтерфейс. Qt відомий своєю підтримкою багатьох платформ та продуктивністю.
- Комерційне використання платне.
- Версія, яка безкоштовно доступна для розробки програмного забезпечення з відкритим кодом, відстає на рік від актуальної.
- Qt постачається з власним інтегрованим середовищем розробки (IDE) під назвою Qt Creator [12], яке спрощує процес створення, налагодження та візуального редагування Qt додатків.
- Qt підтримує не тільки різні операційні системи (Windows, Linux, macOS, Android, iOS), але і різні апаратні платформи, включаючи робочі станції, мобільні пристрої, вбудовані системи та інші.
- Qt має власну мову розмітку QML [13] схожу на JavaScript.

JavaFX [14]:

- Мова програмування: Java [15].
- JavaFX - це графічний фреймворк для створення багатофункціональних графічних додатків на мові програмування Java. Він надає розробникам різноманітні інструменти для створення інтерактивних і красивих інтерфейсів. JavaFX може використовуватися як для розробки настільних, так і для мобільних додатків.
- Прийшов на заміну Swing, але більше не є частиною стандартної бібліотеки Java.

- JavaFX спрощує розробку інтерфейсів за допомогою FXML [16], мови розмітки, що дозволяє визначати інтерфейси користувача у вигляді роздільних файлів. Scene Builder [17] - це візуальний редактор для FXML, який дозволяє розміщувати та налаштовувати елементи інтерфейсу користувача за допомогою простого перетягування та розміщення.
- JavaFX має потужні графічні можливості, такі як вбудована підтримка векторної графіки, анімація, ефекти, трансформації та 3D-графіка.
- JavaFX використовує звичну для веб розробників мову стилів Cascading Style Sheets (CSS [18]) для оформлення інтерфейсу користувача, що дозволяє розробникам легко змінювати вигляд та оформлення додатків.
- JavaFX має вбудовану підтримку відображення веб-сторінок за допомогою класу WebView, що дозволяє вбудовувати веб-контент безпосередньо в додатки.
- JavaFX має вбудовану підтримку аудіо та відео, що дозволяє вбудовувати мультимедійний контент безпосередньо в додатки.

WxWidgets [19]:

- Мова програмування: C++ (але також є обгортки для інших мов, наприклад wxPython [20] для Python).
- wxWidgets - це кросплатформова бібліотека для розробки графічних додатків на мові програмування C++. Вона надає інструменти для створення настільних програм з нативним (тобто рідним для платформи) виглядом на різних операційних системах, включаючи Windows, macOS і Linux.
- wxWidgets поширюється під потрійною ліцензією: LGPL, GPL або комерційна. Це означає, що ви можете використовувати wxWidgets у ваших проектах як у вільних, так і у пропрієтарних додатках.

- wxWidgets має велику кількість вбудованих віджетів і контролів для створення різноманітних інтерфейсів користувача, включаючи кнопки, текстові поля, списки, таблиці, діалогові вікна, меню, панелі і багато іншого.
- wxWidgets підтримує не тільки Windows, macOS і Linux, а й інші операційні системи, такі як FreeBSD, OpenBSD, NetBSD, Solaris, OS/2, і Haiku.

Swing [21]:

- Мова програмування: Java.
- Swing - це бібліотека для розробки графічних інтерфейсів на мові програмування Java. Вона надає інструменти для створення десктопних додатків з графічним інтерфейсом. Swing дозволяє розробникам створювати інтерактивні програми для платформи Java.
- Swing є частиною стандартного набору бібліотек Java (Java API), що дозволяє легко інтегрувати її з іншими Java-технологіями.
- Swing розроблена для роботи на будь-якій платформі, що підтримує віртуальну машину Java (JVM [22]), що робить її ідеальною для розробки кросплатформових додатків.
- Вважається застарівшою технологією.
- Навіть як застаріла технологія, Swing все ще отримує підтримку і розвиток в межах JDK (Java Development Kit), але вона вже не є основним напрямком розвитку для графічних інтерфейсів в Java.

Tkinter [23]**:**

- Мова програмування: Python [24].
- Є частиною стандартної бібліотеки Python, що є перевагою
- Tkinter ідеально підходить для швидкого створення простих інтерфейсів користувача для програм, написаних на Python.
- Tkinter підтримується на різних операційних системах, включаючи Windows, macOS і Linux.

- Tkinter підтримує розширення за допомогою сторонніх бібліотек, таких як ttk (Themed Tkinter), яка надає більше стилізованих віджетів.
- Tkinter має добре документовану API та активну спільноту користувачів, що дозволяє швидко знайти відповіді на питання.

Flutter [25]:

- Мова програмування: Dart.
- Flutter - це відкрита кросплатформова SDK (програмний набір розробки) від Google для створення мобільних, веб- та настільних додатків. Він вирізняється своєю високою продуктивністю та можливістю створювати красивий інтерфейс завдяки вбудованим віджетам.
- Flutter дозволяє розробникам створювати один і той же код для різних платформ і мати нативний вигляд та відчуття. Він активно використовується для розробки мобільних додатків для Android і iOS.
- Однією з ключових особливостей Flutter є можливість гарячого перезавантаження, що дозволяє розробникам швидко бачити зміни в реальному часі без необхідності повторного компіляції додатку.
- Flutter надає вбудовану підтримку для імітації матеріального дизайну (Material Design) для Android і дизайну Cupertino для iOS, що дозволяє додаткам мати нативний вигляд і відчуття на обох платформах.
- Використання власного движка малювання Skia [26] дозволяє Flutter досягати високої продуктивності та плавності анімацій навіть на простіших пристроях.
- Flutter надає розробникам широкі можливості для створення різноманітних анімацій, переходів та спеціальних ефектів для покращення користувацького досвіду.

Windows Forms (WinForms):

- Мова програмування: C# або будь-яка мова .NET, така як VB.NET.
- WinForms містить великий набір вбудованих елементів керування, таких як кнопки, тексти, списки, таблиці, меню і т. д., що спрощує розробку інтерфейсу користувача.
- WinForms інтегрована з Visual Studio, найпопулярнішою середовищем розробки для платформи .NET, що полегшує розробку та відлагодження додатків.
- Починаючи з .NET Framework 3.0, WinForms отримала підтримку мови XAML (eXtensible Application Markup Language), що дозволяє створювати графічні інтерфейси за допомогою декларативного синтаксису.

Windows Presentation Foundation (WPF):

- Мова програмування**:** C# або будь-яка мова .NET, така як VB.NET.
- WPF використовує шаблон проектування MVVM [27] (Model-View-ViewModel), що дозволяє розділити логіку додатку від його представлення і забезпечує більшу модульність і тестованість коду.
- WPF використовує мову XAML (eXtensible Application Markup Language) для декларативного описання інтерфейсу користувача, що робить розробку додатків більш зручною і простою.
- WPF підтримує адаптивний дизайн, що дозволяє автоматично адаптувати інтерфейс додатка до різних розмірів і роздільної здатності екранів.

Avalonia:

- Мова програмування: C#.
- Avalonia - це кросплатформова бібліотека і фреймворк для розробки графічних інтерфейсів (GUI) на C#. Він створений з урахуванням кросплатформовості та можливості розробки для Windows, macOS, і Linux.
- По суті є спробою зробити WPF для більшої кількості платформ.

- Основною особливістю Avalonia є те, що він намагається надати нативний вигляд і відчуття для різних платформ, але залишається кросплатформовим, дозволяючи розробникам використовувати C# для створення GUI-додатків.
- Надає схожий на WPF досвід використання, але при цьому доступний не лише для Windows.
- Аналогічно до WPF, Avalonia надає вбудовану підтримку шаблону проектування MVVM (Model-View-ViewModel), що спрощує розробку та тестування додатків.
- Avalonia використовує мову XAML (eXtensible Application Markup Language) для декларативного описання інтерфейсу користувача, що дозволяє розділити дизайн від логіки додатку.
- Avalonia використовує асинхронний підхід до роботи з інтерфейсом користувача, що забезпечує більш плавний інтерфейс для користувача.
- Avalonia є проектом з відкритим вихідним кодом з активною спільнотою розробників, що постійно розвивається та покращується.
- Avalonia дозволяє застосовувати різні стандарти дизайну, такі як Material Design або Fluent Design, для створення сучасних інтерфейсів користувача.
- Avalonia дозволяє розробникам створювати настільні додатки, які працюють на Windows, macOS, і Linux, використовуючи один і той же код.

Compose for Desktop:

- Мова програмування: Kotlin.
- Compose for Desktop - це бібліотека для розробки графічного інтерфейсу (GUI) на мові Kotlin для настільних платформ.
- Compose for Desktop базується на Jetpack Compose, що відповідає за розробку графічного інтерфейсу для платформи Android.

- Compose for Desktop використовує декларативний підхід до створення графічного інтерфейсу, що дозволяє описувати вигляд і поведінку елементів інтерфейсу за допомогою Kotlin коду.
- Kotlin дозволяє писати компактний та ефективний код, що робить розробку додатків з використанням Compose for Desktop швидшою і зручнішою.
- Compose for Desktop дозволяє розробляти додатки для різних платформ Desktop, таких як Windows, macOS, і Linux, використовуючи один і той же код.

GTK+ (GIMP Toolkit):

- Мова програмування**:** C (Але має прив’язки до низки популярних мов програмування, як C++, Python, JavaScript й тд.)
- GTK+ підтримується на різних платформах, включаючи Linux, macOS та Windows.
- GTK+ надає широкий набір інструментів для створення різноманітних графічних інтерфейсів користувача, включаючи кнопки, поля введення, меню, вкладки, списки, таблиці та багато іншого.
- GTK+ має підтримку адаптивного дизайну, що дозволяє створювати інтерфейси, які автоматично адаптуються до різних розмірів екрану та пристроїв.
- GTK+ використовує мову розмітки інтерфейсу, відому як GTK+ Markup Language (GTK+ ML), або, для мови програмування Python, використовується мова розмітки інтерфейсу, відома як GtkBuilder.
- GTK+ дозволяє легко змінювати вигляд і поведінку елементів інтерфейсу користувача за допомогою стилів і тем.
- GTK+ є проектом з відкритим вихідним кодом з активною спільнотою розробників, що постійно розвивається та покращується.

Electron:

- Мови програмування: HTML, CSS, JavaScript (або інші веб-технології, такі як TypeScript, або те, що може компілюватися у WASM (WebAssembly) як Rust).
- Electron - це фреймворк для розробки кросплатформових десктопних додатків, які використовують веб-технології для створення інтерфейсу користувача. Він дозволяє використовувати веб-технології, такі як HTML, CSS і JavaScript, для створення десктопних додатків для Windows, macOS і Linux.
- Electron є одним з найпопулярніших фреймворків для розробки кросплатформових десктопних додатків. Він використовується тисячами компаній та розробників по всьому світу для створення різноманітних програмних продуктів.
- Electron підтримує гаряче перезавантаження додатків, що дозволяє розробникам швидко бачити зміни в коді без необхідності перезапуску додатку.
- Electron має вбудовану систему оновлень, яка дозволяє автоматично оновлювати додатки на різних платформах без необхідності ручного втручання користувача.

Ці інструменти допомагають розробникам створювати інтуїтивні та естетично привабливі GUI для своїх додатків і забезпечують їхню кросплатформову сумісність.

З перерахованих вище технологій було обрано JavaFX з таких причин:

1. JavaFX - це технологія, з якою я вже знайомий з університетських курсів. Це дозволяє мені швидко та ефективно почати роботу над проектом, використовуючи набутий досвід.
2. JavaFX не потребує компіляції під кожну платформу окремо. Достатньо однієї збірки, яка може працювати на різних операційних системах, таких як Windows, macOS та Linux, що значно спрощує процес розробки та розгортання додатків.

3. Використання FXML дозволяє розробляти інтерфейси у вигляді окремих файлів розмітки, що спрощує їх редагування та підтримку. Scene Builder - це візуальний редактор для FXML, що дозволяє розміщувати та налаштовувати елементи інтерфейсу користувача за допомогою простого перетягування миші по робочому простору.

2.3 Бібліотека для взаємодії з Docker

Для взаємодії з Docker Engine, необхідна бібліотека, яка дозволяє керувати контейнерами, образами, мережами та іншими об'єктами Docker безпосередньо з нашого додатку. У нашому проекті було обрано Docker Java API, оскільки ця бібліотека відповідає всім нашим вимогам і забезпечує наступні переваги:

1. Зручність використання з Java: Docker Java API забезпечує нативну підтримку для Java, що дозволяє легко інтегрувати Docker-контейнеризацію безпосередньо у Java-додатки, яким є наш JavaFX додаток. Це знижує складність і кількість коду, необхідного для взаємодії з Docker, оскільки всі необхідні функції вже реалізовані у вигляді методів Java.
2. Повноцінна функціональність: Docker Java API надає повний доступ до всіх можливостей Docker, включаючи створення, запуск, зупинку контейнерів, роботу з образами, мережами та об'ємами. Це дозволяє виконувати всі необхідні операції з Docker-контейнерами безпосередньо з Java-додатку.
3. Підтримка асинхронних операцій: Docker Java API підтримує асинхронні виклики (тобто такі, які не блокують потік виконання), що дозволяє підвищити продуктивність додатку та зменшити час очікування при виконанні довготривалих операцій з контейнерами.
4. Активна спільнота та документація: Docker Java API має активну спільноту користувачів та розробників, що забезпечує доступ до багатої документації, прикладів коду та підтримки. Це значно спрощує процес навчання та вирішення виникаючих проблем.

5. Забезпечення кросплатформеності: Використання Docker Java API дозволяє розробляти кросплатформенні рішення, оскільки Docker забезпечує однакове середовище виконання на різних операційних системах. Це знижує ризик виникнення проблем, пов'язаних з різницею в налаштуваннях середовища на різних платформах.

Таким чином, Docker Java API надає потужний та зручний інструмент для інтеграції можливостей Docker у Java-додатки, забезпечуючи при цьому гнучкість, масштабованість та простоту у використанні. Це робить його оптимальним вибором для взаємодії з Docker у нашому проекті.

2.4 Бібліотека для взаємодії з JSON

Для роботи з JSON у нашому проекті було обрано бібліотеку Jackson, зокрема її модуль jackson-databind. Вибір саме цієї бібліотеки був обумовлений наступними причинами:

1. Широке використання та популярність: Jackson є однією з найпопулярніших бібліотек для роботи з JSON у Java-спільноті. Вона широко використовується в багатьох проектах і має велику спільноту користувачів та розробників, що забезпечує доступ до великої кількості ресурсів, документації та підтримки.
2. Висока продуктивність: Jackson відомий своєю високою продуктивністю при обробці JSON. Він забезпечує швидку серіалізацію та десеріалізацію JSON-даних, що є критично важливим для додатків, які працюють з великими об'ємами даних або мають високу навантаженість.
3. Простота використання: Jackson надає зручний і простий у використанні API для перетворення об'єктів Java в JSON і навпаки. Завдяки анотаціям та конфігураційним опціям, можна легко налаштувати процес серіалізації та десеріалізації для конкретних потреб додатку.
4. Можливість роботи з великими об'ємами даних: Jackson надає можливість роботи з JSON у потоковому режимі (Streaming API), що дозволяє обробляти великі об'єми даних без значного споживання пам'яті.

Таким чином, використання Jackson, зокрема його модуля jackson-databind, забезпечує високу продуктивність, гнучкість та простоту у роботі з JSON, що робить його оптимальним вибором для нашого проекту.

2.5 Діаграма послідовностей.

Діаграма послідовностей (див. Рисунок 3.1) відображає взаємодію між користувачем і компонентами додатку під час запуску програми та створення нового Docker-контейнера. Вона демонструє кроки, які користувач проходить від запуску програми до створення та запуску контейнера, включаючи взаємодію з інтерфейсом та внутрішніми сервісами.

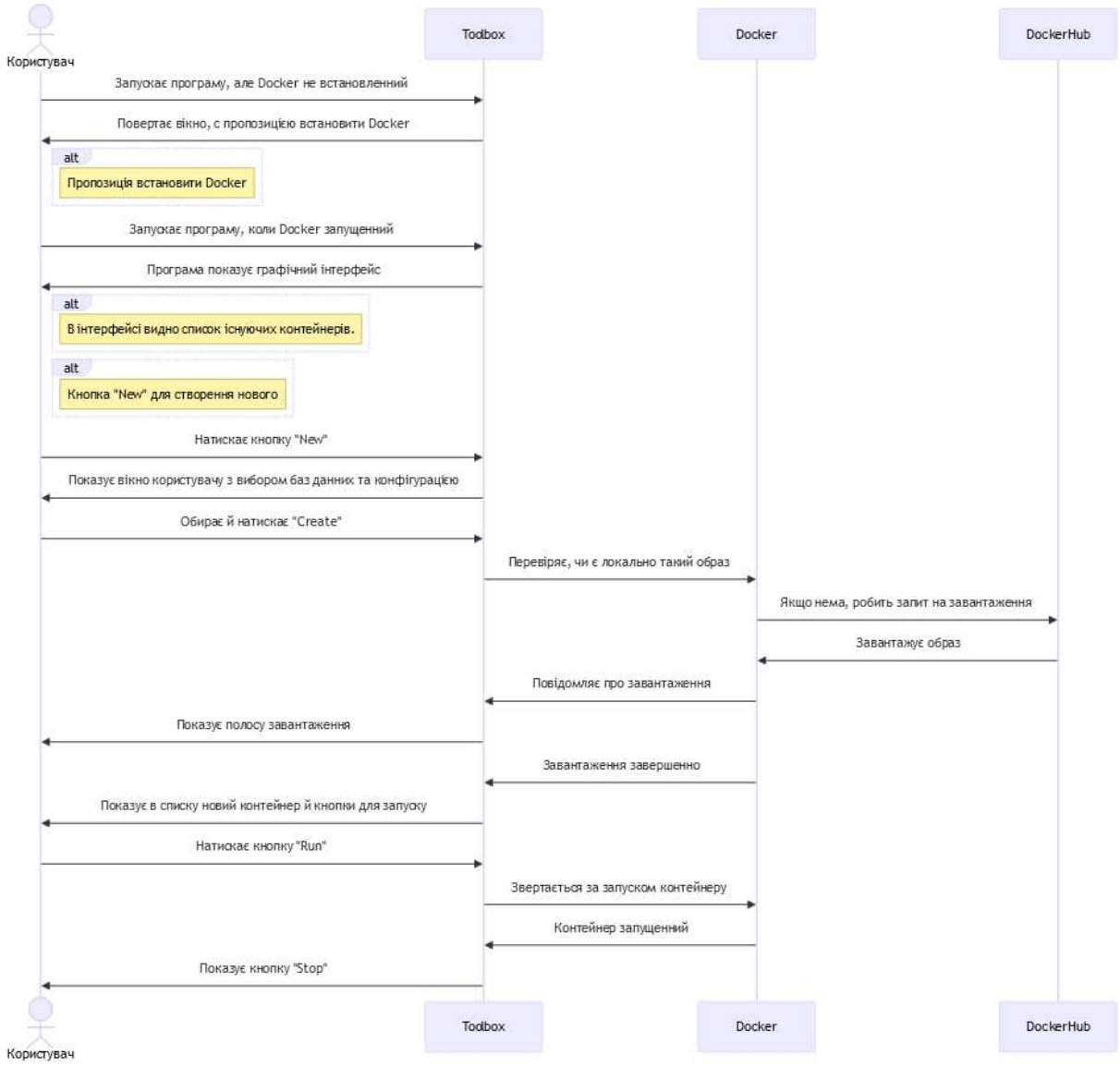

Рисунок 3.1 – Діаграма послідовностей

Діаграма послідовностей демонструє основні етапи взаємодії між користувачем, графічним інтерфейсом додатку, сервісом Docker і DockerHub. Вона дозволяє зрозуміти архітектуру та функціонування додатку, а також підкреслює важливість кожного компонента у процесі створення та управління Docker-контейнерами.

Зокрема, при запуску додатку передбачаються такі перевірки: чи встановлено Docker на комп’ютері користувача та чи є він запущений.

Також, логікою диспетчера передбачено, що, якщо користувач створює екземпляр бази даних, образу якої нема на локальному комп’ютері користувача, цей образ буде завантажено, про що користувач буде повідомлений в графічному інтерфейсі.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Вибір HTTP транспорту для Docker Java API

Для взаємодії з Docker Engine через Docker Java API необхідний надійний і функціональний HTTP транспорт, що забезпечує стабільність і підтримку всіх необхідних функцій. Після розгляду доступних опцій було обрано транспорт на базі Apache HttpClient 5. Цей вибір був обґрунтований наступними причинами:

1. Висока стабільність та підтримка: Транспорт на базі Apache HttpClient 5 має високий рівень стабільності та забезпечує довгострокову підтримку. Це важливий фактор для вибору, оскільки гарантує, що транспорт буде підтримуватися та оновлюватися в майбутньому, забезпечуючи надійну роботу додатку.
2. Підтримка всіх необхідних функцій: Транспорт на базі Apache HttpClient 5 підтримує всі необхідні функції для роботи з Docker, включаючи Unix sockets, Windows Npipe та stdin attachment. Це забезпечує повну сумісність з Docker та дозволяє використовувати всі його можливості без обмежень.
3. Довгострокові плани підтримки: Наявність довгострокових планів підтримки для цього транспорту гарантує, що він буде отримувати оновлення та виправлення помилок протягом тривалого часу. Це забезпечує надійність та безперервність роботи нашого додатку.

Виходячи з наведених причин, вибір на користь транспорту на базі Apache HttpClient 5 для Docker Java API є обґрунтованим і забезпечує всі необхідні умови для стабільної та ефективної роботи з Docker.

3.2 Використання механізму labels в Docker для збереження метаінформації

Для дотримання консистентності даних, було прийнято рішення не використовувати базу даних для збереження інформації про те, які контейнери Docker створила програма-диспетчер. Це рішення обґрунтоване тим, що

користувач може використовувати технологію Docker паралельно для власних потреб, а отже, виконувати певний менеджмент контейнерів поза програмою. Це може призвести до розбіжностей між даними, збереженими в базі даних, і реальним станом контейнерів.

Замість цього було використано механізм labels, який дозволяє додавати метаінформацію безпосередньо до Docker-контейнерів. Labels – це ключ-значення пари, які можуть бути прикріплені до Docker-контейнерів для зберігання додаткової інформації. Використання labels має кілька важливих переваг:

1. Пряма прив'язка метаданих до контейнерів: Labels дозволяють зберігати метаінформацію безпосередньо у контейнерах, що забезпечує її доступність разом із контейнером. Це означає, що інформація завжди залишається актуальною і прив'язаною до відповідного контейнера, навіть якщо контейнер був створений, змінений або видалений поза програмою.
2. Простота і гнучкість: Додавання labels до контейнерів є простим і гнучким процесом. Можна додавати будь-яку кількість ключ-значення пар, що дозволяє зберігати різноманітну інформацію про контейнер, включаючи порти, паролі, імена та інші важливі атрибути.
3. Мінімальний вплив на продуктивність: Використання labels не має значного впливу на продуктивність Docker-контейнерів. Це легкий спосіб зберігання метаінформації, який не вимагає додаткових ресурсів або складних налаштувань.
4. Уніфіковане управління: Оскільки labels зберігаються безпосередньо у контейнерах, вони забезпечують уніфіковане управління метаданими. Це дозволяє легко отримувати та оновлювати інформацію про контейнери за допомогою стандартних інструментів Docker, таких як командний рядок або API.

У нашій програмі-лаунчері labels використовуються для зберігання таких даних:

```
.withLabels(Map.of(
     "dms-toolbox", "",
     "port", database.getPort(),
     "pass", database.getPass(),
     "name", database.getName(),
     "image_name", database.getImage_name(),
     "database", database.getDatabase()
 ))
```

Ці labels дозволяють зберігати важливу інформацію про контейнери, створені програмою-диспетчером, забезпечуючи доступність цієї інформації навіть якщо контейнери керуються поза програмою. Такий підхід забезпечує високу консистентність даних і спрощує управління контейнерами.

3.3 Використання рефлексії в мові Java для читання конфігурації з JSON

З метою спростити додавання нових СКБД до диспетчера без необхідності перекомпіляції треба винести конфігурацію з програмного коду до файлу конфігурації у форматі JSON. Але, оскільки деякі змінні середовища є динамічними, тобто такими, які відомі тільки під час виконання програми (наприклад, порт, який користувач хоче використати для старту СКБД) є необхідність в конфігураційному файлі для певних змін оточення вказувати назву метода, який поверне необхідне значення.

Під час роботи програми, дані про екземпляр бази даних зберігається в об’єкті класу DatabaseInstance:

```
public class DatabaseInstance {
    private String database;
    private String image_name;
    private String name;
    private String port;
    private String pass;
    private String containerId;
    private Map<String, String> envs;
    // Методи
}
```

Приклад JSON конфігурації:

```
{
  "database" : "MySQL",
  "image_name" : "mysql:latest",
  "name" : null,
  "port" : "3306",
  "pass" : "admin",
```

```
  "containerId" : null,
  "envs" : {
    "MYSQL_ALLOW_CLEAR_PASSWORD=" : "yes",
    "MYSQL_SSL_MODE=" : "DISABLED",
    "MYSQL_ROOT_PASSWORD=" : "getPass"
 }
```

Змінні оточення «MYSQL_ALLOW_CLEAR_PASSWOD», «MYSQL_SSL_MODE» мають мати статичні значення, то змінна «MYSQL_ROOT_PASSWORD» має отримати значення динамічно, таке, як вкаже у вікні користувач. Для цього у полі значення, вказано назва методу класу DatabaseInstance – «getPass».

Допоміжний клас, EnvHelper, використовується при зчитуванні конфігурації:

```
public class EnvHelper {
    public static String[] getEnvVars(DatabaseInstance database) {
        List<String> envVars = new ArrayList<>();
        Class<?> clazz = database.getClass();
        for (Map.Entry<String, String> entry :
database.getEnvs().entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();
            try {
                Method method = clazz.getDeclaredMethod(value);
                Object methodResult = method.invoke(database);
                envVars.add(key + methodResult);
            } catch (NoSuchMethodException e) {
                envVars.add(key + value);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return envVars.toArray(new String[0]);
    }
}
```

В методі getEnvVars перевіряється чи є значення назвою метода, і якщо так – то підставляється значення, що повертається після виклику метода, а якщо ні, то статичне значення.

3.2 Додавання нової бази даних в диспетчер

Конфігурація доступних баз даних в диспетчері зберігається у файлі config.json (див. А.2 Конфігурація JSON) в кореневій директорії та представляє собою масив JSON об’єктів. Для прикладу, щоб додати підтримку бази даних DB2 треба додати в масив об’єкт такого вигляду:

```
{
  "database" : "DB2",
  "image_name" : "ibmcom/db2:latest",
  "name" : null,
  "port" : "50000",
  "pass" : "admin",
  "containerId" : null,
  "envs" : {
    "LICENSE=" : "accept",
    "DB2INST1_PASSWORD=" : "getPass",
    "DBNAME=" : "toolbox"
}
```

Структура об’єкту має такі поля, які необхідно заповнити:

- «database» - Назва бази даних, що буде відображатися в графічному інтерфейсі програми.
- «image_name» - Назва Docker-образу, який необхідно завантажити для підтримки та на основі якого будуть створюватися контейнери.
- «port» - Порт, який графічний інтерфейс буде пропонувати для запуску бази даних за замовчуванням.
- «pass» - Пароль, який графічний інтерфейс буде пропонувати за замовчуванням.
- «envs» - Об’єкт, в якому зберігаються пари рядок-рядок, в першій - назва змінної оточення, в другій – значення. В другому рядку, окрім статичного значення, можна вказати метод з класу DatabaseInstance.java, щоб отримати якесь значення динамічно, наприклад, пароль чи порт.

3.3 Опис програмної реалізації

Для функціювання інформаційної системи, необхідно використовувати певні залежності, такі як бібліотеки Docker Java API. Було використано менеджер залежностей й систему збірки Maven [28], який для конфігурації використовує файл pom.xml (див. А.1 Конфігурація Maven), в якому вказані необхідні бібліотеки для підтримки в застосунку JavaFX, Docker Java API, Jackson.

Для того щоб створити готовий JAR-архів, який можна розповсюджувати на комп'ютери користувачів, використовується Maven-плагін maven-shade-plugin. Цей плагін дозволяє об'єднати всі залежності вашого проекту в один JAR-файл, включаючи всі необхідні класи і ресурси. Наведений нижче фрагмент конфігурації показує, як налаштувати maven-shade-plugin для створення такого JAR-файлу.

Для того, щоб зібрати проект, достатньо виконати команду: mvn package. В результаті буде отриманий jar-файл в директорії target.

Структура проекту виглядає наступним чином (див. Рисунок 3.2)

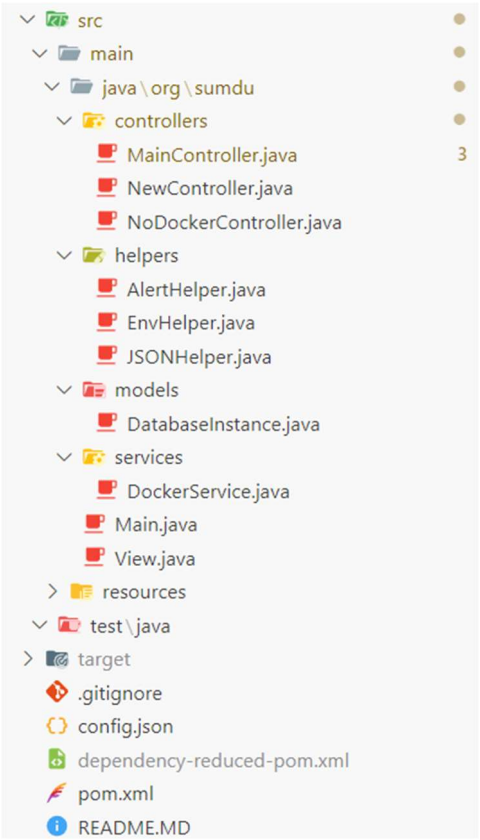

Рисунок 3.2 – Структура проекту

Головним пакетом є org.sumdu, в якому є клас, з якого починається виконання програми – Main.java. Виглядає він наступним чином:

```
package org.sumdu;

public class Main {
    public static void main(String[] args) {
        View.main(args);
    }
}
```

Його завдання полягає в тому, щоб запустити клас, який відповідає за запуск графічного додатку JavaFX – View.java. Розглянемо цей клас:

```
package org.sumdu;

public class View extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        if (isDockerInstalled()) {
            if (isDockerRunning()) {
                // Показати головне вікно
            } else {
                // Показати вікно з пропозицією запустити Docker
            }
        } else {
            // Показати вікно з пропозицією встановити Docker
    }
    // Метод, що перевіряє, чи встановлено Docker
    public static boolean isDockerInstalled() { /* код */ }
    // Метод, що перевіряє, чи запущено Docker
    public static boolean isDockerRunning() { /* код */ }
}
```

Цей клас, при запуску має певну логіку, а саме, перевіряє чи встановлений на комп’ютері користувача Docker, й чи він запущенний, і відповідно має методи які вміють це перевіряти. Також, в залежності від результату в методі start, приймається рішення, яке вікно показати користувачу: головне, чи з попередженням.

Розмітки цих вікон та іконка програми, зберігається в директорії resources (див. Рисунок 3.3).

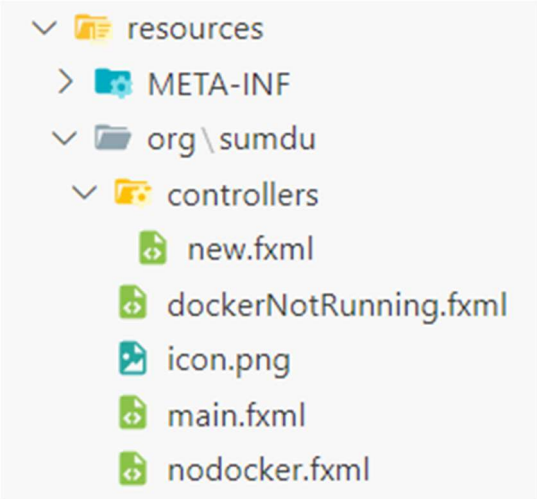

Рисунок 3.3 – Ресурси

Тепер розглянемо структуру пакетів в org.sumdu.

В пакеті contollers зберігаються класи, які є контролерами, тобто містять логіку взаємодії користувачем з графічним інтерфейсом, як, наприклад, оброблення події натискання на кнопку створення нового екземпляра бази даних.

В пакеті models зберігаються класи-моделі, тобто такі, які зберігають в собі інформацію про якусь сутність. Клас DatabaseInstance.java інкапсулює дані про екземпляр бази даних.

В пакеті helpers зберігаються допоміжні класи, які дозволяють зручно виконувати певні дії, як клас JSONHelper.java, що дозволяє зчитувати конфігурацію з JSON та отримувати список об’єктів DatabaseInstance.java.

В пакеті services зберігаються класи-сервіси, тобто такі, що інкапсулють логіку роботи з певним компонентом, як в нашому випадку DockerService.java з Docker.

Розглянемо деякі методи з класу DockerService.java

```
public List<DatabaseInstance> readListsOfDatabases() throws
DockerException {
```

```
        List<DatabaseInstance> databaseInstances = new ArrayList<>();

        dockerClient.listContainersCmd()
                .withShowAll(true)
                .withLabelFilter(Collections.singletonList("dms-
toolbox"))
                .exec()
                .forEach(container -> {
                    Map<String, String> labels =
container.getLabels();
                    DatabaseInstance databaseInstance = new
DatabaseInstance();
                    databaseInstance.setName(labels.get("name"));
                    databaseInstance.setPort(labels.get("port"));
                    databaseInstance.setPass(labels.get("pass"));
                    databaseInstance.setImage_name(labels.get("image_n
ame"));
                    databaseInstance.setDatabase(labels.get("database"
));
                    databaseInstance.setContainerId(container.getId())
;
                    databaseInstances.add(databaseInstance);
                });

        return databaseInstances;
}
```

Цей метод, робить запит до Docker, щоб отримати список контейнерів, з поміткою “dms-toolbox”, тобто таких, що були створені диспетчером. Це потрібно при перезапуску диспетчера, щоб отримати дані про існуючі контейнери.

```
public boolean isImageDownloaded(String imageName) throws
DockerException {
        return dockerClient.listImagesCmd()
                .exec()
                .stream()
                .flatMap(image -> Arrays.stream(image.getRepoTags()))
                .anyMatch(repoTag -> repoTag.equals(imageName));
}
```

Цей метод, робить запит до Docker, з метою перевірити, чи вже завантажений образ бази даних з ім’ям зі змінної imageName.

3.4 Аналіз результатів

Якщо у користувача, не встановлено Docker, він побачить вікно, яке повідомить про необхідність цього, та запропонує посилання на сайт (див. Рисунок 3.4):

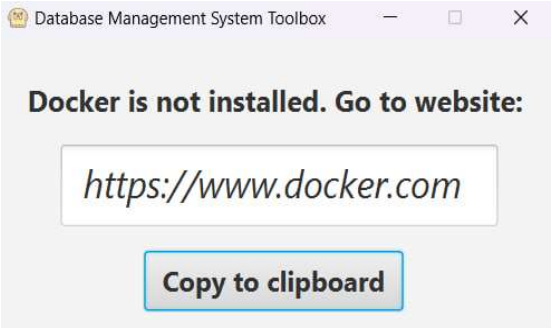

Рисунок 3.4 – Вікно, яке пропонує завантажити Docker

Головне вікно (див. Рисунок 3.5) програми складається зі списку створених екземплярів баз даних, у кожному елементі якого є кнопки для запуску або зупинки, видалення та копіювання строки підключення до буферу обміну.

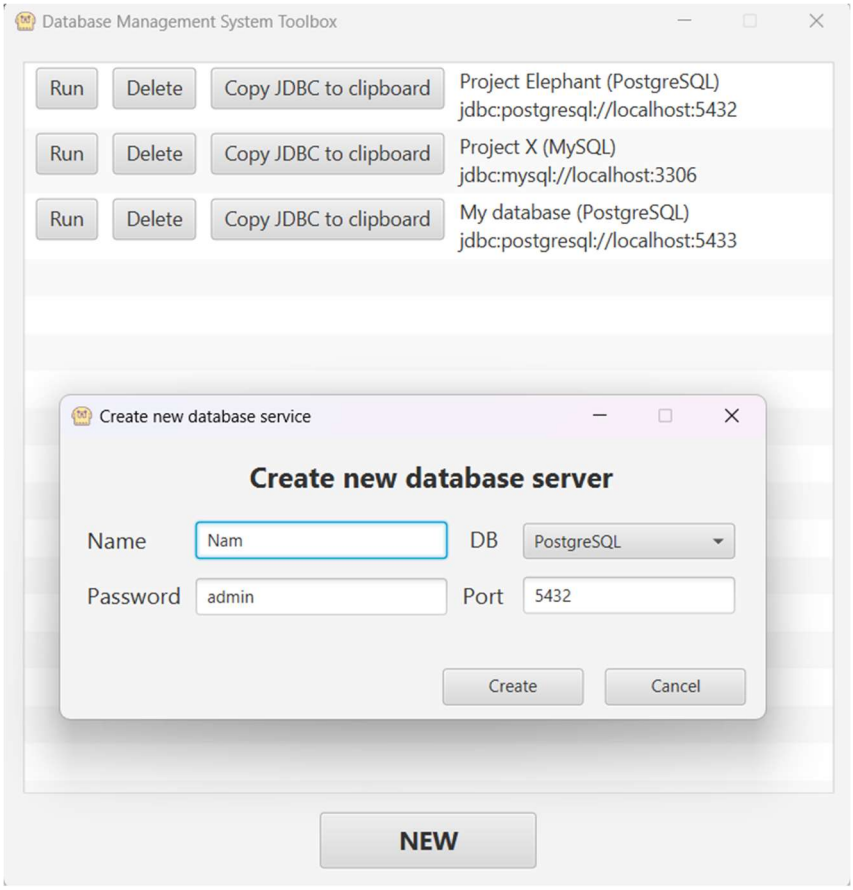

Рисунок 3.5 – Головне вікно програми

Додатковими елементами головного вікна є велика кнопка для створення нового екземпляру, при натисканні на яку, відкривається додаткове

вікно, що пропонує просте налаштування, а саме: обрати вид бази даних, вказати ім’я, пароль та порт.

Присутня обробка помилок, яка не дасть користувачу залишити ім’я або пароль пустим, а також в порті вказати текст замість числа (див. Рисунок 3.6 та Рисунок 3.7)

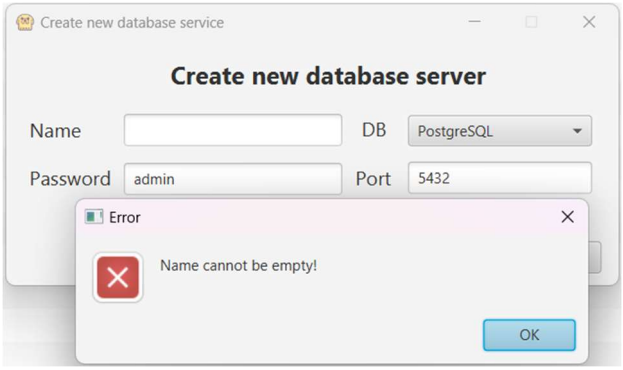

Рисунок 3.6 – Обробка пустого ім’я

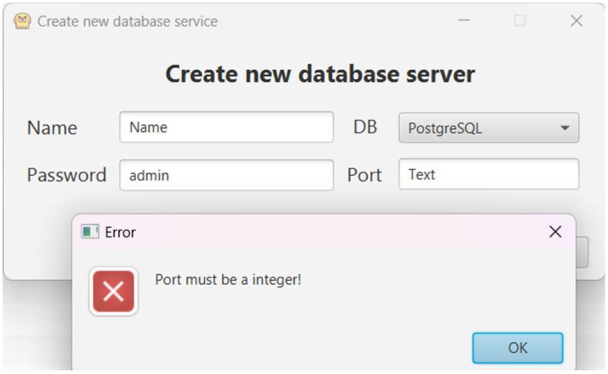

Рисунок 3.7 – Обробка текстового порту

Якщо на комп’ютері користувача, ще не завантажений образ бази даних, екземпляр якої він хоче створити, то програма завантажить його, про що користувач буде повідомлений графічним інтерфейсом полосою завантаження (див. Рисунок 3.8):

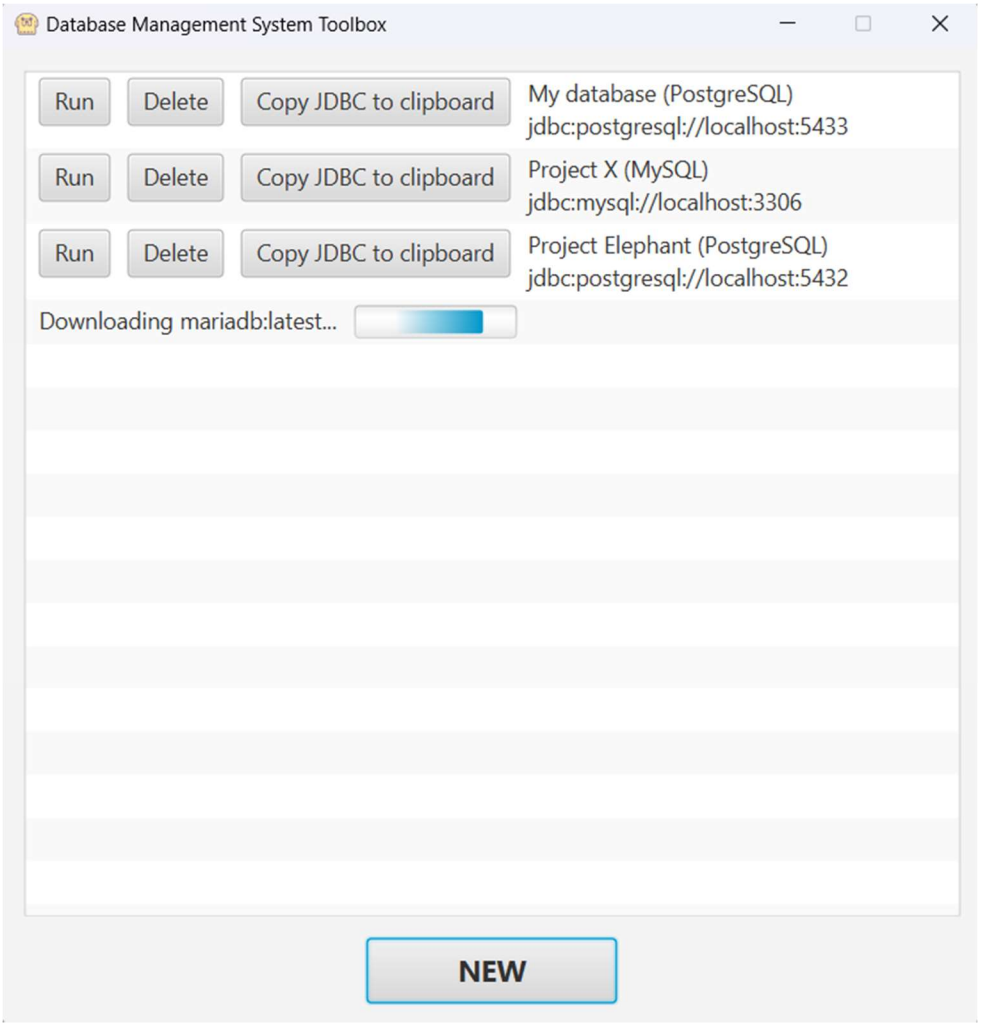

Рисунок 3.8 – Завантаження образу

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи створено інформаційне та програмне забезпечення диспетчера системам керування базами даних.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

1. Розглянуто різні інструменти для створення графічного інтерфейсу, проаналізовано їх переваги та недоліки й зроблений вибір на користь JavaFX.
2. Створено проект Maven, який включає в себе всі необхідні залежності для диспетчера СКБД на основі JavaFX.
3. Розроблено графічний інтерфейс, створено необхідні класи, що включають логіку взаємодію з користувачем, інкапсулюють дані та організовують роботу з системою Docker.
4. Проаналізовані шляхи забезпечення консистентності інформації щодо об’єктів управління. Прийнято рішення застосовувати мета-інформацію середовища Docker (а саме labels) для перевірки наявності та стану екземпляра бази даних.
5. Додано підтримку конфігурації диспетчера у форматі JSON, що дає змогу додавати підтримку нових СКБД в диспетчер без зміни програмного коду та компіляції.
6. Використано механізм рефлексії мови програмування Java для зчитування динамічних даних з конфігурації у форматі JSON.
7. Додано обробку помилок, що не дозволить користувачу створювати екземпляри СКБД з неправильною конфігурацією.

Застосування розробленої інформаційної системи зменшує час необхідний для розгортання СКБД, конфігурації, покращує досвід розробника, позбавляє заплутаних сценаріїв використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. z/OS Basic Skills. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system (дата звернення: 17.05.2024).
2. Docker: Accelerated Container Application Development. Docker. URL: https://www.docker.com/ (дата звернення: 17.05.2024).
3. JetBrains: Essential tools for software developers and teams. JetBrains. URL: https://www.jetbrains.com/ (дата звернення: 17.05.2024).
4. What is an IDE? - Integrated Development Environment Explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/ide/ (дата звернення: 17.05.2024).
5. macOS - What is macOS. Apple (Belarus). URL: https://www.apple.com/by/macos/what-is/ (дата звернення: 17.05.2024).
6. The Editors of Encyclopaedia Britannica. Microsoft Windows | History, Versions, & Facts. Encyclopedia Britannica. URL: https://www.britannica.com/technology/Microsoft-Windows (дата звернення: 17.05.2024).
7. Here’s Why JSON Rules the Web. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/database/what-is-json/ (дата звернення: 17.05.2024).
8. Poriyaalar Pvt Ltd. Desktop applications- What are they? Advantages, uses and technologies involved. LinkedIn: Log In or Sign Up. URL: https://www.linkedin.com/pulse/desktop-applications-what-advantages-uses-technologies-involved-/ (дата звернення: 17.05.2024).
9. Gui: What is a gui | How does GUI work | Lenovo US. Offizielle Lenovo DE Website | Notebooks, Tablets, PCs, Rechenzentren, Phones | Lenovo Deutschland. URL: https://www.lenovo.com/us/en/glossary/what-is-a-

gui/?orgRef=https%3A%2F%2Fwww.google.com%2F (дата звернення: 17.05.2024).

10. Qt | Development Framework for Cross-platform Applications. Qt | Tools for Each Stage of Software Development Lifecycle. URL: https://www.qt.io/product/framework (дата звернення: 17.05.2024).
11. Volle A. C++ | Definition, History, & Facts. Encyclopedia Britannica. URL: https://www.britannica.com/technology/C-computer-language (дата звернення: 17.05.2024).
12. Embedded Software Development Tools & Cross Platform IDE | Qt Creator. Qt | Tools for Each Stage of Software Development Lifecycle. URL: https://www.qt.io/product/development-tools (дата звернення: 17.05.2024).
13. The QML Reference | Qt QML 6.7.1. Qt Documentation | Home. URL: https://doc.qt.io/qt-6/qmlreference.html (дата звернення: 17.05.2024).
14. JavaFX. JavaFX. URL: https://openjfx.io/ (дата звернення: 17.05.2024).
15. Java Software. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/java/ (дата звернення: 17.05.2024).
16. Getting Started with JavaFX: Using FXML to Create a User Interface | JavaFX 2 Tutorials and Documentation. Moved. URL: https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm (дата звернення: 17.05.2024).
17. JavaFX Scene Builder Information. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html (дата звернення: 17.05.2024).
18. CSS: Cascading Style Sheets | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 17.05.2024).

19. wxWidgets: Cross-Platform GUI Library. wxWidgets: Cross-Platform GUI Library. URL: https://www.wxwidgets.org/ (дата звернення: 17.05.2024).
20. Welcome to wxPython!. wxPython. URL: https://wxpython.org/index.html (дата звернення: 17.05.2024).
21. javax.swing (Java Platform SE 7). Moved. URL: https://docs.oracle.com/javase/7/docs/api///javax/swing/package-summary.html (дата звернення: 17.05.2024).
22. Java Developer's Guide. Oracle Help Center. URL: https://docs.oracle.com/en/database/oracle/oracle-database/18/jjdev/Oracle-JVM-overview.html (дата звернення: 17.05.2024).
23. Tkinter Python interface to Tcl/Tk. Python documentation. URL: https://docs.python.org/3/library/tkinter.html (дата звернення: 17.05.2024).
24. What is Python? Executive Summary. Python.org. URL: https://www.python.org/doc/essays/blurb/ (дата звернення: 17.05.2024).
25. FAQ. Docs | Flutter. URL: https://docs.flutter.dev/resources/faq#what-is-flutter (дата звернення: 17.05.2024).
26. Skia. Skia. URL: https://skia.org/ (дата звернення: 17.05.2024).
27. Model-View-ViewModel - .NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (дата звернення: 17.05.2024).
28. Maven Introduction. Maven Welcome to Apache Maven. URL: https://maven.apache.org/what-is-maven.html (дата звернення: 17.05.2024).

ДОДАТОК А

А.1 Конфігурація Maven

```
// pom.xml – Конфігурація проекту для менеджера залежностей та системи
// збірки Maven
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.sumdu</groupId>
    <artifactId>DatabaseManagmentSystemToolbox</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>21</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>21</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.17.1</version>
        </dependency>
        <dependency>
            <groupId>com.github.docker-java</groupId>
            <artifactId>docker-java-core</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.httpcomponents.client5</groupId>
            <artifactId>httpclient5</artifactId>
            <version>5.3.1</version>
```

```
        </dependency>
        <dependency>
            <groupId>com.github.docker-java</groupId>
            <artifactId>docker-java-transport-httpclient5</artifactId>
            <version>3.3.6</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.5.6</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>2.0.13</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.0.0</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <filters>
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-
INF/*.SF</exclude>
                                        <exclude>META-
INF/*.DSA</exclude>
                                        <exclude>META-
INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer
                                        implementation="org.apache.mav
en.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>org.sumdu.Main</mainCla
ss>
                                </transformer>
```

```
                        </transformers>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

</project>
```

A.2 Конфігурація JSON

```
// config.json – Конфігурація баз даних, які буде підтримувати
диспетчер.
[ {
  "database" : "PostgreSQL",
  "image_name" : "postgres:latest",
  "name" : null,
  "port" : "5432",
  "pass" : "admin",
  "containerId" : null,
  "envs" : {
    "PGPORT=" : "getPort",
    "POSTGRES_PASSWORD=" : "getPass"
  }
}, {
  "database" : "MySQL",
  "image_name" : "mysql:latest",
  "name" : null,
  "port" : "3306",
  "pass" : "admin",
  "containerId" : null,
  "envs" : {
    "MYSQL_ALLOW_CLEAR_PASSWORD=" : "yes",
    "MYSQL_SSL_MODE=" : "DISABLED",
    "MYSQL_ROOT_PASSWORD=" : "getPass"
  }
}, {
  "database" : "MariaDB",
  "image_name" : "mariadb:latest",
  "name" : null,
  "port" : "3307",
  "pass" : "admin",
  "containerId" : null,
  "envs" : {
    "MARIADB_SSL_MODE=" : "DISABLED",
    "MARIADB_ROOT_PASSWORD=" : "getPass",
    "MARIADB_ALLOW_CLEAR_PASSWORD=" : "yes"
  }
}, {
```

```
  "database" : "DB2",
  "image_name" : "ibmcom/db2:latest",
  "name" : null,
  "port" : "50000",
  "pass" : "admin",
  "containerId" : null,
  "envs" : {
    "LICENSE=" : "accept",
    "DB2INST1_PASSWORD=" : "getPass",
    "DBNAME=" : "toolbox"
  }
} ]
```

```
// Main.java – Головний клас програми
package org.sumdu;

public class Main {
    public static void main(String[] args) {
        View.main(args);
    }
}
```

```
// View.java – Клас додатку JavaFX
package org.sumdu;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import org.sumdu.controllers.MainController;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class View extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        if (isDockerInstalled()) {
            if (isDockerRunning()) {
                FXMLLoader loader = new
FXMLLoader(getClass().getResource("main.fxml"));
                Parent root = loader.load();
                MainController mainController =
loader.getController();
                primaryStage.getIcons().add(new
Image(String.valueOf(getClass().getResource("icon.png"))));
```

```
                primaryStage.setTitle("Database Management System
Toolbox");
                primaryStage.setScene(new Scene(root, 600, 600));
                primaryStage.setResizable(false);
                primaryStage.setOnCloseRequest(event ->
mainController.stopAllContainers());
                primaryStage.show();
            } else {
                Parent root =
FXMLLoader.load(getClass().getResource("dockerNotRunning.fxml"));
                primaryStage.getIcons().add(new
Image(String.valueOf(getClass().getResource("icon.png"))));
                primaryStage.setTitle("Database Management System
Toolbox");
                primaryStage.setScene(new Scene(root, 400, 200));
                primaryStage.setResizable(false);
                primaryStage.show();
            }
        } else {
            Parent root =
FXMLLoader.load(getClass().getResource("nodocker.fxml"));
            primaryStage.getIcons().add(new
Image(String.valueOf(getClass().getResource("icon.png"))));
            primaryStage.setTitle("Database Management System
Toolbox");
            primaryStage.setScene(new Scene(root, 400, 200));
            primaryStage.setResizable(false);
            primaryStage.show();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }

    public static boolean isDockerInstalled() {
        try {
            Process process = new ProcessBuilder("docker", "--
version").start();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains("Docker version")) {
                    return true;
                }
            }
            return false;
        } catch (IOException e) {
            e.printStackTrace();
            return false;
```

```
        }
    }

    public static boolean isDockerRunning() {
        try {
            Process process = new ProcessBuilder("docker",
"ps").start();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains("CONTAINER ID")) {
                    return true;
                }
            }
            return false;
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
}

// MainController.java – Контроллер головного вікна
package org.sumdu.controllers;

import com.github.dockerjava.api.exception.DockerException;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.input.Clipboard;
import javafx.scene.input.ClipboardContent;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.geometry.Pos;

import org.sumdu.helpers.AlertHelper;
import org.sumdu.models.DatabaseInstance;
import org.sumdu.services.DockerService;

import java.io.IOException;
import java.util.List;

public class MainController {
    public Button NewButton;
    public ListView DatabasesView;
```

```
    private DockerService dockerService = new DockerService();
    private List<DatabaseInstance> databaseInstances;

    public void initialize() {
        databaseInstances = dockerService.readListsOfDatabases();

        for (var databaseInstance : databaseInstances) {
            addButtonAndLabel(databaseInstance);
        }
    }

    public void newButtonClicked(MouseEvent mouseEvent) {
        try {
            FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("new.fxml"));
            Parent root = fxmlLoader.load();
            NewController newController = fxmlLoader.getController();
            newController.setMainController(this);
            Stage stage = new Stage();
            stage.setScene(new Scene(root));
            stage.getIcons().add(new
Image(String.valueOf(getClass().getResource("../icon.png"))));
            stage.setTitle("Create new database service");
            stage.setResizable(false);
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void addNewDatabaseInstance(DatabaseInstance database)
throws DockerException {
        databaseInstances.add(database);

        if (dockerService.isImageDownloaded(database.getImage_name()))
{
            addButtonAndLabel(database);
        } else {
            ProgressBar progressBar = new ProgressBar();
            progressBar.setProgress(ProgressBar.INDETERMINATE_PROGRESS
);

            Label label = new Label("Downloading " +
database.getImage_name() + "...");
            HBox.setMargin(label, new Insets(0, 10, 0, 0));
            HBox hbox = new HBox(label, progressBar);

            VBox vbox = new VBox(hbox);

            DatabasesView.getItems().add(vbox);
```

```
            dockerService.pullImage(database, vbox, this);
        }
    }

    public void addButtonAndLabel(DatabaseInstance database) {
        Button deleteButton = new Button("Delete");
        deleteButton.setOnAction(event -> {
            try {
                dockerService.removeContainer(database.getContainerId(
));
                HBox vboxToRemove = (HBox) ((Button)
event.getSource()).getParent().getParent();
                DatabasesView.getItems().remove(vboxToRemove);
            } catch (InterruptedException e) {
                AlertHelper.showAlert(e.getMessage());
            }
        });

        Button runAndStopButton = new Button("Run");
        runAndStopButton.setOnAction(event -> {
            var button = (Button) event.getSource();
            if (button.getText().equals("Run")) {
                try {
                    dockerService.runDockerContainer(database);
                    button.setText("Stop");
                } catch (DockerException | InterruptedException e) {
                    AlertHelper.showAlert(e.getMessage());
                }
            } else {
                dockerService.stopContainer(database.getContainerId())
;
                button.setText("Run");
            }
        });

        Label nameLabel = new Label(String.format("%s (%s)",
database.getName(), database.getDatabase()));
        String jdbcStr = database.getJDBCStr();
        Label jdbcLabel = new Label(jdbcStr);

        Button copyButton = new Button("Copy JDBC to clipboard");
        copyButton.setOnAction(event -> {
            Clipboard clipboard = Clipboard.getSystemClipboard();
            ClipboardContent content = new ClipboardContent();
            content.putString(jdbcStr);
            clipboard.setContent(content);
        });

        VBox labelBox = new VBox(nameLabel, jdbcLabel);
```

```
        HBox hbox = new HBox(runAndStopButton, deleteButton,
copyButton);
        labelBox.setAlignment(Pos.TOP_LEFT);

        HBox item = new HBox(hbox, labelBox);
        item.setSpacing(10);
        hbox.setSpacing(10);

        DatabasesView.getItems().add(item);
    }

    public void stopAllContainers() {
        for (DatabaseInstance database : databaseInstances) {
            if
(dockerService.isContainerRunning(database.getContainerId())) {
                dockerService.stopContainer(database.getContainerId())
;
            }
        }
    }
}

// NewController.java – Контролер вікна створення нового екземпляру
базу
package org.sumdu.controllers;

import com.github.dockerjava.api.exception.DockerException;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.scene.control.*;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
import org.sumdu.helpers.AlertHelper;
import org.sumdu.helpers.JSONHelper;
import org.sumdu.models.DatabaseInstance;

import java.io.IOException;
import java.util.List;
import java.util.stream.Collectors;

public class NewController {
    public Button CancelButton;
    public Button Create;
    public Label NameLabel;
    public TextField NameText;
    public TextField PortText;
    public Label PortLabel;
    public TextField PasswordText;
    public Label PasswordLabel;
    public ComboBox<String> DatabaseComboBox;
```

```
    public Label DatabaseLabel;
    public Label TitleLabel;

    private MainController mainController;
    private List<DatabaseInstance> databases;

    public void initialize() throws IOException {
        databases =
JSONHelper.readDatabaseInstancesFromFile("./config.json");

        ObservableList<String> databaseList =
FXCollections.observableArrayList(
                databases.stream()
                        .map(DatabaseInstance::getDatabase)
                        .collect(Collectors.toList())
        );
        var postgresInstance = databases.stream()
                .filter(instance ->
"PostgreSQL".equals(instance.getDatabase()))
                .findFirst()
                .orElse(null);

        DatabaseComboBox.setItems(databaseList);
        DatabaseComboBox.getSelectionModel().select(postgresInstance.g
etDatabase());
        PasswordText.setText(postgresInstance.getPass());
        PortText.setText(postgresInstance.getPort());
    }

    public void cancelButtonClicked(MouseEvent mouseEvent) {
        Stage stage = (Stage) CancelButton.getScene().getWindow();
        stage.close();
    }

    public void createButtonClicked(MouseEvent mouseEvent) {
        if (NameText.getText().isEmpty()) {
            AlertHelper.showAlert("Name cannot be empty!");
        } else if (PasswordText.getText().isEmpty()) {
            AlertHelper.showAlert("Password cannot be empty!");
        } else if (!PortText.getText().matches("\\d+")) {
            AlertHelper.showAlert("Port must be a integer!");
        } else {
            var database_type =
DatabaseComboBox.getSelectionModel().getSelectedItem();
            var dbInstance = databases.stream()
                    .filter(instance ->
database_type.equals(instance.getDatabase()))
                    .findFirst()
                    .orElse(null);
            var database =  new DatabaseInstance(
                    dbInstance.getDatabase(),
```

```
                    dbInstance.getImage_name(),
                    NameText.getText(),
                    PortText.getText(),
                    PasswordText.getText(),
                    dbInstance.getEnvs()
            );

            try {
                mainController.addNewDatabaseInstance(database);
            } catch (DockerException e) {
                AlertHelper.showAlert(e.getMessage());
            }

            Stage stage = (Stage) CancelButton.getScene().getWindow();
            stage.close();
        }
    }

    public void onDatabaseComboBoxAction(ActionEvent actionEvent) {
        String selectedDatabase =
DatabaseComboBox.getSelectionModel().getSelectedItem();
        var dbInstance = databases.stream()
                .filter(instance ->
selectedDatabase.equals(instance.getDatabase()))
                .findFirst()
                .orElse(null);
        PasswordText.setText(dbInstance.getPass());
        PortText.setText(dbInstance.getPort());
    }

    public void setMainController(MainController mainController) {
        this.mainController = mainController;
    }
}

// NoDockerController.java – Контролер вікна, що повідомляє про
відсутність Docker
package org.sumdu.controllers;

import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.Clipboard;
import javafx.scene.input.ClipboardContent;
import javafx.scene.input.MouseEvent;

public class NoDockerController {
    public TextField DockerWebsite;
    public Button CopyButton;

    public void onCopyClick(MouseEvent mouseEvent) {
        String text = DockerWebsite.getText();
```

```
        Clipboard clipboard = Clipboard.getSystemClipboard();
        ClipboardContent content = new ClipboardContent();
        content.putString(text);
        clipboard.setContent(content);
    }
}

// DockerService.java
package org.sumdu.services;

import com.github.dockerjava.api.DockerClient;
import com.github.dockerjava.api.async.ResultCallback;
import com.github.dockerjava.api.command.CreateContainerResponse;
import com.github.dockerjava.api.command.PullImageCmd;
import com.github.dockerjava.api.exception.DockerException;
import com.github.dockerjava.api.model.*;
import com.github.dockerjava.core.DefaultDockerClientConfig;
import com.github.dockerjava.core.DockerClientImpl;
import com.github.dockerjava.httpclient5.ApacheDockerHttpClient;
import com.github.dockerjava.transport.DockerHttpClient;
import javafx.application.Platform;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import org.sumdu.controllers.MainController;
import org.sumdu.helpers.EnvHelper;
import org.sumdu.models.DatabaseInstance;

import java.io.Closeable;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.util.*;

public class DockerService {
    private final DockerClient dockerClient;

    public DockerService() {
        DefaultDockerClientConfig config =
DefaultDockerClientConfig.createDefaultConfigBuilder()
                .withDockerHost("npipe:////./pipe/docker_engine")
                .build();

        DockerHttpClient httpClient = new
ApacheDockerHttpClient.Builder()
                .dockerHost(config.getDockerHost())
                .sslConfig(config.getSSLConfig())
                .maxConnections(100)
                .connectionTimeout(Duration.ofSeconds(30))
                .responseTimeout(Duration.ofSeconds(45))
                .build();
```

```
        dockerClient = DockerClientImpl.getInstance(config,
httpClient);
    }

    public boolean isImageDownloaded(String imageName) throws
DockerException {
        return dockerClient.listImagesCmd()
                .exec()
                .stream()
                .flatMap(image -> Arrays.stream(image.getRepoTags()))
                .anyMatch(repoTag -> repoTag.equals(imageName));
    }

    public boolean isContainerExists(String containerId) throws
DockerException {
        return dockerClient.listContainersCmd()
                .withShowAll(true)
                .withIdFilter(Collections.singletonList(containerId))
                .exec()
                .stream()
                .anyMatch(container ->
container.getId().equals(containerId));
    }

    public void runDockerContainer(DatabaseInstance database) throws
DockerException, InterruptedException {
        if (isContainerExists(database.getContainerId())) {
            startContainer(database.getContainerId());
        } else {
            createAndStartContainer(database);
        }
    }

    public void startContainer(String containerName) throws
DockerException {
        dockerClient.startContainerCmd(containerName).exec();
    }

    public void stopContainer(String containerName) {
        dockerClient.stopContainerCmd(containerName).exec();
    }

    public void createAndStartContainer(DatabaseInstance database)
throws DockerException {
        int hostPort = Integer.parseInt(database.getPort());

        ExposedPort exposedPort = ExposedPort.tcp(hostPort);
        Ports portBindings = new Ports();
        portBindings.bind(exposedPort,
Ports.Binding.bindPort(hostPort));
```

```
        String hostDirectory = String.format("C:\\%s\\data",
database.getDatabase().toLowerCase());
        createDirectoryIfNotExists(hostDirectory);

        Volume volume = new Volume("/database");
        Bind bind = new Bind(hostDirectory, volume);

        HostConfig hostConfig = HostConfig.newHostConfig()
                .withPortBindings(portBindings)
                .withBinds(bind);

        var envVars = EnvHelper.getEnvVars(database);
        CreateContainerResponse container =
dockerClient.createContainerCmd(database.getImage_name())
                .withHostConfig(hostConfig)
                .withExposedPorts(exposedPort)
                .withEnv(envVars)
                .withLabels(Map.of(
                        "dms-toolbox", "",
                        "port", database.getPort(),
                        "pass", database.getPass(),
                        "name", database.getName(),
                        "image_name", database.getImage_name(),
                        "database", database.getDatabase()
                ))
                .exec();

        database.setContainerId(container.getId());

        dockerClient.startContainerCmd(container.getId()).exec();
    }

    private void createDirectoryIfNotExists(String directoryPath) {
        Path path = Paths.get(directoryPath);
        try {
            if (!Files.exists(path)) {
                Files.createDirectories(path);
            }
        } catch (Exception e) {
            throw new RuntimeException("Failed to create directory: "
+ directoryPath, e);
        }
    }

    public void pullImage(DatabaseInstance database, VBox vbox,
MainController mainController) {
        PullImageCmd pullImageCmd =
dockerClient.pullImageCmd(database.getImage_name());
        pullImageCmd.exec(new ResultCallback<PullResponseItem>() {
            @Override
```

```
            public void onStart(Closeable closeable) {
            }

            @Override
            public void onNext(PullResponseItem object) {
            }

            @Override
            public void onError(Throwable throwable) {
                Platform.runLater(() -> {
                    Label errorLabel = new Label("Error downloading "
+ database.getImage_name());
                    vbox.getChildren().set(0, errorLabel);
                });
            }

            @Override
            public void onComplete() {
                Platform.runLater(() -> {
                    mainController.DatabasesView.getItems().remove(vbo
x);
                    mainController.addButtonAndLabel(database);
                });
            }

            @Override
            public void close() {
            }
        });
    }

    public List<DatabaseInstance> readListsOfDatabases() throws
DockerException {
        List<DatabaseInstance> databaseInstances = new ArrayList<>();

        dockerClient.listContainersCmd()
                .withShowAll(true)
                .withLabelFilter(Collections.singletonList("dms-
toolbox"))
                .exec()
                .forEach(container -> {
                    Map<String, String> labels =
container.getLabels();
                    DatabaseInstance databaseInstance = new
DatabaseInstance();
                    databaseInstance.setName(labels.get("name"));
                    databaseInstance.setPort(labels.get("port"));
                    databaseInstance.setPass(labels.get("pass"));
                    databaseInstance.setImage_name(labels.get("image_n
ame"));
```

```
                    databaseInstance.setDatabase(labels.get("database"
));
                    databaseInstance.setContainerId(container.getId())
;
                    databaseInstances.add(databaseInstance);
                });

        return databaseInstances;
    }

    public void removeContainer(String containerId) throws
DockerException, InterruptedException {
        dockerClient.removeContainerCmd(containerId).exec();
    }

    public boolean isContainerRunning(String containerId) throws
DockerException {
        List<Container> containers = dockerClient.listContainersCmd()
                .withShowAll(true)
                .withIdFilter(Collections.singletonList(containerId))
                .exec();
        return containers
                .stream()
                .anyMatch(container ->
container.getState().equals("running"));
    }
}

// DatabaseInstance.java – Клас що інкапсулює дані про екземпляр бази
даних
package org.sumdu.models;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import java.util.Map;

@JsonIgnoreProperties({"jdbcStr"})
public class DatabaseInstance {
    private String database;
    private String image_name;
    private String name;
    private String port;
    private String pass;
    private String containerId;
    private Map<String, String> envs;

    public DatabaseInstance(String database, String image_name, String
name, String port, String pass, Map<String, String> envs) {
        this.database = database;
        this.image_name = image_name;
        this.name = name;
```

```
        this.port = port;
        this.pass = pass;
        this.envs = envs;
    }

    public DatabaseInstance() {
    }

    public String getDatabase() {
        return database;
    }

    public String getImage_name() {
        return image_name;
    }

    public String getName() {
        return name;
    }

    public String getPort() {
        return port;
    }

    public String getPass() {
        return pass;
    }

    public String getContainerId() {
        return containerId;
    }

    public Map<String, String> getEnvs() {
        return envs;
    }

    public String getJDBCStr() {
        return switch (getDatabase().toLowerCase()) {
            case "postgresql", "db2" -> String.format(
                    "jdbc:%s://localhost:%s",
                    getDatabase().toLowerCase(),
                    getPort()
            );
            case "mysql", "mariadb" -> String.format(
                    "jdbc:%s://localhost:%s?allowPublicKeyRetrieval=tr
ue&useSSL=False;",
                    getDatabase().toLowerCase(),
                    getPort()
            );
            default -> throw new IllegalArgumentException("Unsupported
database type: " + getDatabase());
```

```
        };
    }

    public void setDatabase(String database) {
        this.database = database;
    }

    public void setImage_name(String image_name) {
        this.image_name = image_name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setPort(String port) {
        this.port = port;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    public void setContainerId(String containerId) {
        this.containerId = containerId;
    }

    public void setEnvs(Map<String, String> envs) {
        this.envs = envs;
    }
}


// EnvHelper.java – Допоміжний клас, що обробляє змінні оточення
package org.sumdu.helpers;

import org.sumdu.models.DatabaseInstance;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class EnvHelper {
    public static String[] getEnvVars(DatabaseInstance database) {
        List<String> envVars = new ArrayList<>();
        Class<?> clazz = database.getClass();
        for (Map.Entry<String, String> entry :
database.getEnvs().entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();
```

```
            try {
                Method method = clazz.getDeclaredMethod(value);
                Object methodResult = method.invoke(database);
                envVars.add(key + methodResult);
            } catch (NoSuchMethodException e) {
                envVars.add(key + value);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return envVars.toArray(new String[0]);
    }
}


// JSONHelper.java – Допоміжний клас, що допомагає зчитувати
конфігурацію з .json файлу
package org.sumdu.helpers;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.sumdu.models.DatabaseInstance;

import java.io.File;
import java.io.IOException;
import java.util.List;

public class JSONHelper {
    private static final ObjectMapper objectMapper = new
ObjectMapper();

    public static void
writeDatabaseInstancesToFile(List<DatabaseInstance> databaseInstances,
String filePath) throws IOException {
        objectMapper.writerWithDefaultPrettyPrinter().writeValue(new
File(filePath), databaseInstances);
    }

    public static List<DatabaseInstance>
readDatabaseInstancesFromFile(String filePath) throws IOException {
        return objectMapper.readValue(new File(filePath), new
TypeReference<List<DatabaseInstance>>() {});
    }
}

// AlertHelper.java – Допоміжний клас для відображення повідомлень та
попереджень.
package org.sumdu.helpers;

import javafx.scene.control.Alert;
```

```
public class AlertHelper {
    public static void showAlert(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```