Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Розробка застосунку для збору даних про навколишнє середовище та аналізування його поточного стану»
за освітньою програмою «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконала: студентка групи «ПЗ1911»

_____ /Дарія ЗЕЛЕНЬКО/
(підпис студента)

Керівник: _____ /доц. Олександра ГОРБОВА/
(підпис)

Нормоконтролер: _____ / доц. Світлана ВОЛКОВА /
(підпис)

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

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis
master

on the topic: «Development of an application to collect environmental data and analyze its current state»

according to educational curriculum «**Software engineering»**
in the Speciality: «**121 Software engineering»**

Done by the student of the group PZ1911: _____ /Dariia ZELENKO/

Scientific Supervisor: _____ /Olexandra GORBOVA/

Normative controller: _____ / Svitlana VOLKOVA/

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри __КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 202_ р.

ЗАВДАННЯ

На кваліфікаційну роботу __Бакалавр

студенту Зеленько Дарії Миколаївні

1. Тема дипломної роботи: «Розробка застосунку для збору даних про навколишнє середовище та аналізування його поточного стану»

   Керівник роботи: Горбова Олександра Вікторівна

   затверджені наказом № 1209 ст від 07.12.2022
2. Строк подання студентом роботи ___.___. 202_ року
3. Вихідні дані до дипломної роботи: _____
4. Зміст пояснювальної записки:
   4.1. Аналітична частина: Збір та аналіз вимог до програми, побудова процесів взаємодії з програмою;
   4.2. Основна частина: Проектування та розробка системи, вибір парадигми та мови програмування, підбір засобів обробки та збереження даних, розробка архітектури системи, документування та опис функціональних можливостей;
   4.3. Тестування: Провести процес тестування та налагодження розроблюваного застосунку;

5. Перелік графічного матеріалу: Структура архітектури та компонентів системи, приклади оцінки ризику, таблиця кольорової ідентифікації ризику, елементи гео-картування, веб-інтерфейс станцій моніторингу.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір та аналіз вимог	03.07.22	15%
2	Зовнішнє проектування	17.07.22	35%
3	Внутрішнє проектування	15.12.22	50%
4	Розробка алгоритмів	22.03.23	60%
5	Розробка програмного забезпечення	08.04.23	80%
6	Тестування програмного забезпечення	23.05.23	90%
7	Подання кваліфікаційної роботи до кафедри	13.06.23	100%
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	26.06.23	

Студент _____ /Дарія ЗЕЛЕНЬКО/

Керівник роботи _____/Олександра ГОРБОВА/

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра 86 с., 53 рис., 2 додатки, 59 джерел.

Об'єкт розробки – Програмне забезпечення для збору даних про навколишнє середовище та аналізування його поточного стану

Мета роботи – Розробка інформаційно-аналітичної системи розрахунку потенційних екологічних ризиків їх аналітики та ранжирування для координування та прийняття управлінських рішень, на підставі отриманої від системи громадського моніторингу інформації.

Ключові слова: аналіз стану навколишнього середовища, оцінка ризиків, прогнозування ризиків для здоров'я, інтерфейс програмування застосунків, моніторинг стану повітря.

ЗМІСТ

ВСТУП 13
1 ЗБІР ТА АНАЛІЗ ВИМОГ 17
1.1. Методи збору вимог до програмного забезпечення 17
1.2. Прототипування 20
1.3. Бізнес-процеси 24
1.4. Опис аналогів 28
2 ПРОЕКТУВАННЯ 30
2.1. Зовнішнє проєктування 30
2.2. Внутрішнє проектування 32
2.2.1 Проектування архітектури системи 40
2.2.2 Проектування інтерфейсу користувача 46
2.2.3 Вибір парадигми проектування 49
2.2.4 Проектування динаміки системи 50
2.2.5 Проектування системи на фізичному рівні 54
2.2.6 Проектування бази даних 55
3 РОЗРОБКА 58
3.1.1. Алгоритм, його властивості, засоби описування 58
3.1.2. Графічні засоби описування алгоритмічних структур 61
3.1.3. Види алгоритмічних структур 63
3.2. Метод покрокової деталізації 64
4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ 68
ВИСНОВКИ 81
БІБЛІОГРАФІЧНИЙ СПИСОК 82
ДОДАТКИ 88

ПЕРЕЛІК УМОВНИХ ОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Референтна доза - допустиме добове надходження хімічної речовини протягом усього життя (яке виражається в мг/кг-день), яке не супроводжується відчутним ризиком для здоров'я; встановлюється на основі всіх фактів, що є на час проведення токсикологічної оцінки.

Референтна зона – зона контрольних досліджень, у якій мінімальне забруднення, прийняте за зразок-еталон (контрольна зона);

Референтна концентрація - допустиме добове надходження хімічної речовини протягом усього життя (для оцінки інгаляційної токсичності в мг/м3-день), яке встановлюється з урахуванням усіх наявних сучасних наукових даних, яке не супроводжується відчутним ризиком для здоров'я.

Гранично допустима концентрація (ГДК) - затверджений у законодавчому порядку санітарно-гігієнічний або рибогосподарський норматив, це максимальна концентрація хімічних елементів та їх сполук у навколишньому середовищі, яка при повсякденному впливі протягом тривалого часу на організм людини не викликає патологічних змін або захворювань, що встановлюються методами досліджень, у будь-які терміни життя сьогодення та наступного поколінь.

Екологічне нормування - є процес визначення видів, розмірів, змісту шкідливих впливів на навколишнє середовище в цілому або на окремі середоутворюючі елементи, що дозволяє гарантувати не заподіяння шкоди життю та здоров'ю людини, іншим правом, що охороняється об'єктам.

Фактори ризику – це такі вроджені чи набуті специфічні особливості зовнішнього чи внутрішнього середовища організму, які формують підвищену ймовірність розвитку шкідливих шляг організму реакцій за наявності певного потенційно шкідливого впливу.

Методологія оцінки ризику - це систематичний підхід до визначення та аналізу потенційних ризиків, пов'язаних з впливом забруднення повітря на здоров'я людей. Ця методологія використовується для оцінки можливих наслідків забруднення повітря, таких як вплив шкідливих речовин, хімічних речовин або токсичних газів на здоров'я людей.

Соціально-гігієнічний моніторинг (СГМ) – державна система спостереження, аналізу, оцінки та прогнозу стану здоров'я населення та довкілля людини, а також визначення причинно-наслідкових зв'язків між станом здоров'я населення та впливом факторів довкілля людини, є, з одного боку, засобом управління ризиками (у тому числі шляхом моніторингу експозицій та ризиків, динамічного стеження за прямими і непрямими індикаторними показниками). Коригує принципи та критерії характеристики ризиків та надає відомості про реальні концентрації хімічних речовин в об'єктах довкілля людини, фактори експозиції та інших. У цьому плані методологію оцінки ризику можна як одного з основних, системообразующих елементів сгм.

API (Application Programming Interface) - це набір правил та протоколів, які дозволяють різним програмам взаємодіяти між собою. Воно визначає набір функцій, методів та даних, які можна використовувати для створення програмного забезпечення або взаємодії між різними програмами. API дозволяє розробникам використовувати функціональні можливості іншого програмного забезпечення без необхідності розуміти всі деталі його реалізації. Воно встановлює стандарти та правила для передачі даних між програмами, що робить їх взаємодію більш простою та надійною. API може бути представленим у різних форматах, таких як веб-сервіси, бібліотеки програмного забезпечення, протоколи комунікації та інші. Використання API дозволяє розробникам ефективно використовувати готові компоненти та сервіси, спрощуючи розробку програмного забезпечення, прискорюючи процес та полегшуючи взаємодію між різними системами.

REST API (Representational State Transfer Application Programming Interface) - це стандартний архітектурний стиль для розробки веб-сервісів, що ґрунтується на принципах REST. Використання REST API дозволяє створювати легкодоступні, масштабовані та розширювані веб-сервіси. Він використовує HTTP методи, такі як GET, POST, PUT та DELETE, для взаємодії з ресурсами. Кожен ресурс має свій унікальний ідентифікатор, який передається у URL-адресі запиту. Дані передаються у репрезентаційних форматах, таких як JSON або XML. REST API також працює в безстановному режимі, не зберігаючи стан клієнта між запитами, що сприяє масштабованості та надійності. Кешування дозволяє зберігати копії відповідей,

зменшуючи навантаження на сервер та покращуючи продуктивність. REST API є широко використовуваним стандартом для створення веб-сервісів, що забезпечує ефективну комунікацію між клієнтськими та серверними додатками.

Spring - це потужний фреймворк для розробки програмного забезпечення на платформі Java. Він надає комплексний підхід до створення додатків, пропонуючи різноманітні інструменти та модулі для різних аспектів розробки. Spring дозволяє розробникам будувати масштабовані, гнучкі та легко тестовані додатки, забезпечуючи інверсію керування, керування транзакціями, обробку веб-запитів, доступ до бази даних та багато іншого. Він також надає підтримку для використання інших популярних технологій і фреймворків, таких як Hibernate, JPA, Spring MVC, Spring Security та інші. Spring є одним з провідних фреймворків у світі Java, заснованим на принципах інверсії керування та аспектно-орієнтованого програмування.

Spring Boot - це фреймворк, що базується на фреймворку Spring, і призначений для спрощення розробки програмного забезпечення на платформі Java. Він надає зручність у конфігурації та створенні самостійних додатків, забезпечуючи готовість до використання налаштування за замовчуванням і автоматичне конфігурування. Spring Boot дозволяє розробникам швидко створювати додатки, що легко масштабувати та тестувати. Він поєднує у собі потужність фреймворка Spring зі зручністю та простотою використання.

ВСТУП

Актуальність роботи. Актуальність проекту полягає в тому, що на даний момент у сучасному світі тема турботи про стан довкілля та майбутнього людства порушується все частіше. Суспільство приходить до того, що довкілля дуже впливає на людину, тому для забезпечення безпеки та здоров'я людських індивідів повинні вводитися певні стандарти контролю якості таких необхідних для життя ресурсів як вода, повітря і грунт. Це дуже непросте завдання, яке потребує великої кількості дослідницької роботи. За роки розвитку цього напряму було виведено формули та алгоритми для оцінки показників життєво-важливих компонентів. І найціннішим з них є повітря, стан повітря дуже складно контролювати і підтримувати на певному рівні, на це впливає велика кількість зовнішніх і внутрішніх факторів таких як: кількість виробничих підприємств у зоні, густота населення, кліматичні особливості та інше.

З метою спостереження за станом повітря у світі, а зокрема в Україні, було започатковано багато проектів, метою яких є моніторинг показників чистоти повітря, вмісту та концентрації низки хімічних речовин та інших показників, що впливають на оцінку стану атмосфери. Подібні проекти за рахунок збору інформації та динаміки зміни показників надають цінну аналітику, яка допоможе змінити ситуацію в проблематичних зонах на краще, звертаючи увагу на проблему та мотивуючи державу та мешканців вживати заходів, бо екологія – наше майбутнє. Однак навіть аналітики подібних сервісів не вистачає для адекватної оцінки ситуації, пошуку можливих рішень та розрахунку впливу результатів на якість життя людей, адже в результаті це лише цифри, які так само мають мало реального застосування.

У цій дипломній роботі буде розроблено додаток, призначений для вирішення саме цього ряду завдань. Додаток буде доставляти повну та деталізовану розшифровку за даними, які вона обробила, що виділяє його серед альтернативних рішень та робить інформацію доступною навіть для користувача без специфічних знань у даному напрямку. Це допоможе звернути увагу якомога більшої кількості людей на проблеми екології та залучити громадськість до їх вирішення. Тому що стан повітря не повинен бути далекою і абстрактною темою, це дуже актуальна проблема

на сьогоднішній день і яка залишатиметься такою і в найближчому майбутньому.

Сутність проблеми дослідження. Якість повітря в основній зоні життя людини дуже позначається на її здоров'ї і за допомогою методів оцінки ризиків можна розрахувати можливі ризики виникнення тих чи інших захворювань людини. Але подібні алгоритми потребують автоматизації, тому їх реалізація у вигляді гнучкого програмного комплексу-сервісу для інтеграції з різними платформами з метою розробки екологічних рішень, що відповідають усім медичним рекомендаціям, – необхідна: промисловим підприємствам, медичним комісіям, державним установам та науковим товариствам. В останні роки в сучасній науці виник новий напрямок, пов'язаний з виявленням та оцінкою факторів ризику розвитку несприятливих змін у людини як на популяційному, так і на індивідуальному рівні. Відомо, що людина у своїй повсякденній діяльності стикається з величезною кількістю потенційних факторів ризику.

Завданням екології є виявлення, оцінка та розробка найбільш ефективних заходів щодо усунення тих факторів ризику, які можна усунути, та щодо мінімізації до безпечного, прийнятного рівня тих факторів ризику, якими суспільство здатне керувати.

Методологія оцінки ризику для здоров'я людини, пов'язаного з впливом факторів навколишнього середовища, включає аналіз фізичних, у тому числі радіаційних факторів, хімічних сполук, що забруднюють атмосферне повітря населених місць, ґрунт, воду водних об'єктів, харчові продукти, численних біологічних факторів (наприклад, мікробіологічний ризик), шкідливих та небезпечних факторів виробничого середовища та трудового процесу (професійний ризик). Характеристика ризику заснована на результатах оцінки рівнів експозиції в популяції, що вивчається, а також на новітніх наукових експериментальних, клінічних та епідеміологічних даних про шкідливі ефекти, викликані даним фактором у людини, і параметри залежностей.

Суттєве значення для подальшого розвитку програмної інженерії:

Так як на даний момент подібних програм дуже мало, а попит все зростає, то розробка такого цілісного продукту може сприяти загальному розвитку даного

напряму. А саме, запропонувати унікальне рішення для завдання, що стоїть перед розробником. Так як додаток поєднує в собі функціонал, який раніше надавалися тільки у вигляді розрізнених компонентів, а в даній реалізації розрахункові дані обробляються в реальному часі для різних станцій моніторингу, підсумкові дані «розширюють» інформацію з датчиків і дають повний опис результатів розрахунку, зрозуміле не тільки вузько спрямованому спеціалісту, а й звичайному середньостатистичного користувача. Що є дуже новим для діапазону вже існуючих рішень і може розширити межі використання такого програмного забезпечення.

Значення для створення нових напрямів програмної інженерії. Завдання та виклики такого рівня створюють гібридну сферу еко-кібернетики, яка є синтезом продуктів на основі екологічних методів у комбінації із засобами та інструментами, які може надати сфера інформаційних технологій. Тобто дві різні сфери розширюють свої межі застосування, створюючи під собою свою власну базу знань і рішень. Щодо більш вузької класифікації, додаток, що розробляється, бере за свою основу автоматизацію процесів розрахунку потенційного ризику для здоров'я людини, враховуючи дослідження з галузі медицини, норми концентрації речовин з хімії і самі розроблені алгоритми з екології. При більш детальному розгляді дійсно помітно куди більша кількість взаємозв'язків між різними прикладними науками та результаті подібного «перетину», з'являється можливість проектування низки додатків з метою вирішити завдання, що стоять в даний момент перед наукою, а зокрема вирішення проблеми здоров'я населення, через процес контролю та спостереження. за якістю рівня повітря, відстеження та запобігання розвитку низки захворювань у жителів тих чи інших областей. Крім того, продукт, що розробляється, проектується з урахуванням подальшої можливості інтеграції у вигляді незалежного сервісу з іншими додатками подібної спрямованості для розширення їх функціоналу та можливостей. Що також дозволить розвивати діапазон вже присутніх рішень даної задачі на основі вже запропонованої реалізації. І це відіграє дуже велику роль у створенні та розвитку нових течій, оскільки на даний момент у доступності ми маємо не так багато програмних засобів, які могли б не тільки розрахувати, але й здійснити прогнозування можливих наслідків від знаходження та проживання у зонах високої

концентрації низки хімічних. речовин у повітряному середовищі. Крім прогнозування потенційних ризиків, додаток також проводить оцінку стану навколишнього середовища на основі розрахованих ризиків та результатів прогнозів.

Доцільність роботи, її відмінність порівняно з відомими розв'язаннями проблеми. Основна відмінність полягає в тому, що хоч програми для фактичного розрахунку ризику існують, жодна з них не відображає інформацію в реальному часі, взаємодіючи з реальними верифікованими системами моніторингу якості повітря. Вся аналітика, яку можу надати альтернативні рішення, є статичною, у нашому випадку додаток постійно проводить сканування станцій, зберігає інформацію про показники і веде статистику показників. Щодо систем моніторингу, вони не надають таку повну інформацію щодо хімічних речовин, моніторинг яких вони ведуть. Ця програма є об'єднанням кількох підходів, якого дійсно потребує кінцевий користувач.

Мета роботи. Розробка інформаційно-аналітичної системи розрахунку потенційних екологічних ризиків їх аналітики та ранжирування для координування та прийняття управлінських рішень, на підставі інформації отриманої від системи громадського моніторингу Eco City[1].

Експлуатаційне призначення. Використання програмного забезпечення як незалежного мікро-сервісу для отримання додаткової аналітики кінцевими користувачами: фахівцями-екологами (аналітика та моніторинг довкілля); медичним персоналом (розробка рекомендацій для чутливих груп населення); громадськістю (інформування, забезпечення особистої безпеки). Обрана реалізація дає високу гнучкість, а саме, спрощує обслуговування самого додатка та інтеграцію з іншими ресурсами, що є основними критеріями в процесі експлуатації даного програмного забезпечення. І як було згадано раніше, призначення цього додатка полягає в розрахунку даних про ризик, на підставі отриманої від системи громадського моніторингу інформації, кінцева аналітика цих даних та подальше їх відправлення користувачеві через внутрішній доступ по арі токену.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

1.1. Методи збору вимог до програмного забезпечення

Методи збору вимог до програмного забезпечення грають важливу роль у розробці ефективних програмних продуктів. Один із таких методів - це метод бесіди, який дозволяє отримати цінну інформацію від зацікавлених сторін, таких як клієнти, користувачі або експерти галузі.

Метод бесіди дозволяє розробникам отримати глибоке розуміння про потреби та вимоги користувачів. Вони можуть поставляти питання, отримувати пояснення і з'ясовувати деталі, що допомагає уникнути непорозумінь та неточностей. Бесіда може розкрити вимоги, які можуть бути прихованими або неочевидними. Розмова з зацікавленими сторонами дозволяє виявити незадекларовані потреби та додаткові функції, які можуть бути корисними для користувачів. Бесіда є інтерактивним процесом, що дозволяє розробникам задавати питання, вносити зауваження та виразити свої думки. Це стимулює діалог між розробниками і зацікавленими сторонами, що сприяє кращому зрозумінню вимог і можливостей програмного продукту. Бесіда може допомогти вирішити конфліктні вимоги або протиріччя між різними зацікавленими сторонами. Шляхом обговорення і встановлення пріоритетів можна знайти компромісні рішення та досягти згоди. Бесіда може виявити обмеження, такі як технічні, бюджетні або ресурсні. Розробники можуть отримати уявлення про можливості та обмеження реалізації вимог і знайти оптимальні рішення.

Метод бесіди дозволяє враховувати думки та потреби користувачів, що допомагає створити продукт, що задовольняє їх очікування. Це може підвищити задоволення від використання програмного забезпечення та покращити його прийняття користувачами.

Загалом, метод бесіди є ефективним інструментом для збору вимог до програмного забезпечення, оскільки він сприяє взаєморозумінню, виявленню прихованих вимог і створенню користувач орієнтованого продукту.

За допомогою способу опитування, саме розмови із замовниками і користувачами та способами вивчення документів, були вивчені інформаційно-технічні, методичні, технічні та технологічні документи. І в результаті були

розроблені вимоги до проекту, що розробляється.

Мета і область дослідження. З метою оцінки екологічної ситуації, на конкретній території, пов'язаній з якістю атмосферного повітря та станом здоров'я населення, використовуються різні показники: індекси стану атмосфери та ризики виникнення та реалізації негативних наслідків, пов'язаних із забрудненням атмосфери. Одним з основних показників якості атмосфери, не пов'язаний з радіоактивним забрудненням, є показник "ризик розвитку неканцерогенних ефектів", який можна розрахувати шляхом мануального розрахунку слідуючи формулам або використовуючи оперативну інформацію онлайн-моніторингу Eco city[1] концентрації різних токсичних речовин в атмосферному повітрі.

Вибір учасників:

Участь у бесіді брали майбутні користувачі та експерти галузі.

Звіт зібраних вимог:

Аналіз інформації, отриманої від користувача:

Профіль користувача:

Вік: від 20 років.

Стать: М/Ж

Освіта: Медична/технічна.

Експертні знання: Екологія та технології захисту навколишнього середовища.

Кваліфікаційний рівень: молодший спеціаліст, бакалавр, магістр.

Наявність практичного досвіду роботи у предметній галузі: необов'язковий.

Рівень володіння комп'ютерною технікою та іншим необхідним обладнанням: - рівень користувача.

Навички роботи з різним програмним забезпеченням: пакет програм Office, вміння користуватись Internet.

Спеціальних вимог, фізичних обмежень: відсутні.

Аналіз завдань, які стоять перед користувачем:

За допомогою аналітичних програмних модулів визначає розрахункові значення ризиків, може диференціювати отримані результати за допустимими нормативними показниками та визначити зони несприятливих та небезпечних значень.

Найбільш важливими завданнями користувача є інформація про показники неканцерогенних, потенційних та комбінованих ризиків.

Вихідну інформацію для обробки (концентрації, дози, показники індексу забруднення) отримують з офіційного сайту цивільного моніторингу, проводять розрахунок неканцерогенних ризиків, виконують оцінку потенційних та комбінованих ризиків, порівнюють з нормативними значеннями та показниками, визначають речовини та зони з наднормативними показниками, виконують колірну диференціацію. показників за величиною перевищень, наводять рекомендації фахівців та експертів.

Мета користувача: визначити зони та речовини з наднормативними показниками ризиків.

На виконання завдання необхідно мати: концентрації, дози, показники індексу забруднення основних промислових токсичних речовин, нормативні значення цих показників, аналітичні моделі до розрахунку ризиків.

Очікуваний результат від вирішення завдання: Розрахункові значення ризиків для основних промислових регіонів України щодо основних речовин, зони перевищень, елементи геокартування, можливість надання оперативної інформації з метою розробки рекомендацій для особливо чутливих груп населення щодо мінімізації негативного впливу на здоров'я.

Під час виконання завдання користувач взаємодіє з іншими особами за допомогою ПК: онлайн-консультацій, онлайн-інформування.

За допомогою віддаленого сервера платформи моніторингу є можливість цілодобового використання та обробки даних. Для складання звітності та динаміки моніторингу: щоденно, щотижня, щомісяця – на запит.

Наявність інтегрованих аналітичних програмних модулів дозволяє обробляти значні масиви даних та видавати важливу оперативну інформацію для професіоналів, фахівців та населення.

Користувач виконує свою роботу на комп'ютері.

Збір вимог користувача:

Користувачеві необхідні Методики розрахунку екологічних ризиків, технології

обробки масивів даних, гео-інформаційні технології

За програмний продукт готовий заплатити Від 10 000 грн. до 70000 грн. Залежно від можливостей програмного продукту.

Супроводжувати продукт буде група технічної підтримки та розробники.

Аналіз робочого середовища користувача:

Фізична сторона робочого середовища: як природне, і штучне освітлення не більше норм для офісних робочих місць; шум – до 70 дБ; температура - 19-25 ° С; ПК, кольоровий принтер та інша допоміжна офісна техніка; Інтернет, телефон для мобільного зв'язку.

Місце роботи користувача та його мобільність: робота в офісі, можлива робота віддалено (вдома) – стаціонарно.

Питання ергономіки та умов праці: рекомендовано раціональне зонування простору, ретельно продуманий дизайн інтер'єру, зручні меблі, оптимальне освітлення.

Особливі запити (рівень підготовки, фізичний стан, особливості вимови та можливі недоліки): можливість оперативної передачі даних.

Інтернаціоналізація та інші культурологічні умови (переклад, кольори, текст, повідомлення, іконки та інше): Англійська, використання колірної диференціації для зонування районів за показниками ризиків за загальноприйнятим стандартом позначення зон ризику, оперативні повідомлення.

1.2. Прототипування

Концепція цієї програми розроблялася і розвивалася протягом кількох років. Тобто такий прототип є еволюційним.

Початковий прототип являв собою просту програму мовою C++ для розрахунку та оцінки потенційного ризику здоров'ю населення при хронічному впливі забруднення атмосфери. Програма мала просту структуру, користувач мануально вводив значення концентрації речовин та далі програма проводила повний розрахунок за формулами та виводила результати на екран (рис. 1.1).

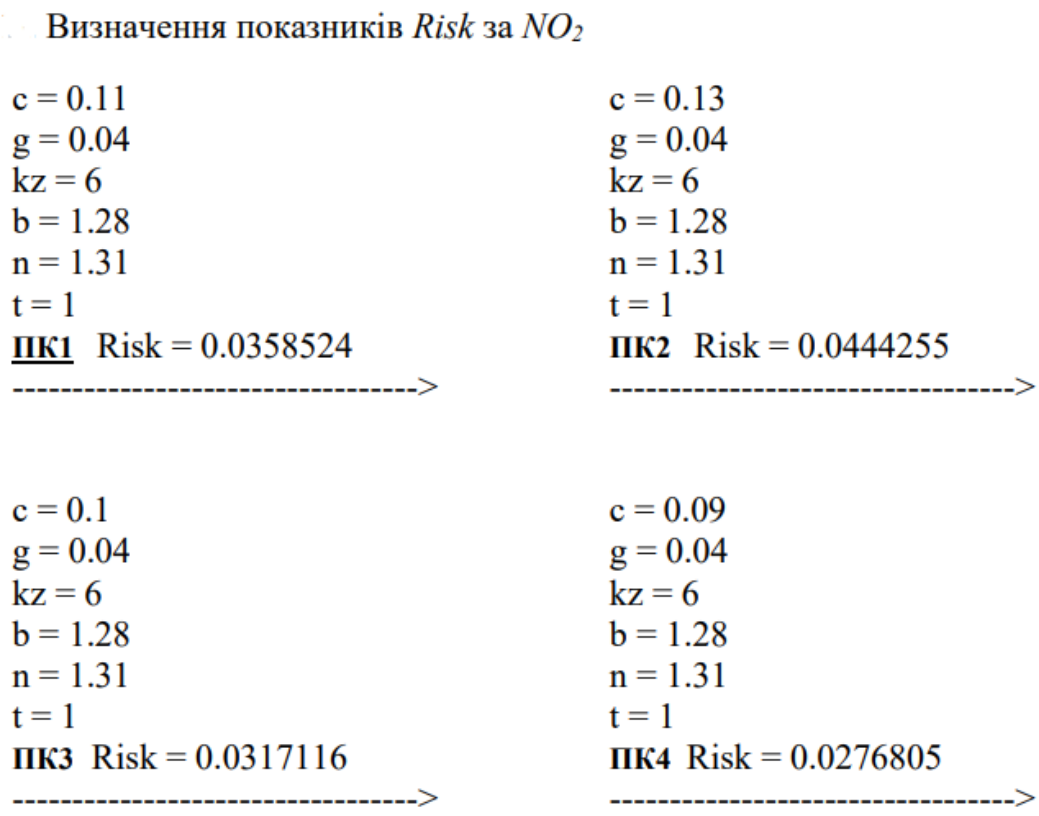


Рисунок 1.1 - Результат виконання першого прототипу програми

Далі був розроблений другий прототип програми, куди більш незалежний та багатофункціональний. Цей прототип також мав два підпрототипи:

Перший з яких базувався мовою програмування Python у поєднанні з фреймворком Selenium Web driver. Завданням такої програми було автоматичне сканування значень концентрації речовин у повітрі зі сторінки eco-city.org.ua та подальший їх розрахунок та аналітика всередині програми, дані про табличні концентрації зберігалися у статичних масивах самої програми.

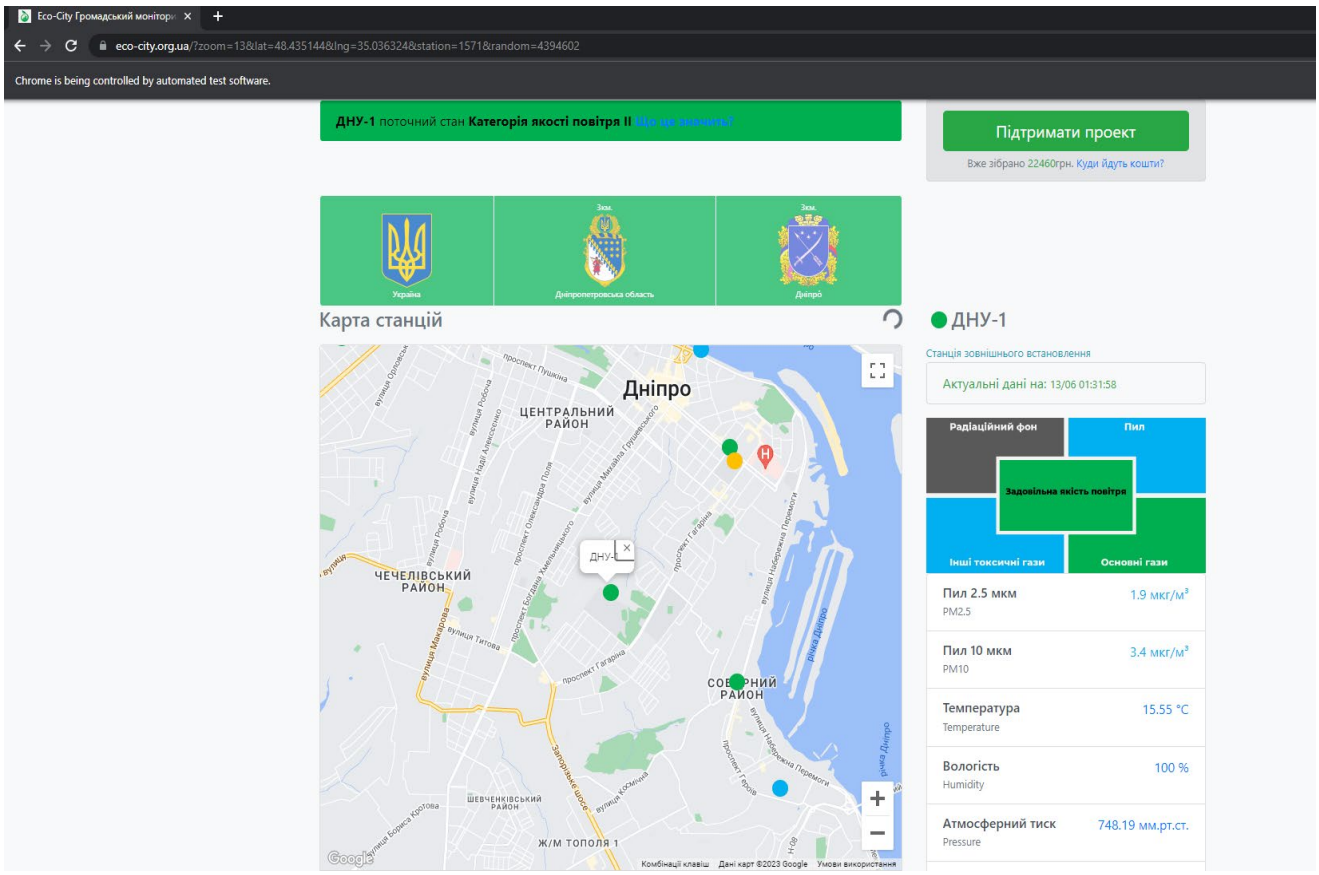


Рисунок 1.2 - Результат виконання другого прототипу програми

Другий прототип мав орієнтований на кінцевого користувача, мав візуальний інтерфейс, для якого була використана бібліотека PyQt. Процес роботи програми будувався таким чином: користувач міг мануально ввести концентрацію речовин і отримував результати розрахунку або скористатися автоматичною опцією, при виборі цього пункту меню програма запускала автоматичне сканування значень. Тобто другий прототип частково за структурою базувався на першому, але якість представлення даних та загальна структура програми були покращені та орієнтовані на використання даної програми як стаціонарного клієнта на персональному комп'ютері.

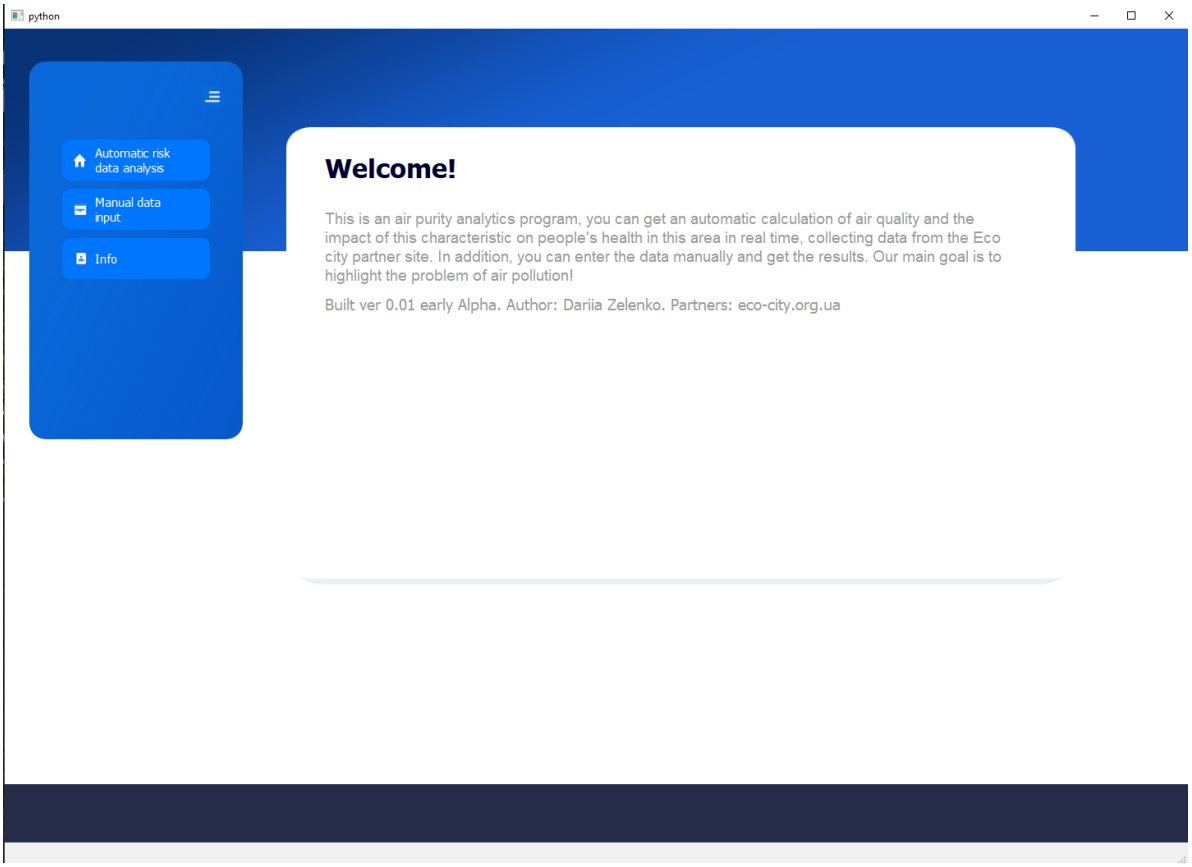


Рисунок 1.3 - Головна сторінка третього прототипу програми

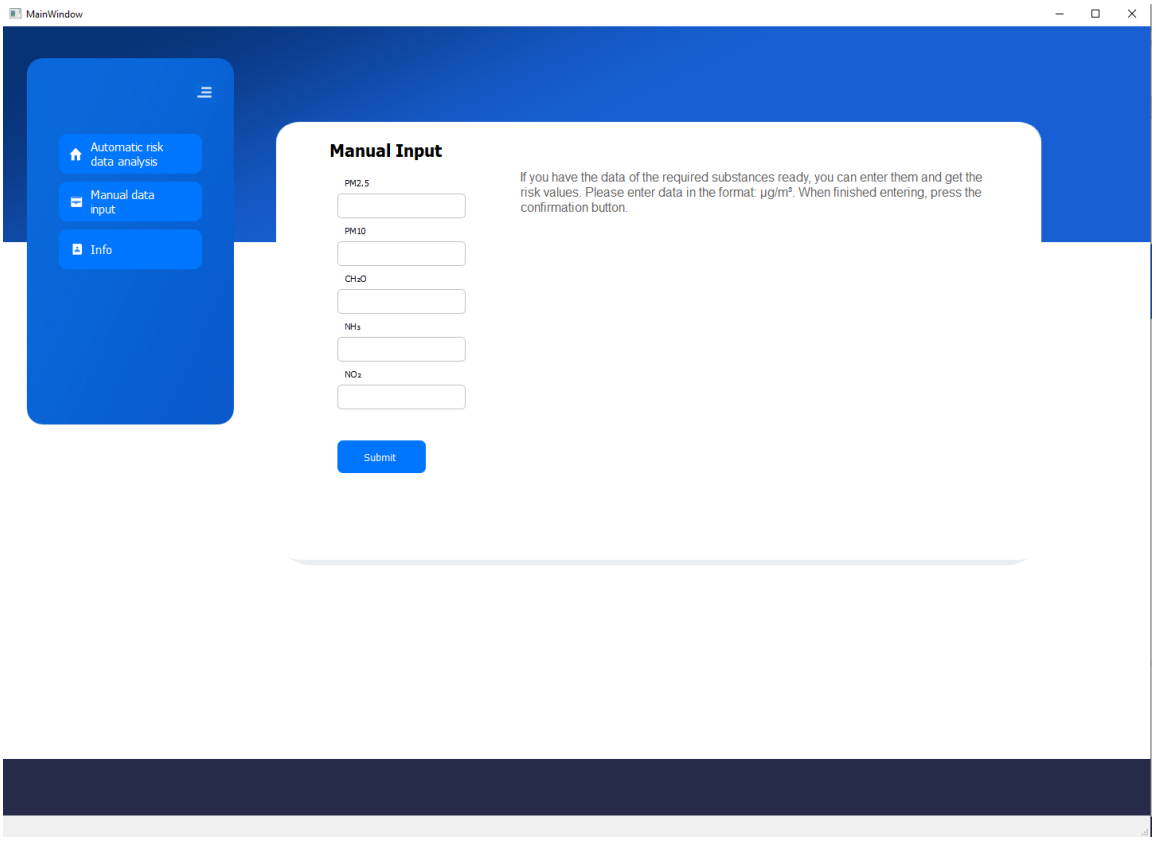


Рисунок 1.4 - Сторінка мануального введення даних третього прототипу програми

Рисунок 1.5 - Сторінка аналітики третього прототипу програми

1.3. Бізнес-процеси

Представлення бізнес-процесів у програмному додатку має велику важливість для успіху бізнесу. Організації у всіх галузях використовують програмні додатки для автоматизації та управління своїми процесами. Правильне представлення бізнес-процесів у програмному додатку дозволяє забезпечити ефективність, прозорість, керованість та вдосконалення роботи організації. Ось кілька ключових причин, чому це так важливо:

Ефективність: Представлення бізнес-процесів у програмному додатку дозволяє автоматизувати рутинні та повторювані завдання. Це допомагає збільшити продуктивність та ефективність роботи, оскільки комп'ютери можуть виконувати завдання швидше та без помилок, порівняно з ручним виконанням. Крім того, програмні додатки можуть оптимізувати послідовність процесів, що дозволяє знизити час виконання та забезпечити оптимальне використання ресурсів.

Прозорість: Візуалізація бізнес-процесів у програмному додатку дозволяє усім

зацікавленим сторонам (менеджмент, співробітники, клієнти) бачити та розуміти, як працюють різні етапи та взаємозв'язки в організації. Це створює прозоре середовище, де всі можуть спілкуватися та співпрацювати, розуміючи свою роль у загальному процесі. Прозорість сприяє зниженню помилок, усуненню зайвих витрат та покращенню комунікації.

Керованість: Представлення бізнес-процесів у програмному додатку надає можливість керувати та контролювати різні етапи процесів. Можна встановлювати терміни виконання, розподіляти завдання, відстежувати прогрес та оцінювати результати. Аналітичні інструменти в програмному додатку дозволяють збирати дані про продуктивність та ефективність процесів, що дає можливість аналізувати результати та приймати обґрунтовані рішення для поліпшення бізнесу.

Вдосконалення: Представлення бізнес-процесів у програмному додатку створює можливість для постійного вдосконалення та оптимізації. За допомогою програмного забезпечення можна аналізувати поточні процеси, виявляти слабкі місця, ідентифікувати можливості для автоматизації та удосконалення. Більш того, програмні додатки дозволяють впроваджувати зміни та тестувати нові ідеї без значного впливу на діючу систему.

Усе вищеозначене демонструє, наскільки важливим є представлення бізнес-процесів у програмному додатку. Це допомагає створити організацію, яка працює ефективно, забезпечує прозорість та контроль, а також постійно вдосконалюється для досягнення більшого успіху.

Перейдемо до опису бізнес-процесів.

Коли додаток отримує запит, проводиться опрацювання та валідація запиту, арі перевіряє правильність, тип запиту та валідність токену. На приклад, у даному випадку, програма очікує запит на отримання аналітики з онлайн сервісу, конкретну точку сканування, додаткові параметри та та правильний ключ, який дозволить доступ клієнту до точки-арі. Далі запит маршрутизується до логічного рівня, для формування та відправки запиту до зовнішнього інформаційного ресурсу, якщо доступ до даних не дозволено, сервер отримує повідомлення про відмову у доступі і тоді програма це обробляє та закінчує процес збору даних повертаючи повідомлення

про неуспішність спроби. Якщо доступ дозволено, програма отримує масив даних, відправляє його до іншого модулю для форматування та очистки, після чого дані будуть збережені у базу даних, далі проводиться розрахунок отриманої інформації за формулами, після чого вони зберігаються в іншу базу даних, звідти результати потрапляють на останній етап, кінцевою підготовки даних для відображення, далі дані повертаються клієнту та відображаються на екрані.

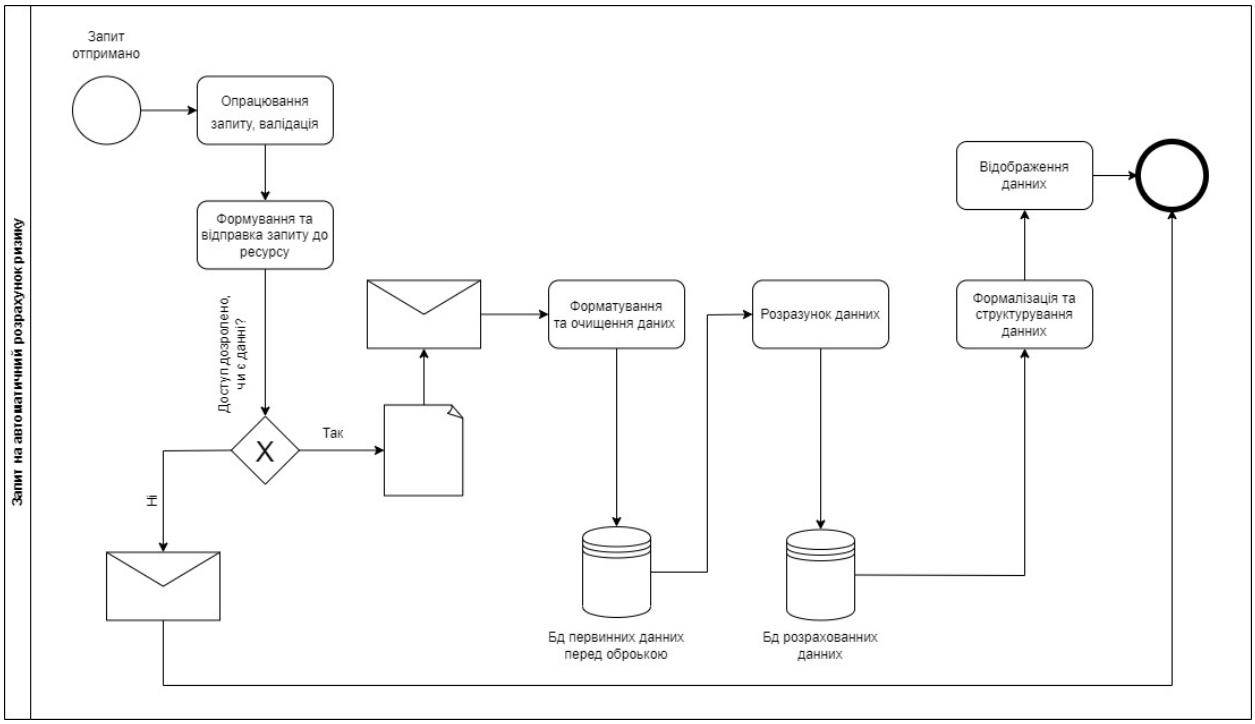


Рисунок 1.6 - Діаграма бізнес-процесів для запиту на автоматичний розрахунок

Контролер отримує запит та перевіряє його, відправляється запит до бази даних для отримання даних за деякий проміжок часу, після чого данні форматуються та структуруються та відправляються до клієнту.

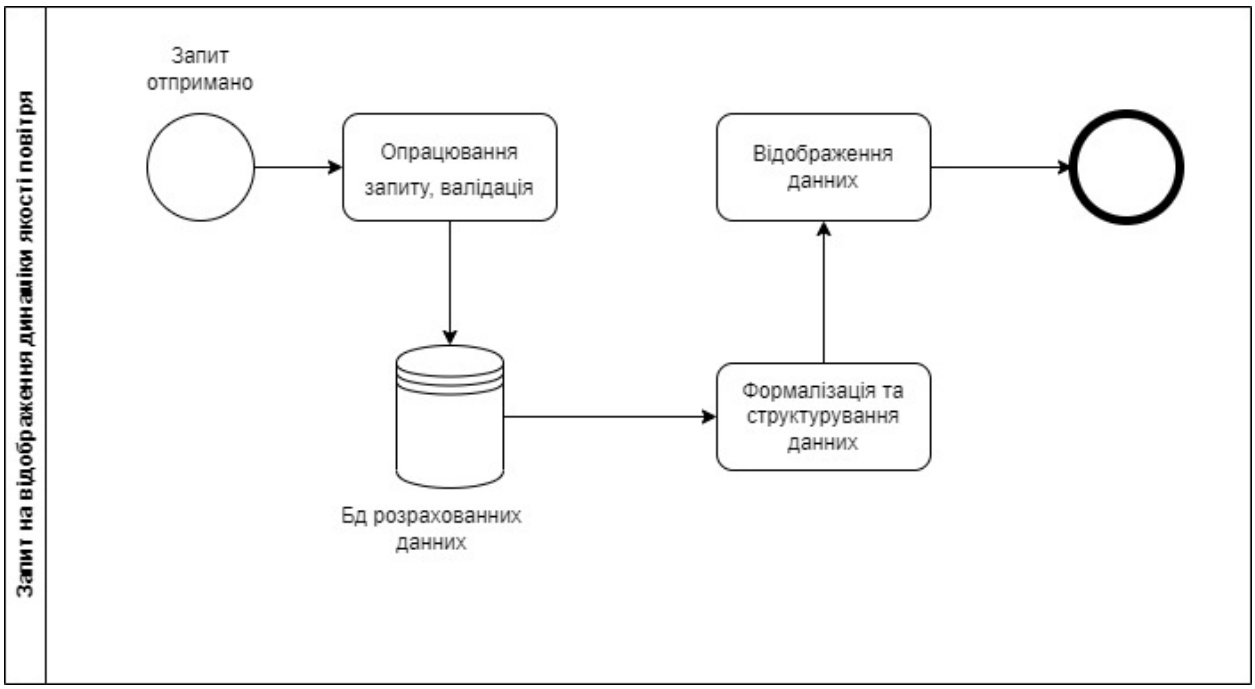


Рисунок 1.7 - Діаграма бізнес-процесів для запиту на відображення динаміки якості повітря

Сервіс отримує запит, оброблює його та перевіряє передані параметри, запит переспрямовується до методу обробки цієї операції, після чого проводиться валідація відправлених клієнтом даних, також в залежності від набору даних, будуть проведені відповідні розрахунки. Тому що, у програмі передбачено, що клієнт може відправити не повний набір даних, або відправити більше інформації, ніж обробляє програма. Після чого, дані форматуються та підготовлюються для відображення, та наприкінці, дані повертаються до клієнта та відображаються.

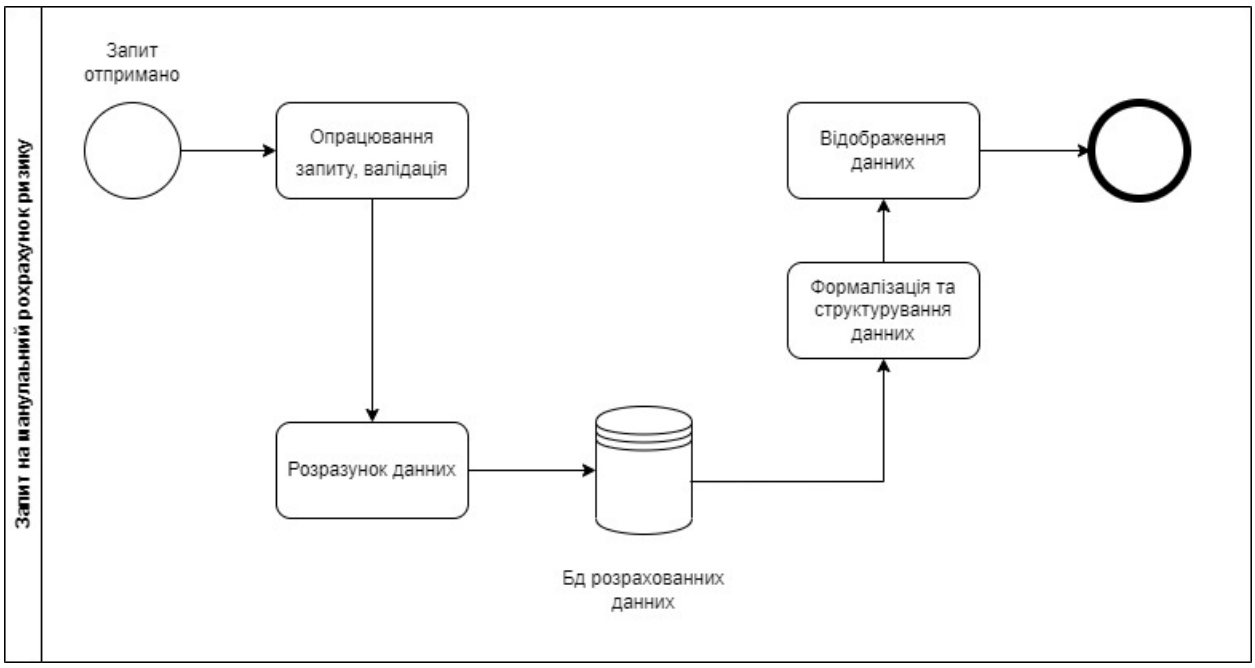


Рисунок 1.8 - Діаграма бізнес-процесів для запиту на розрахунок ризику з мануальним введенням даних

1.4. Опис аналогів

Основні критерії опису аналогів:

- Функціональні можливості розглянутої програми у порівнянні з запропонованим варіантом у цій науковій роботі
- Потенціал розвитку програми
- Якість подання інформації користувачу та її загальна зрозумілість та можливість використання

Список аналогів:

1. Програма експрес-оцінки ризиків здоров'ю населення[2]

Даний онлайн-додаток дозволяє проводити аналітику ризиків на основі мануально-введених показників кількості речовин на території, але результати розрахунків не показують глобальної аналітики та детального опису результатів. Якщо порівняти з запропонованою реалізацією, у загаданій програмі більш вузький спектр функціоналу. Крім того, програма може бути розширена, але краще було б, якщо це був би веб сервіс. Інформація подана у простому та зрозумілому форматі, але дуже лімітована, використовувати програму просто, інтерфейс дуже наочний та

зрозумілий.

2. Copernicus[3]

Copernicus створює області зонування за кольорами і показує індекс якості повітря, але не може розраховувати ризики для населення, що проживає в конкретних областях. Дуже зручний сервіс, який має дуже багато можливостей, інформація збирається у режимі реального часу. Програма не має можливості мануального обчислення показників. Цей веб додаток мае великий потенціал з точки зору масштабування.

3. World air quality index project[4]

На цій веб-сторінці візуалізовано показники з датчиків якості повітря, але аналітика та розширений спектр показників – відсутні. Даті структуровані зрозуміло та зручно, програма може мати ширший функціонал, але для цього потрібно буде переробити сам по собі додаток.

4. Sh**t! I Smoke[5]

Ця програма здатна проводити територіальну аналітику атмосферного забруднення, проте без конкретики, шкода здоров'ю описується у вигляді “викурених цигарок”. Це дуже не наочній спосіб опису даних. Мобільний додаток – це перспективний формат для популяризації подібних додатків. Але якщо порівнювати його з програмою, розробленою у цій науковій роботі, то другий варіант більш інформативний та має загалом більше можливостей.

5. Airq software tool for health risk assessment of air pollution[6]

Програма може оцінювати якість повітря, розраховувати індекс якості повітря та не більше. Такої інформації недостатньо для проведення наукових досліджень та прогнозування.

Висновки до розділу 1

Під час збору та аналізу вимог були сформовані основні вимоги до розробки програми, побудовані бізнес-процеси, проведена бесіда з замовником, користувачами та проведений огляд аналогів.

2 ПРОЕКТУВАННЯ

2.1. Зовнішнє проєктування

Функціональне призначення. Оцінка ризику розвитку неканцерогенних ефектів, оцінка сумарного ризику комбінованої дії кількох речовин, ймовірність розвитку неспецифічних токсичних ефектів та оцінка потенційного ризику здоров'ю населення за хронічного впливу.

Експлуатаційне призначення. Експлуатаційне призначення полягає у автоматизації алгоритмів оцінки ризику, необхідних для проведення наукових дослідів та аналітики.

Функціональні вимоги:

- Завдання концентрації шкідливих речовин у форматі параметрів запиту для подальшого розрахунку за формулами
- Відправка запиту на автоматичний збір концентрації речовин з ресурсу eco-city та збір даних
- Менеджмент розрахованих даних по станціях
- Відображення результатів розрахунків
- Отримання аналітики
- Аналітика даних у динаміці

Вхідні та вихідні дані:

Вхідні: значення показників пилу, формальдегіду, аміаку, оксиду азоту, монооксиду вуглецю, ідентифікатор станції, часовий період.

Вихідні: розрахований HQ, розрахований HQi за кожною речовиною, ймовірність розвитку неспецифічних токсичних ефектів при хронічній інтоксикації в заданих умовах; потенційний ризик для здоров'я населення при хронічному впливу кількох домішок забруднення атмосфери, аналітика по всіх розрахованих речовинах.

Виконаємо специфікацію функціональних вимог у вигляді прецедентів (рис. 2.1).

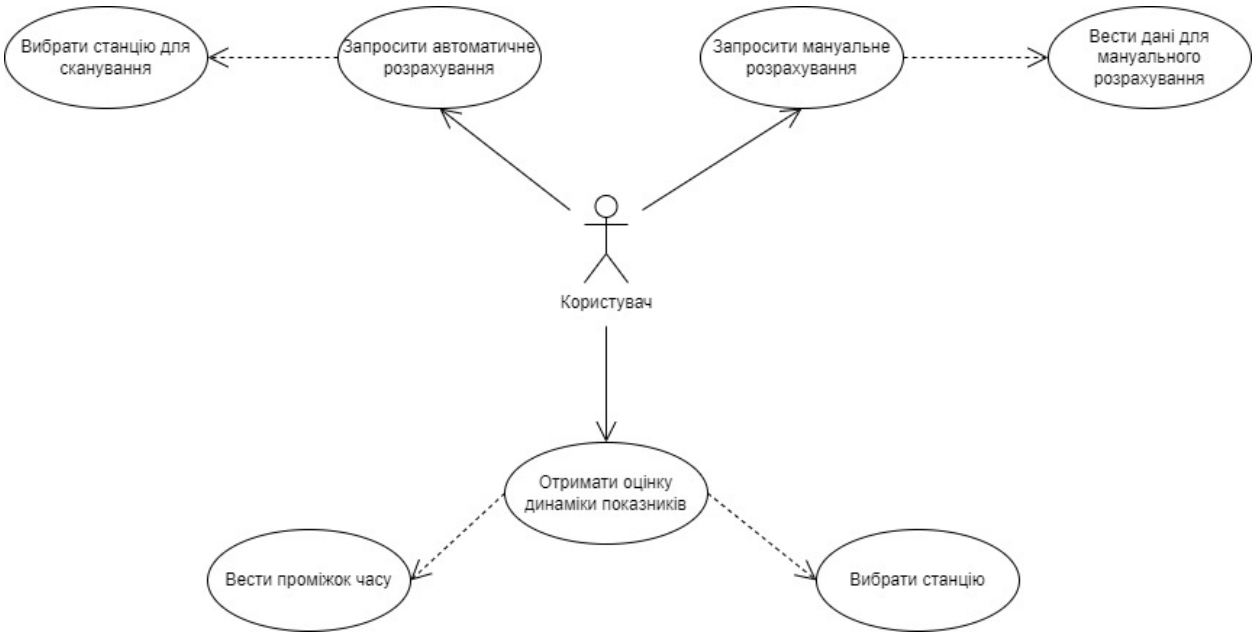


Рисунок 2.1 - Діаграма прецедентів

У користувача є всього три варіанти взаємодій, користувач може запросити автоматичне розрахування ризику за одною або декількома станціями моніторингу. Або запросити мануальне розрахування, тоді користувач повинен ввести всі показники, які будуть потрібні для обчислення. І третій варіант – це оцінка динаміки, в такому випадку, користувач вводить проміжок часу за яким хочу побачити аналітику і на основі цієї інформації отримає результат.

Опис зовнішнього інформаційного середовища, в якому буде експлуатуватися ПЗ:

Інфраструктура програми має такі компоненти:

1. Docker контейнер з Spring Boot додатком:
   - Контейнер містить готовий до використання Spring Boot додаток.
   - Зазначені параметри конфігурації для додатку, такі як порт, логін, пароль тощо.
2. Docker контейнер бази даних:
   - Контейнер містить базу даних, яку використовує Spring Boot додаток.
   - Зазначені параметри конфігурації для бази даних, такі як тип бази даних, адреса, порт, логін, пароль тощо.
3. Docker контейнер з Grafana:

- Контейнер містить сервіс візуалізації даних Grafana.
- Зазначені параметри конфігурації для Grafana, такі як адреса, порт, логін, пароль тощо.

4. Docker контейнер з Prometheus:
   - Контейнер містить сервіс збору та аналізу метрик Prometheus.
   - Зазначені параметри конфігурації для Prometheus, такі як адреса, порт, логін, пароль тощо.

Це інформаційне середовище забезпечує готову до використання систему, де Spring Boot додаток, база даних, Grafana та Prometheus знаходяться у відповідних Docker контейнерах з відповідними налаштуваннями. Контейнери можуть бути легко розгорнуті та запущені в одноразовому середовищі, що забезпечує простоту налаштування та реплікації середовища для розробки, тестування або продакшн використання.

Для запуску такого додатку, буде потрібен становлений та налаштований Docker на робочій станції або сервері, де планується запуск контейнерів. Docker дозволяє створювати, управляти та запускати контейнери. Також сам контейнеризований додаток з усіма конфігураційними файлами, таким чином контейнер може буде запущеним.

2.2. Внутрішнє проектування

Додаток відповідає багаторівневій архітектурі, у якій кожен рівень взаємодіє з іншими рівнями (вищими або нижчими в ієрархічному порядку).

У програмі є чотири рівні:

- Рівень презентації (Presentation Layer)
- Бізнес-рівень (Business Layer)
- Рівень збереження (Presistence Layer)
- Рівень бази даних (Database Layer)

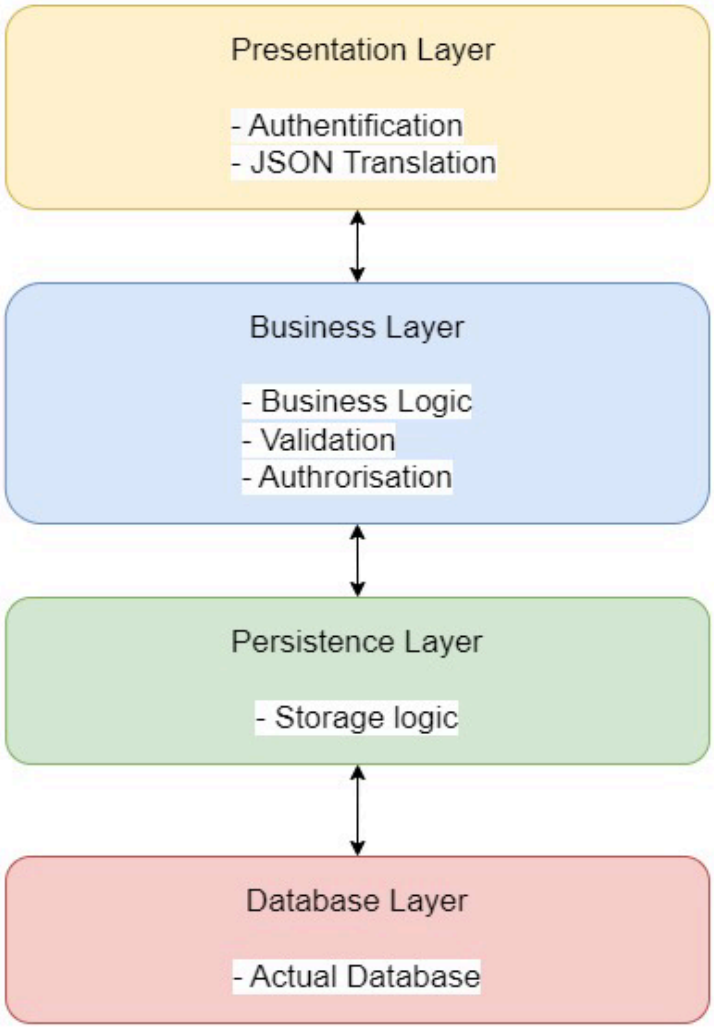


Рисунок 2.2 - Діаграма представлення рівнів програми

Презентаційний рівень: презентаційний рівень обробляє HTTP-запити, перетворює параметр JSON в об’єкт, а також автентифікує запит і передає його на бізнес-рівень. Коротше кажучи, він складається з інтерфейсної частини.

Бізнес-рівень: бізнес-рівень обробляє всю бізнес-логіку. Він складається з класів обслуговування та використовує послуги, що надаються рівнями доступу до даних. Він також виконує авторизацію та перевірку.

Рівень збереження: рівень збереження містить усю логіку зберігання та перекладає бізнес-об’єкти з рядків бази даних у них.

Рівень бази даних: на рівні бази даних виконуються операції CRUD (створення, отримання, оновлення, видалення).

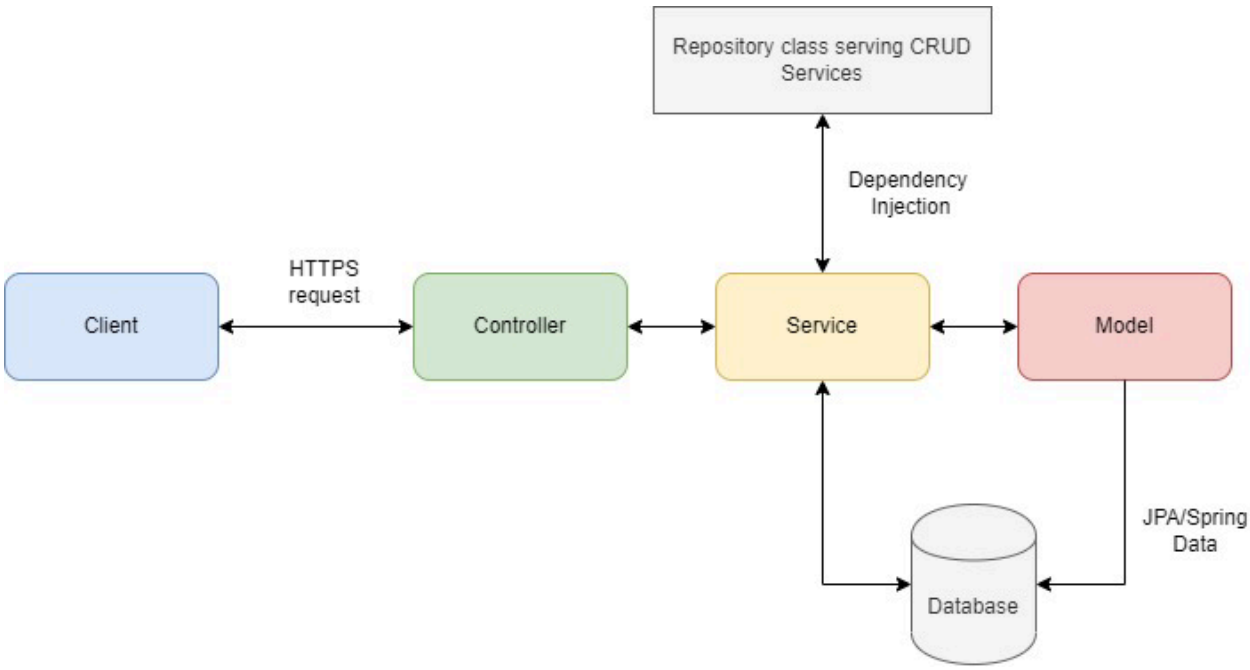


Рисунок 2.3 - Діаграма загальної архітектури системи

Модулі:

Структура модулів складається з трьох основних компонентів: контролер для програмного інтерфейсу, сервіс та репозиторій.

Ціль контролеру у даному випадку приймати запроси від клієнту, обробляти їх та перенаправляти до сервісу для подальшої обробки (як обробник запитів). Якщо описати більш конкретно, цей контролер приймає тільки rest запроси та валідує їх за стандартом, передбаченим логікою RestApi.

Рівень контролера. Перша частина системи, яка взаємодіє із запитом клієнта, — це контролери. Вони визначають кінцеві точки API, можна уявити собі кінцеві точки як дійсні маршрути та метод запиту (GET, POST, PUT). Головна мета контролера — пропонувати послуги клієнту, тобто надавати відповідь, статус тощо. Контролер використовує послуги, які надає рівень сервісу, щоб обслуговувати клієнта.

Що стосується сервісу, це основний шар розподілу обов’язків та зв’язків між компонентами, він приймає виклик від контролеру та відправляє отриманні дані до рівню бізнес-логіки, на якому і проводяться основні розрахунки. Сервісні рівні призначені для реалізації бізнес-логіки. Основна мета сервісного рівня – пропонувати послуги рівню контролера. На цьому рівні виконуються всілякі обчислення з даними,

тому сервісному рівню потрібні дані. Тому вони покладаються на послуги, які пропонує рівень DAO/Repository.

Репозиторій відповідає за взаємодію з базою даних. Він є абстракцією для розділення низькорівневих операцій від високорівневих розрахунків та компонентів програми.

Рівень сховища/DAO. DAO розшифровується як Data Access Object. Головною метою цього рівня є ефективний доступ (запит) до даних із бази даних і надання послуг сервісному рівню. У Spring Boot існують інтерфейси, які надають операції CRUD. Рівень сховища може реалізовувати ці операції. Варто зазначити, що різні сховища можуть надавати різні функціональності.

Модель. Модель представляє об'єкти реального світу. Тому ці об’єкти називають моделями. JPA (Java Persistence API) надає довідкові відомості або докладні відомості про ORM (Object–relational mapping), що означає, що клас Java можна пов’язати з таблицею бази даних. Існує багато реалізацій JPA ORM, однією з них є Hibernate. Тому потребується клас Java сутностей реального світу, а потім зіставити його з відношенням (таблиця).

Окрім розподілу компонентів за рівнем, фукціонально-спеціфичні компоненти були додатково розподілені у під-модулі за логічним призначенням у системі.

Так виглядає загальна структура модулів програми:

Компоненти програми розділені на основні рівні, крім того для додаткового налаштування безпеки та логування був доданий модуль конфігурації, який має в собі тематичні під-модулі. Модуль моделі також має в собі під-модулі за функціоналом, класи для специфічних калькуляцій отримали свій додатковий під-модуль, теж саме для всіх класів, які виконують формалізацію та пост-опрацювання розрахунків. Всі інші моделі є більш загальними, тому не потребують своїх модулів. Разом з цім у структуру включені тестові класи, тести мають свій незалежний модуль, але з метою простоти візуального представлення вони відображені разом із самим класами, тестування яких вони виконують. Також для гнучкої конфігурації самого додатку, разом з класами, у додатковому модулі є файли налаштування, вони регулюють поведінку самого додатку або його компонентів.

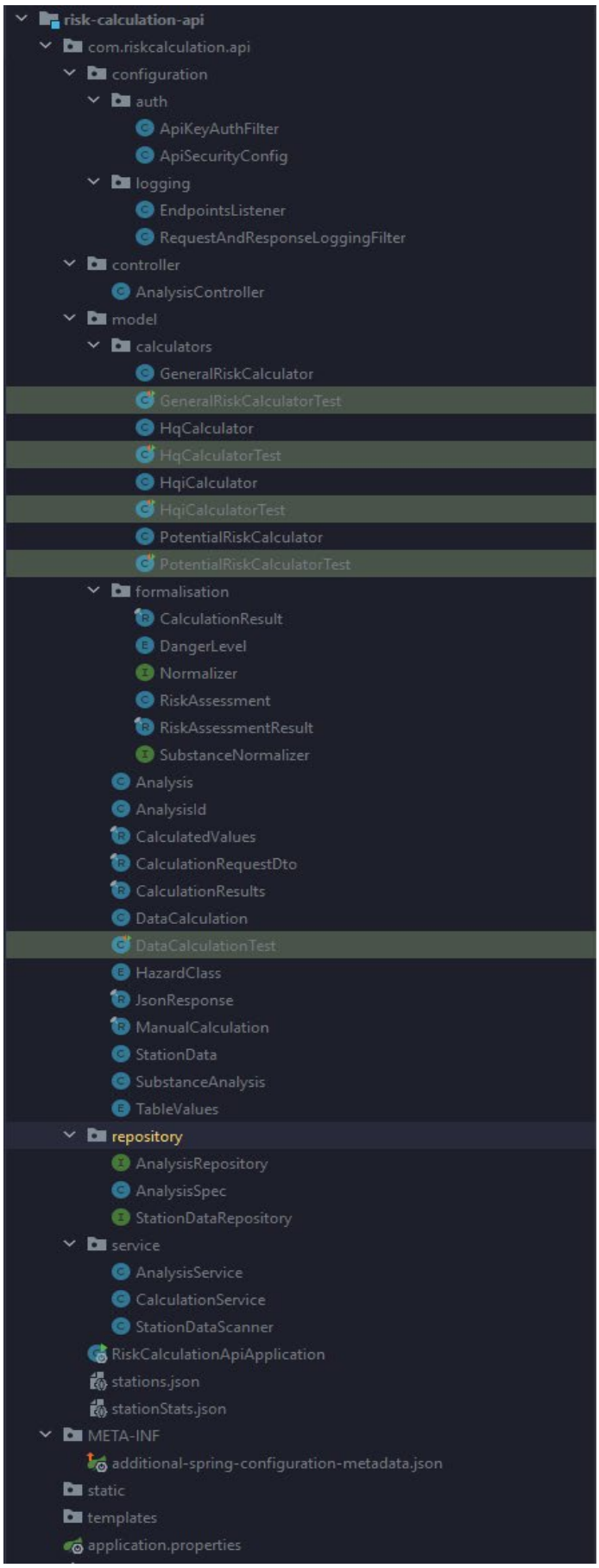


Рисунок 2.4 - Структура модулів програми

На цій діаграми представлені залежності головного класу додатку до інших виконавчих класів програми (рис. 2.5).

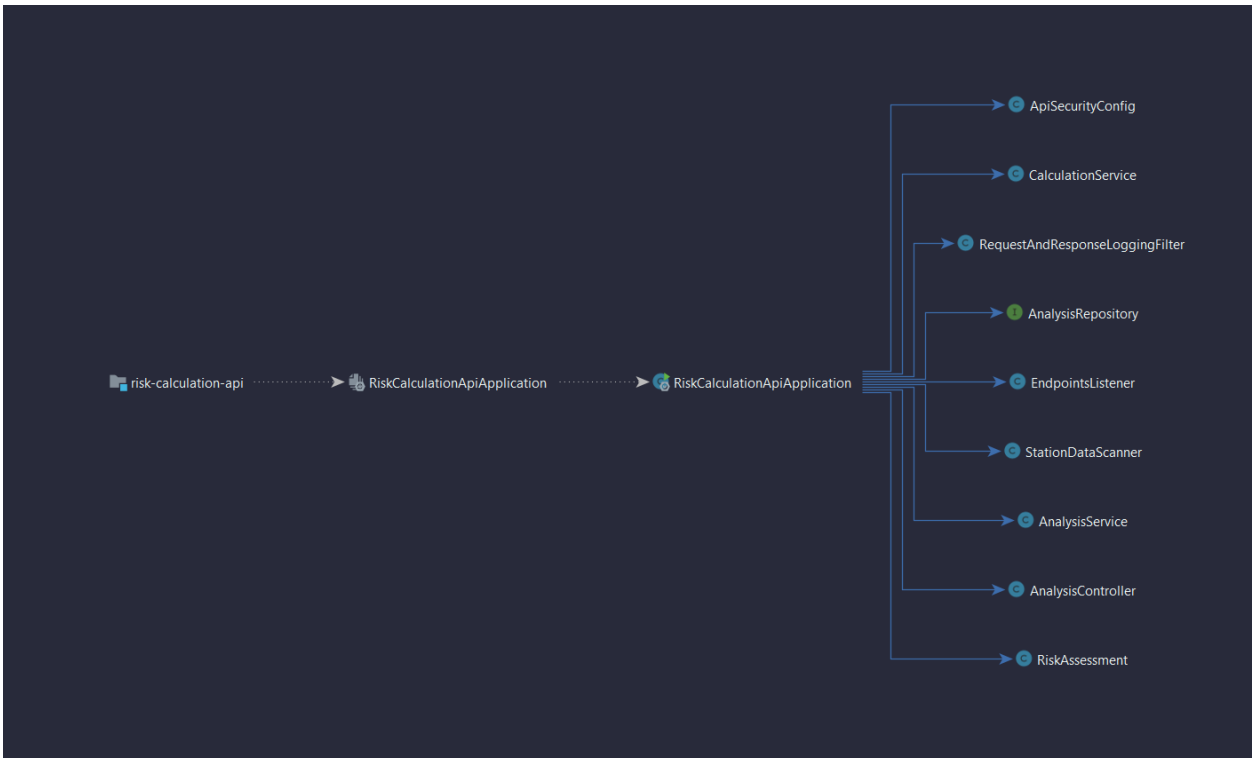


Рисунок 2.5 - Залежності моделей у spring.

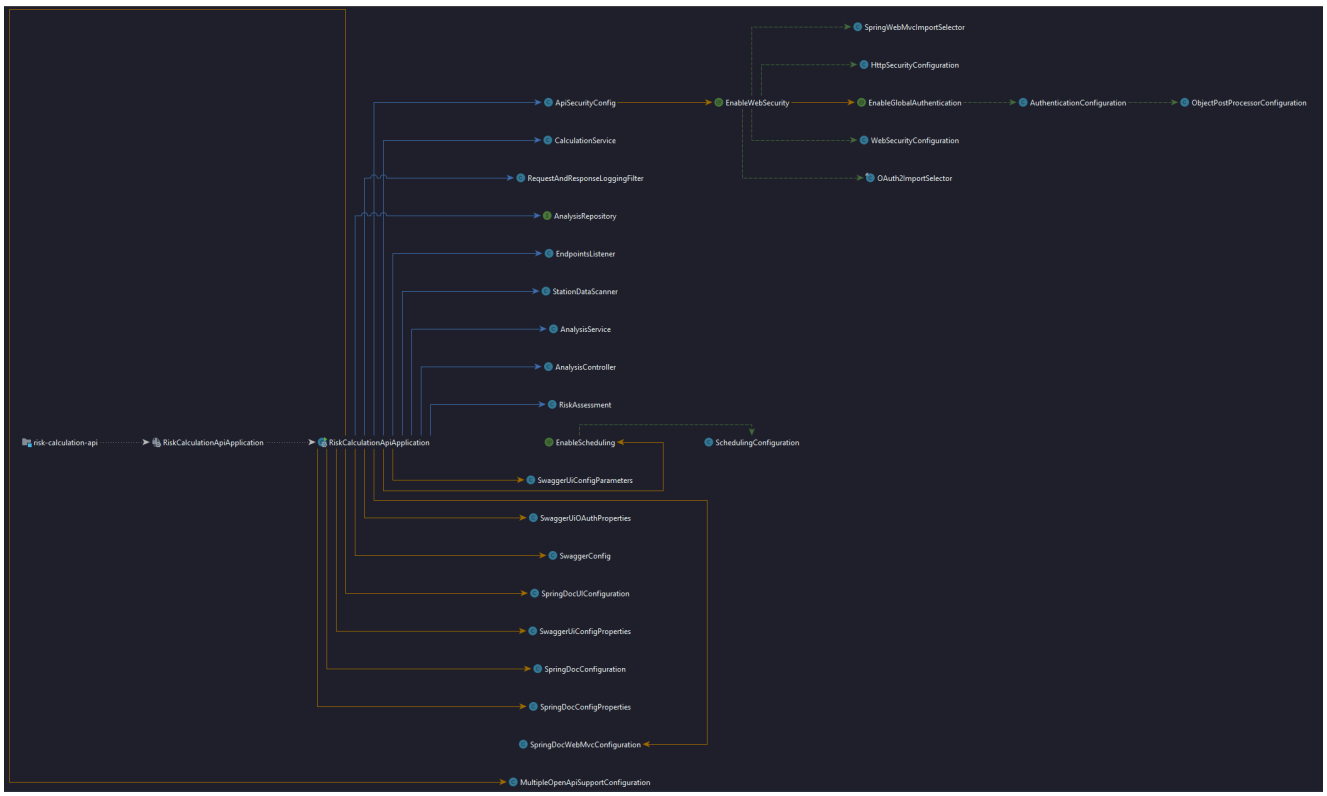


Рисунок 2.6 - Залежності та зв’язки моделей на рівні використаних компонентів бібліотек.

Поведінка системи. Джерелом даних для програми є веб-ресурс Eco city[1], програма обробляє конкретні дані виключно за допомогою інформації зі згаданого веб-ресурсу. Незалежно від запитів користувача, система робить запит до інформаційного ресурсу для збору даних кожні 20 хвилин для того щоб забезпечити актуальність даних. Коли користувач відправляє запит на автоматичний розрахунок даних за станцією, програма бере останній запис з бази даних зібраної інформації, проводить аналітику та повертає результат. Якщо користувач відправляє запит на мануальне розрахування, то програма проводить розрахунки “на льоту”, без зберігання даних, вона тільки обробляє подані концентрації хімічних речовин та проводить аналітику, повертаючи результат користувачу. І якщо користувач робить запит на відображення аналітики за часовою динамікою, програма вилучає всі розрахунки по запитаній станції, враховуючи поданий часовий проміжок, проводить аналітику та повертає результати.

Топологія системи (фізичне розміщення системи та її частин):

Spring Boot додаток знаходиться у власному контейнері, в якому він ізольований та може виконувати свою функціональність. Додаток може надає API та обробляє запити від клієнтів. Він також має доступ до інших контейнерізованих сервісів.

База даних, Grafana та Prometheus також розташовані у власних контейнерах. Spring Boot додаток може здійснювати звернення до бази даних для отримання та збереження даних. Grafana може використовуватися для візуалізації та моніторингу даних, а Prometheus - для збору та аналізу метрик системи.

Ця топологія забезпечує відокремленість та масштабованість окремих компонентів системи, спрощує керування та підтримку, а також дозволяє кожному компоненту працювати у своєму власному контейнері зі своїми власними залежностями.

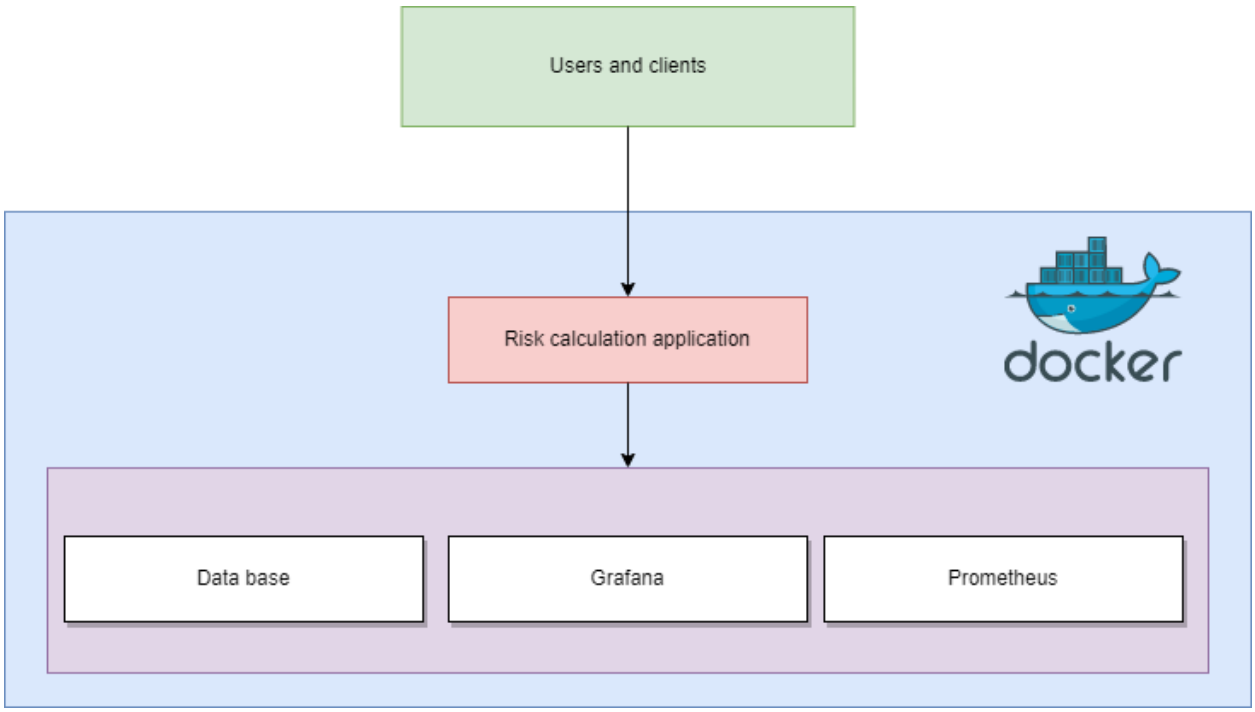


Рисунок 2.7 - Топологія системи

Структура баз та сховищ даних:

Всього програма має дві основні бази даних. Перша для зберігання розрахованих даних за станцією, то що це важливо для зберігання інформаційної історії. Друга база даних потрібна для зберігання зібраних даних, а конкретно показників якості повітря, зі зовнішнього ресурсу.

Особливості інтерфейсу користувача:

Робота користувача з API відрізняється від класичного візуального інтерфейсу, оскільки взаємодія з API відбувається через протоколи обміну даними, такі як HTTP або HTTPS, і зазвичай використовує формати даних, такі як JSON або XML.

Основні кроки взаємодії користувача з API включають:

1. Отримання доступу: Користувач повинен отримати необхідний доступ до API, такий як ключ API або токен автентифікації. Це може залежати від політики доступу, встановленої власником API.

2. Формулювання запиту: Користувач формулює запит до API, використовуючи певний HTTP метод, такий як GET, POST, PUT або DELETE. Запит може включати шлях до певного ресурсу, параметри, заголовки та тіло запиту з даними.

3. Відправлення запиту: Запит відправляється на сервер, який містить API, за допомогою використання URL-адреси API та відповідного HTTP методу. Запит може бути відправлений через командний рядок, спеціальні програми або з використанням різних програмних бібліотек.

4. Отримання відповіді: Сервер оброблює запит та надсилає відповідь у форматі, зазначеному в запиті. Відповідь може містити різну інформацію, таку як дані, статус виконання запиту, заголовки тощо.

5. Обробка відповіді: Користувач обробляє отриману відповідь, аналізує дані та при необхідності здійснює додаткові дії на основі результатів.

У порівнянні з класичним візуальним інтерфейсом, взаємодія з API не передбачає прямого візуального сприйняття інтерфейсу. Взамін, користувач має працювати з документацією API, де описані доступні шляхи, параметри та формати даних. Крім того, кваліфікація користувача передбачае розуміння HTTP-протоколу та форматів даних для ефективної взаємодії з API.

Також важливо зазначити, що використання API вимагає розробки програмного коду для здійснення запитів та обробки відповідей.

2.2.1 Проектування архітектури системи

Для спрощення візуального представлення, загальна діаграма класів показана без зав’язків між класами (рис. 2.8).

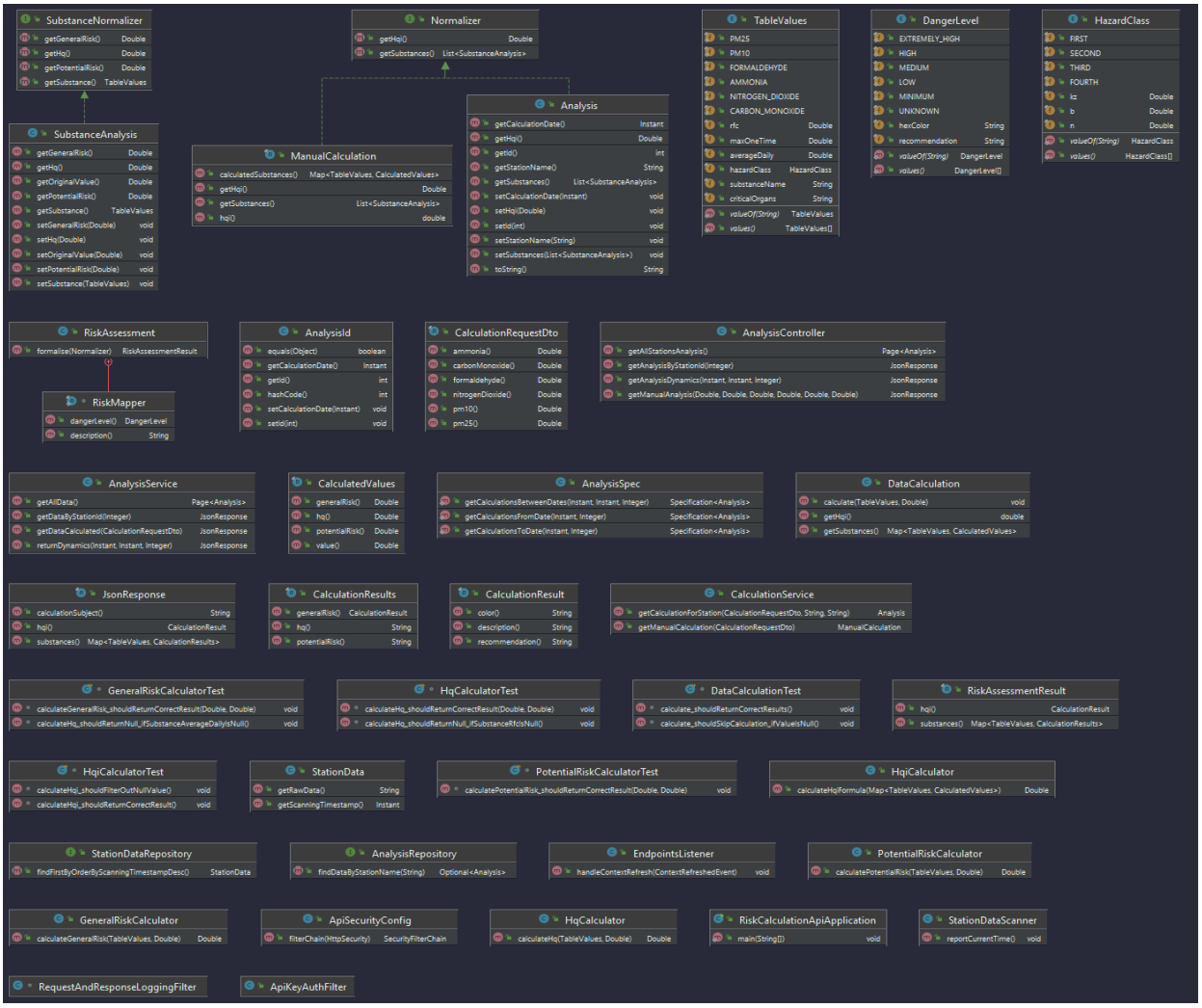


Рисунок 2.8 - Загальна діаграма класів

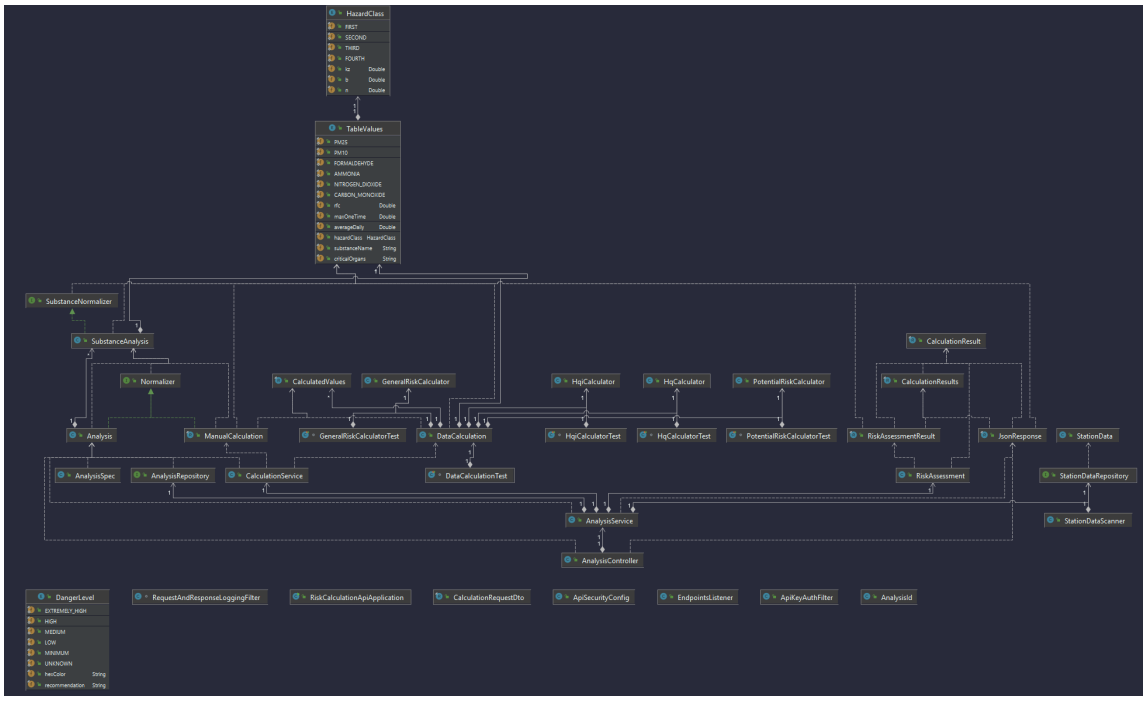


Рисунок 2.9 - Залежності між класами

Аналіз зовнішніх специфікацій системи:

Для цього був використаний OpenAPI Specification, бо він є потужним інструментом для аналізу зовнішніх специфікацій системи, який допомагає зрозуміти та перевірити функціональність та властивості API.

OpenAPI Specification (раніше відомий як Swagger) - це специфікація, яка визначає структуру та формат документації для RESTful API. OpenAPI Spec описує доступні ендпоінти, параметри, формати даних, ресурси та інші деталі API, що дозволяє забезпечити його інтеграцію та використання. OpenAPI Spec представляє собою машинночитаний файл у форматі YAML або JSON, який може бути використаний для автоматичної генерації документації, клієнтських бібліотек, серверних скелетонів та інших інструментів для роботи з API.

Основні елементи, які можуть бути визначені в OpenAPI Spec, включають:

1. Інформація про API: версія, опис, контактна інформація та інші метадані.

2. Ендпоінти та їх операції: опис доступних ендпоінтів API, методи HTTP (GET, POST, PUT, DELETE і т. д.), шляхи до ресурсів та параметри запитів.

3. Параметри: опис параметрів запиту, які можуть бути передані у URL, заголовках, запитах або тілі запиту.

4. Формати даних: опис форматів даних, які використовуються у відповідях та запитах, наприклад, JSON, XML і т. д.

5. Відповіді: опис можливих відповідей від сервера, статусні коди, заголовки та формати даних, які використовуються.

6. Безпека: визначення методів автентифікації та авторизації, які використовуються для захисту доступу до API.

OpenAPI Spec дозволяє розробникам та командам розробки програмного забезпечення легко спілкуватися, документувати та розуміти API. Він також дозволяє автоматизувати певні процеси, такі як генерація коду, тестування та валідація API, що сприяє прискоренню розробки та забезпеченню стабільності та сумісності API.

Всього в програмі представлено 4 кінцеві точки доступу.

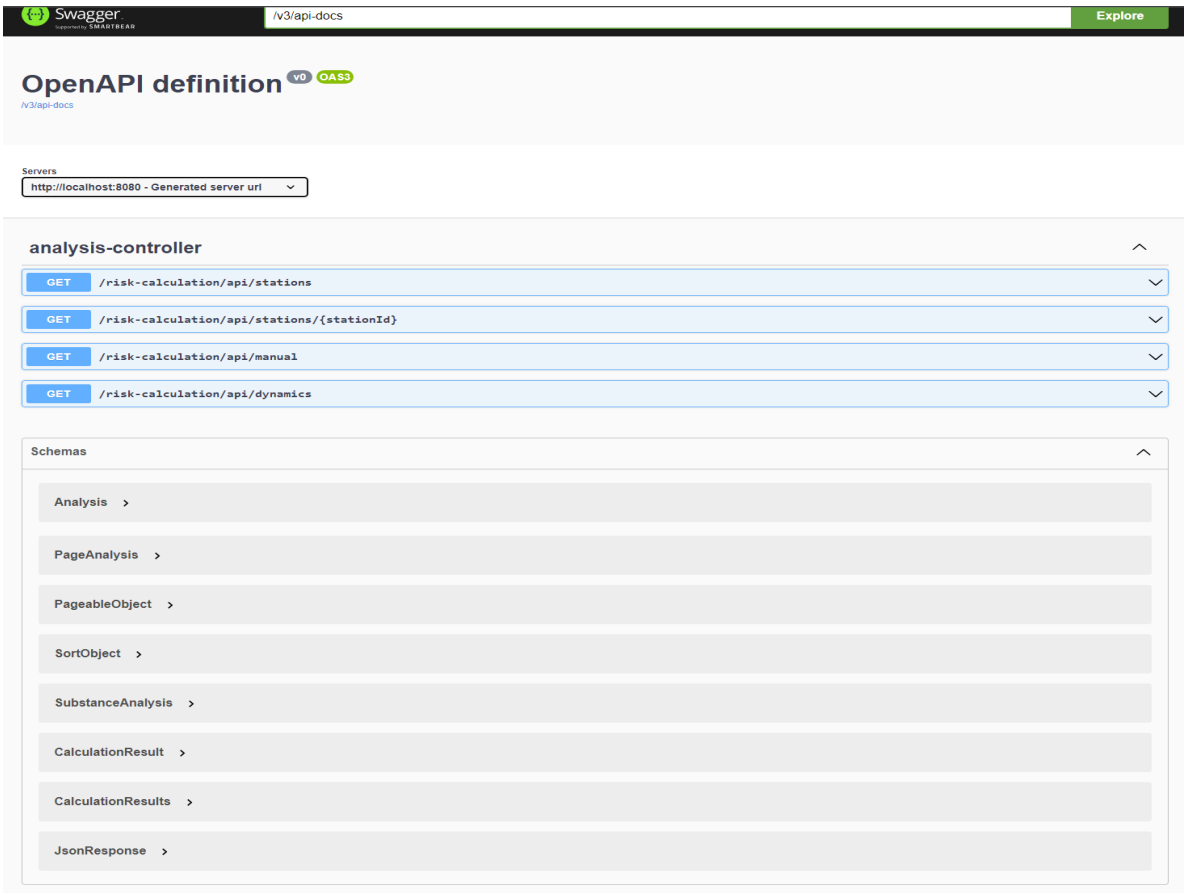


Рисунок 2.10 - Головна сторінка інтерфейсу Swagger

Перша точка доступу, має тип GET, функція цієї точки – отримання всієї інформації з бази даних розрахованих даних (рис 2.11).

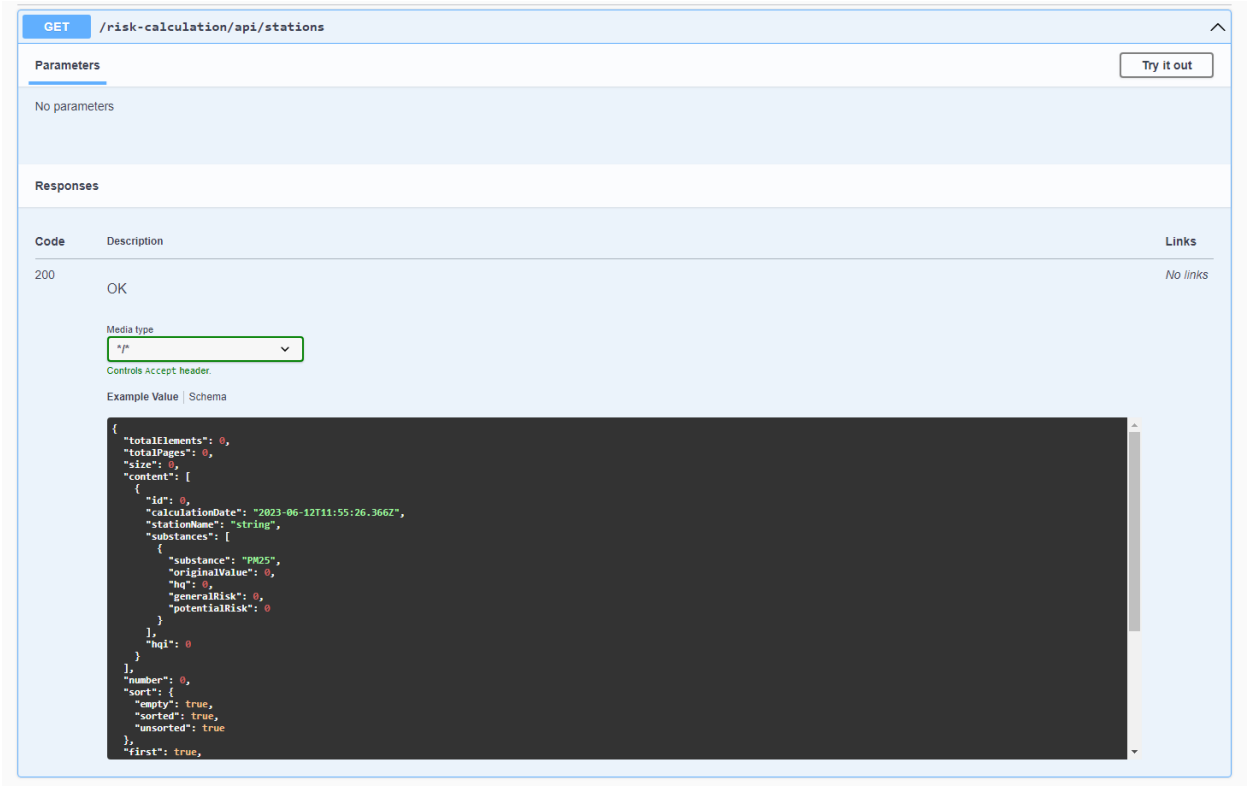


Рисунок 2.11 - Опис точки доступу для отримання розрахованих даних по всіх станціях

Друга точка доступу, тип GET, створена для ініціації автоматичної оцінки ризику на основі зібраних даних про станцію, користувач повинен надати id станції для розрахування (рис. 2.12).

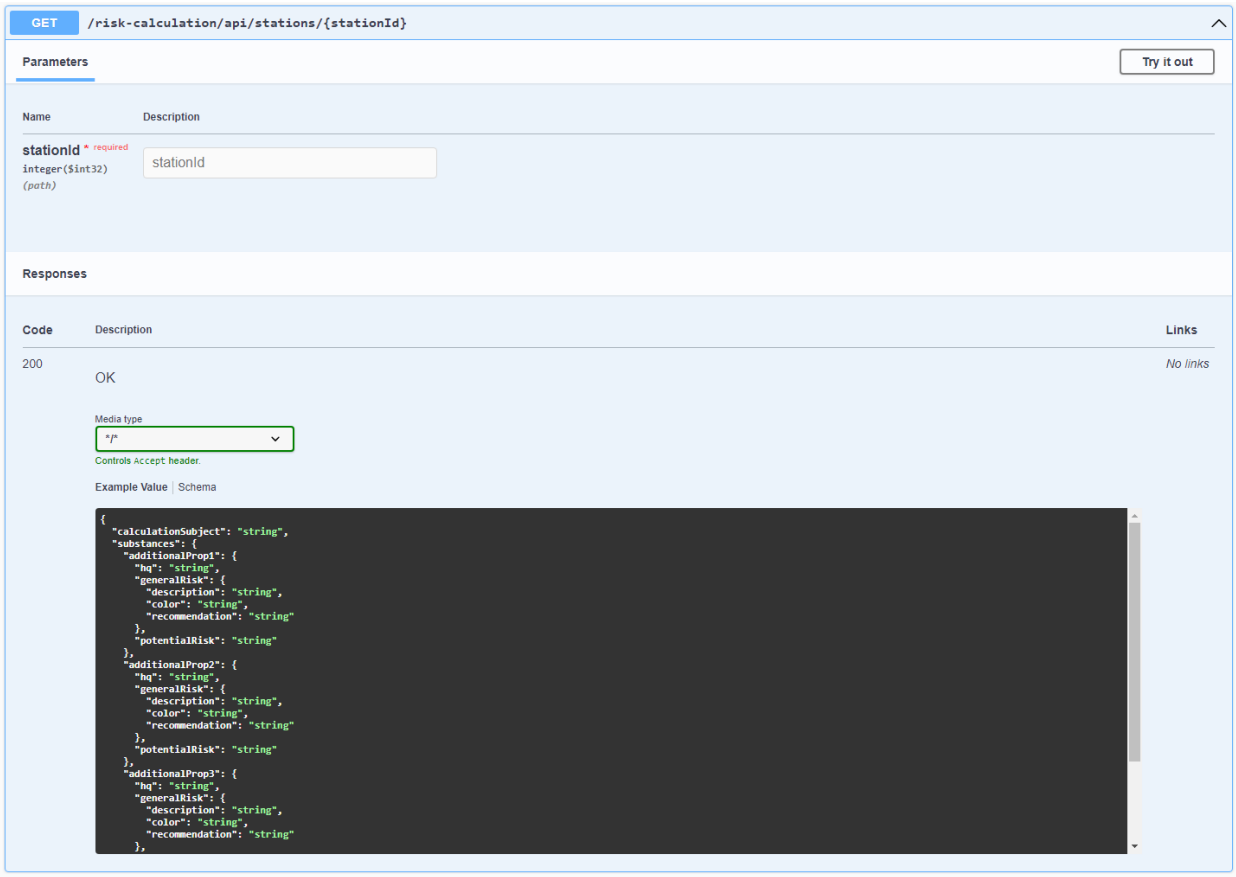


Рисунок 2.12 - Опис точки доступу для отримання автоматичного розрахунку ризиків за станцією

Третя точка доступу, тип GET, створена для розрахунку з мануальним введенням параметрів для розрахування. Тобто користувач повинен надати концентрації речовин, які підтримує програма, для отримання аналітики ризику за концентраціями (рис.2.13).

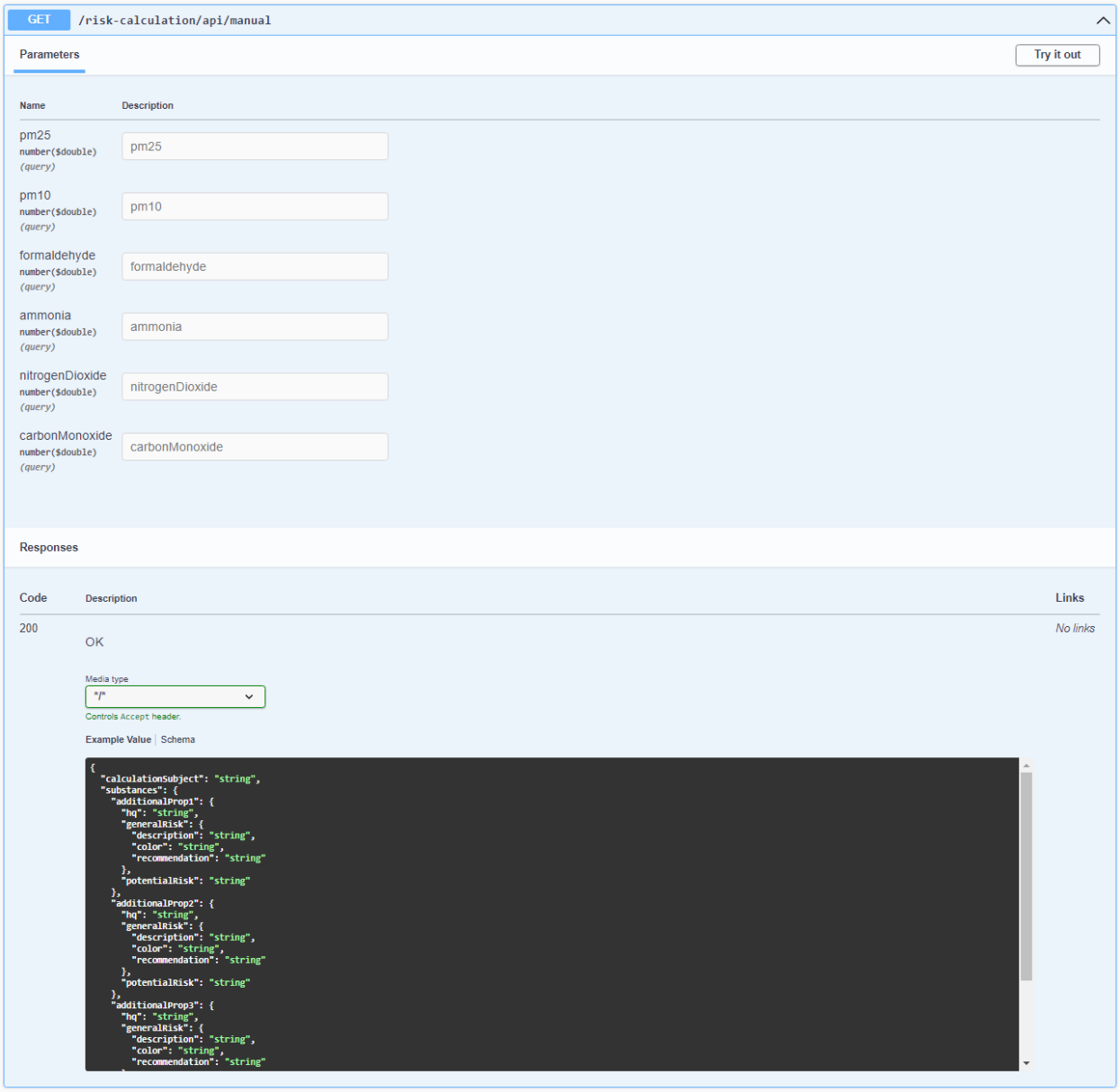


Рисунок 2.13 - Опис точки доступу для отримання ризиків за введеними концентраціями речовин

Четверта точка доступу, тип GET, створена для формування аналітики по станції з динамікою за деякий проміжок часу. Користувач повинен надати проміжок часу за яким повинна бути сформована динаміка показників, це повинен буди або проміжок часу, або починаючи з якої дати, або до якої дати, окрім інформації про час, також id станції є обов'язковим параметром (рис. 2.14).

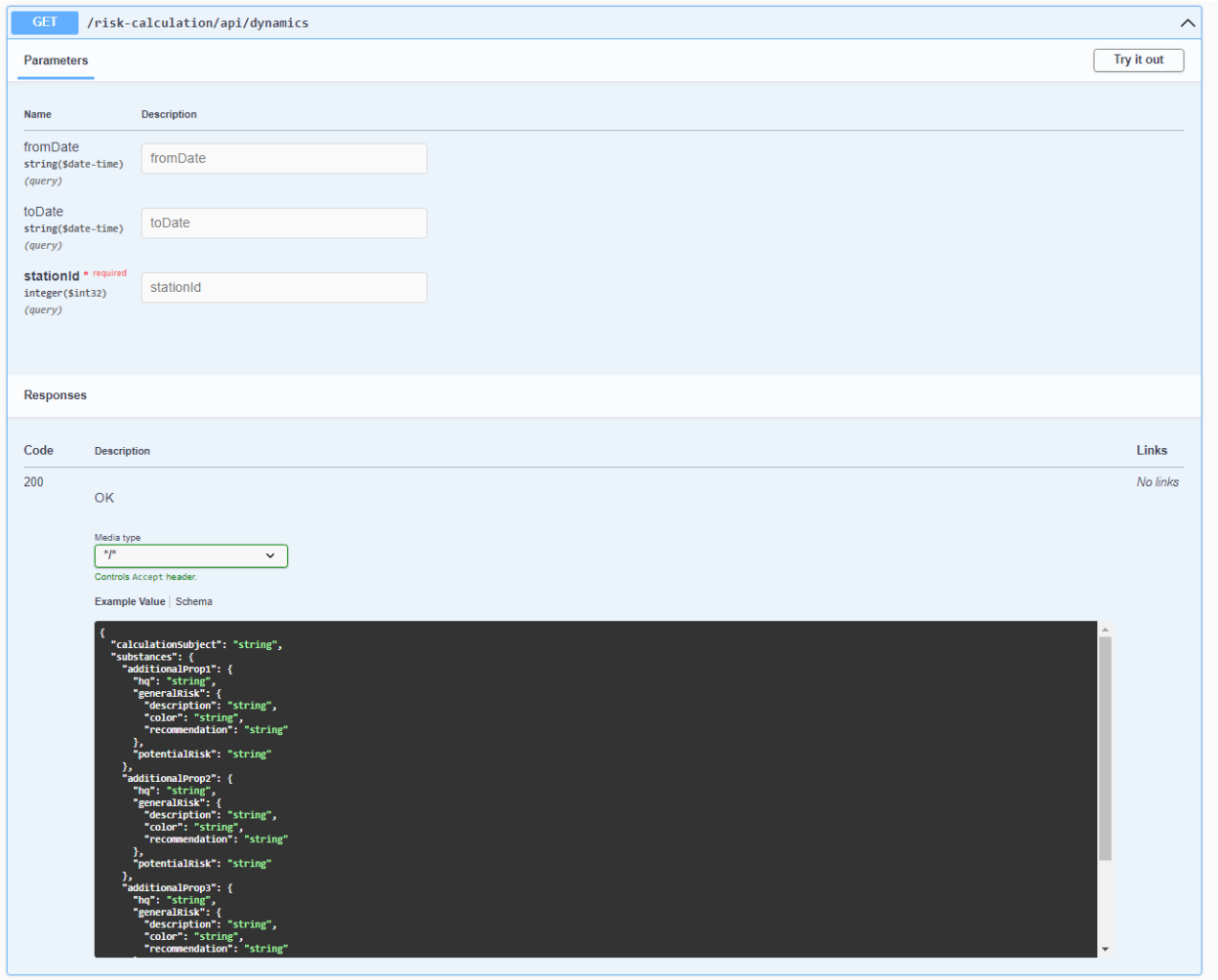


Рисунок 2.14 - Опис точки доступу для отримання динаміки показників за станцією

JSON файл з повною документацією структури кінцевих точок API, див. додаток Б.

2.2.2 Проектування інтерфейсу користувача

Через те, що це API, прямої потреби у інтерфейсі не має. Програма повністю виконує свої функції і без його участі. До того ж програма проектувалась як інтеграційний системний компонент і задачу візуалізації та транспортування даних, в цьому випадку, може виконувати зовнішній сервіс, розроблений спеціально для цієї мети.

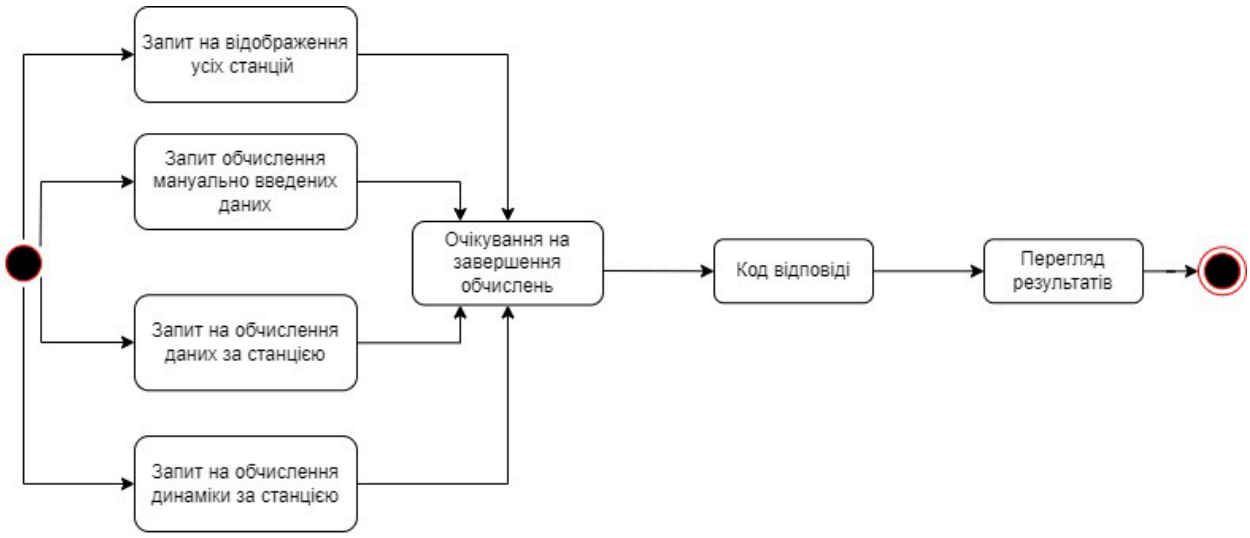


Рисунок 2.15 - Діаграма станів користувача

Подання та склад даних:

Програма повертає відповіді у форматі JSON (JavaScript Object Notation), бо він є зручним форматом даних для представлення та розуміння з точки зору користувача. Він має простий та лаконічний синтаксис, який легко читати та розуміти. JSON підтримується багатьма мовами програмування і має велику популярність у веб-розробці та обміні даними між системами. Він зручний для вкладення об'єктів та масивів, що дозволяє створювати складні структури даних. Крім того, JSON легко інтегрується з JavaScript, що полегшує роботу з даними у веб-додатках. Всі ці переваги роблять JSON популярним та зручним вибором для обміну та обробки даних.

Структура json відповіді:

Так виглядає шаблона структура відповіді, calculationSubject – заголовок файлу, в основному містить загальну інформацію про виконаний розрахунок. Substances – масив хімічних речовин за якими був виконаний розрахунок, в даному випадку additionalProp1, additionalProp2, additionalProp3 – описують собою назви речовин. Кожна речовина має вкладену структуру, туди входить три основних показники – hq, hqi, generalRisk, potentialRisk – це контейнери для групування та відображення аналітики. Hqi та generalRisk мають додаткові властивості, description – це опис типу ризику, color – колір ризику у таблиці кольоровій ідентифікації та recommendation – представляє рекомендовані заходи для запобігання небезпеки для

здоров'я.

Приклад шаблону json відповіді:


```
{
 "calculationSubject": "string",
 "substances": {
  "additionalProp1": {
   "hq": "string",
   "generalRisk": {
    "description": "string",
    "color": "string",
    "recommendation": "string"
   },
   "potentialRisk": "string"
  },
  "additionalProp2": {
   "hq": "string",
   "generalRisk": {
    "description": "string",
    "color": "string",
    "recommendation": "string"
   },
   "potentialRisk": "string"
  },
  "additionalProp3": {
   "hq": "string",
   "generalRisk": {
    "description": "string",
    "color": "string",
    "recommendation": "string"
   },
```

```
      "potentialRisk": "string"
    }
  },
  "hqi": {
    "description": "string",
    "color": "string",
    "recommendation": "string"
  }
}
```


2.2.3 Вибір парадигми проектування

Використання парадигми об'єктно-орієнтованого програмування (ООП), особливо для мови Java, має безліч переваг, і ось декілька ключових аргументів, що підкреслюють, чому це правильний вибір:

1. Модульність та повторне використання коду: ООП дозволяє розділити програму на невеликі модулі, які називаються класами. Класи містять як дані, так і функції, які з ними пов'язані. Це забезпечує модульність, тобто можливість працювати з окремими частинами програми незалежно від інших. Крім того, класи можуть бути використані повторно в різних частинах програми, що сприяє економії часу та зусиль у розробці.

2. Спадкування та поліморфізм: ООП надає можливість створювати ієрархії класів, де класи можуть успадковувати властивості та методи від інших класів. Це спрощує структуру програми та дозволяє створювати більш абстрактні та зрозумілі моделі. Крім того, поліморфізм дозволяє об'єктам одного класу використовуватись як об'єкти іншого класу, що покращує гнучкість та розширюваність коду.

3. Інкапсуляція та захист даних: ООП сприяє інкапсуляції, тобто затриманню даних та методів, що з ними пов'язані, разом у межах класу. Це дозволяє контролювати доступ до даних та методів, забезпечує їх правильне використання та захищає від несанкціонованого доступу. Інкапсуляція сприяє покращенню безпеки та стабільності програми.

4. Розширюваність та підтримка: Java, яка базується на парадигмі ООП, має широку підтримку та велику спільноту розробників. Це означає, що існує багато стандартних бібліотек, інструментів та ресурсів, які полегшують розробку та підтримку програмного забезпечення. Також, ООП сприяє розширюваності програми, оскільки нові класи можуть бути додані без зміни вже існуючого коду.

5. Простота та зрозумілість: ООП надає можливість моделювати реальні об'єкти та взаємозв'язки між ними у вигляді класів, що дозволяє створювати програми, які легко зрозуміти. Код, написаний у відповідності до принципів ООП, має чітку структуру та може бути більш зрозумілим для інших розробників.

Ця парадигма підходить для розробки програмного комплексу з використанням мови програмування Java. Вона сприяє полегшенню розробки, підтримці та вдосконаленню програмного забезпечення, що робить її правильним вибором для багатьох проектів. У випадку з розробкою структури API та його функціоналу, об'єктно-орієнтована парадигма забезпечує зручну опцію, оскільки дозволяє розділяти класи та функціонал, структурувати внутрішні компоненти та побудову загальної логіки програмного інтерфейсу. Це спрощує розподіл відповідальностей між компонентами, специфікацію функціоналу та забезпечує простоту розуміння моделі. Крім того, така парадигма має чіткі рамки задач для класів, функціонал, специфікований конкретним use case, та зручний менеджмент залежностей між модулями. Також до цього слід брати до уваги майбутнє проекту, в такій ситуації вибір лежіть між ООП та функціональною парадигмою, функціональна парадигма має перевагу у випадку коли програма має набір структур та у перспективі вже присутні структури будуть покращені та розвинуті, а у випадку з ООП парадигмою, до вже присутніх структур будуть додані нові. І цей проект має на меті розвинення функціоналу з доданням нових функцій через додавання нових структур.

2.2.4 Проектування динаміки системи

Через те, що загальний процес має багато кроків, діаграма послідовностей було спрощена та доповнена діаграмами викликів для кожного методу контролеру та структурною діаграмою системи.

Актор відправляє один з чотирьох запитів, у кожного запиту є обробляючий

запит у сервісі. Запит на отримання всіх даних отримує дані та одразу повертає їх. Якщо це запити, які потребують розрахунків, то дані перенаправляються до сервісу розрахування, який далі перенаправить їх до відповідних моделей-калькуляторів. Після розрахування вони будуть перенаправленні до сервісу формалізації. Запит на отримання динаміки, не потребує розрахунків, тому дані відправляються до сервісу формалізації напряму. І на основі формалізованого результату буде повернута відповідь до контролеру.

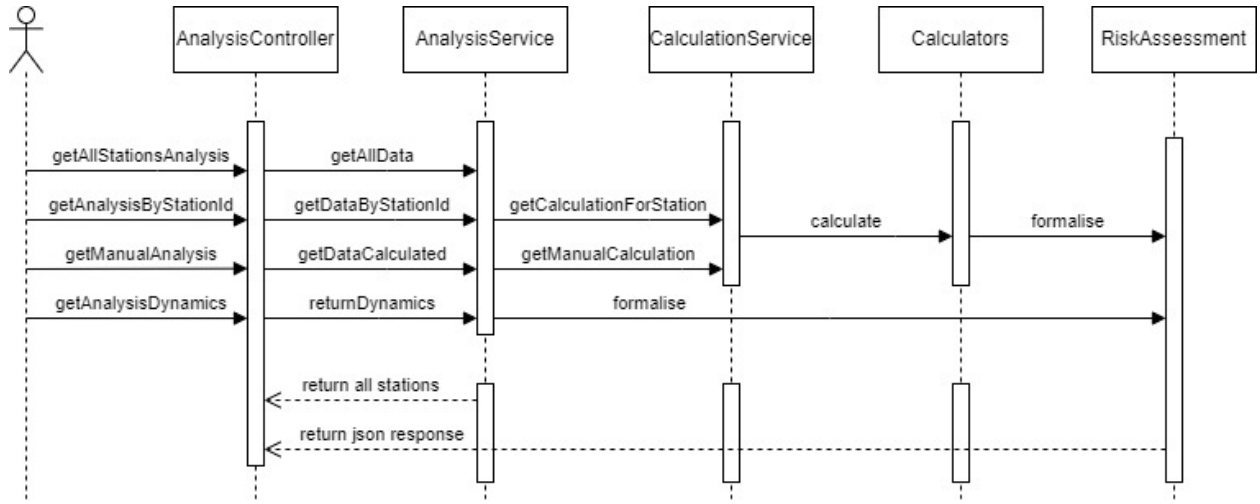


Рисунок 2.16 - Загальна діаграма послідовностей з точки зору контролеру

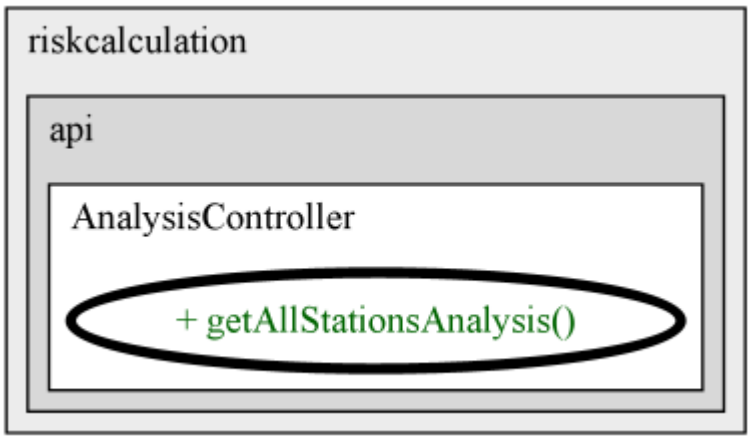


Рисунок 2.17 - Діаграма викликів для методу getAllStationsAnalysis

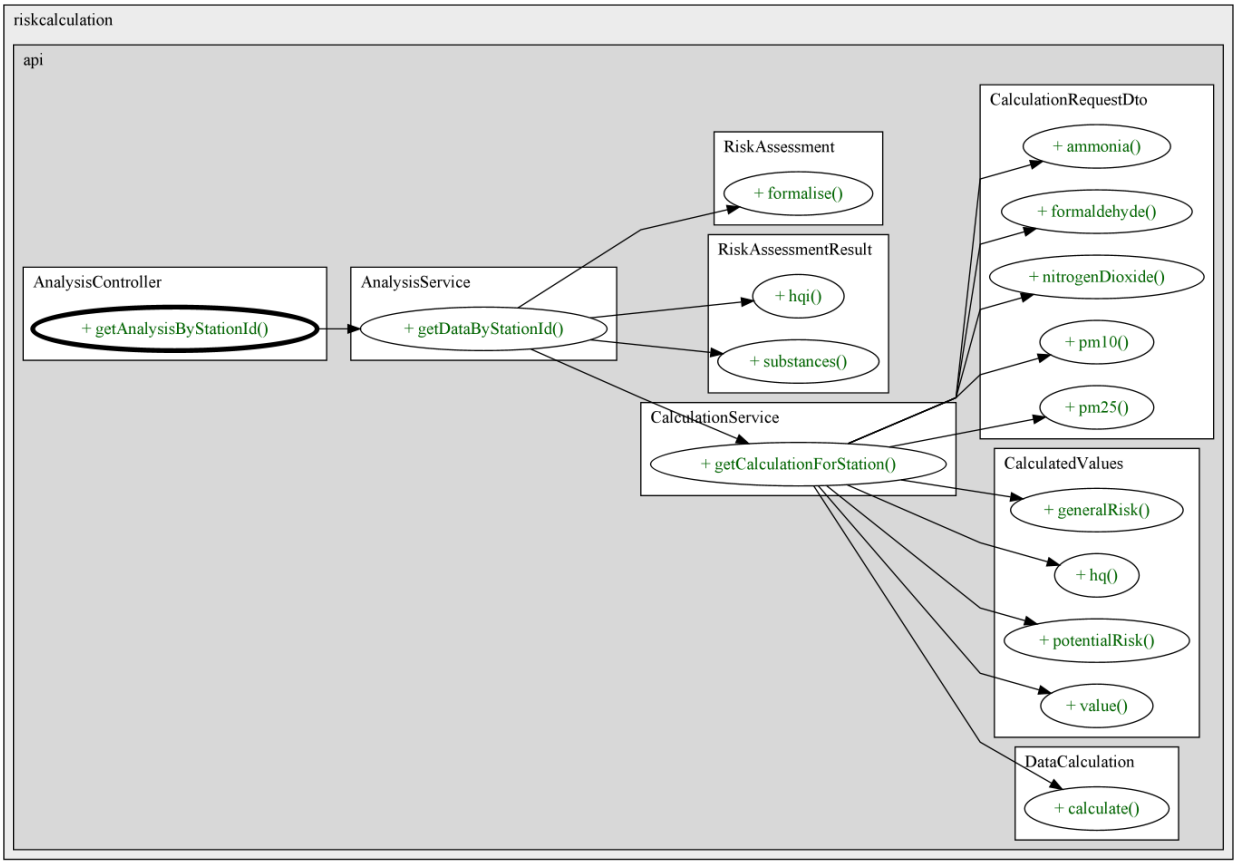


Рисунок 2.18 - Діаграма викликів для методу getAnalysisByStationId

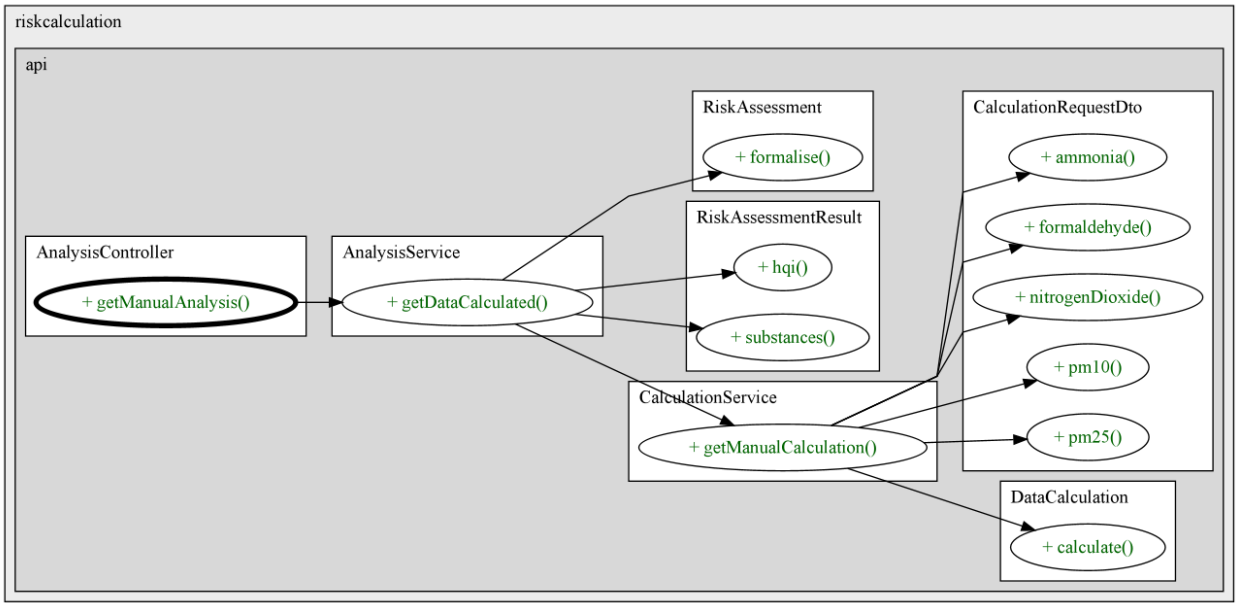


Рисунок 2.18 - Діаграма викликів для методу getManualAnalysis

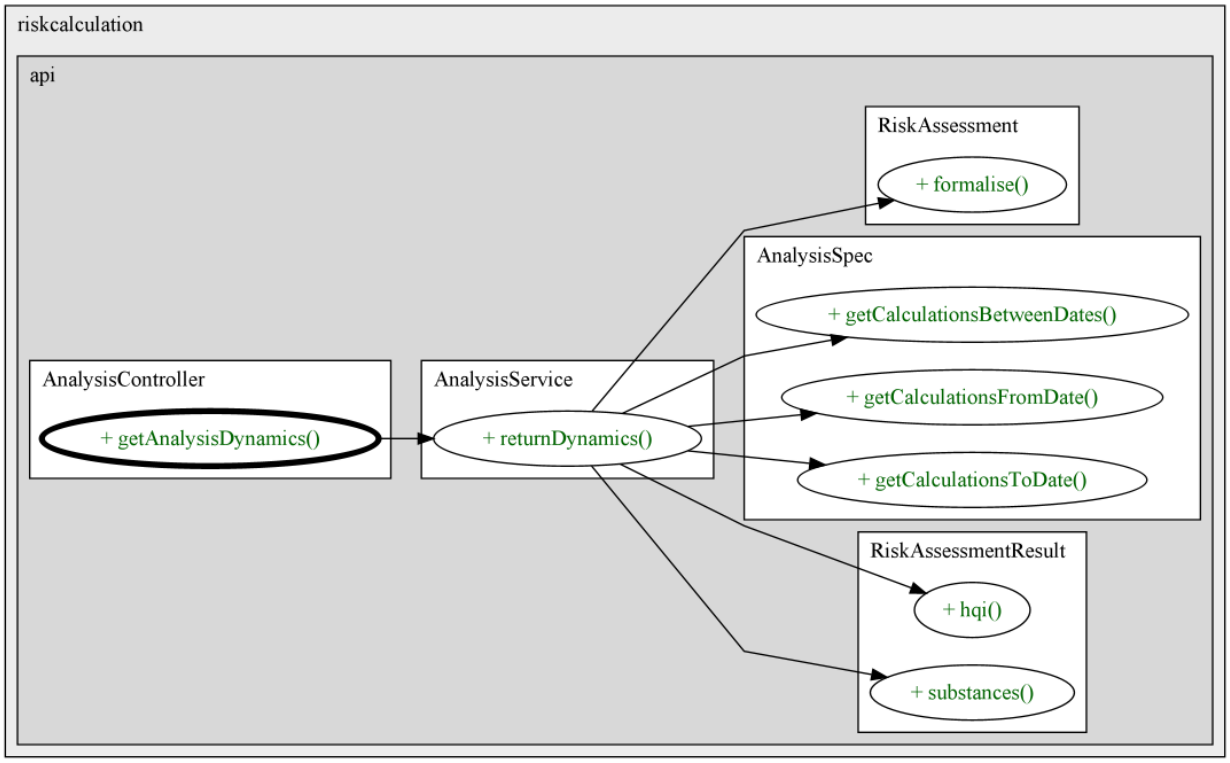


Рисунок 2.19 - Діаграма викликів для методу getAnalysisDynamics

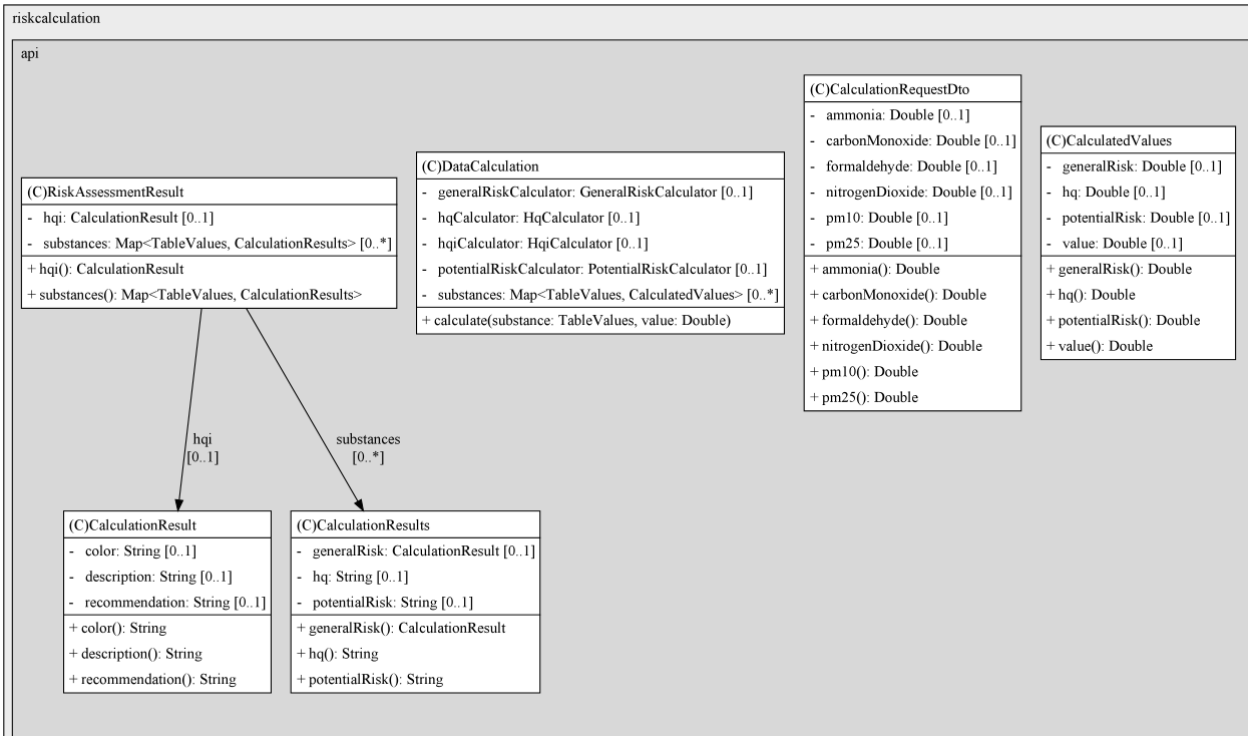


Рисунок 2.20 - Структурна діаграма з точки зору контролеру частина 1

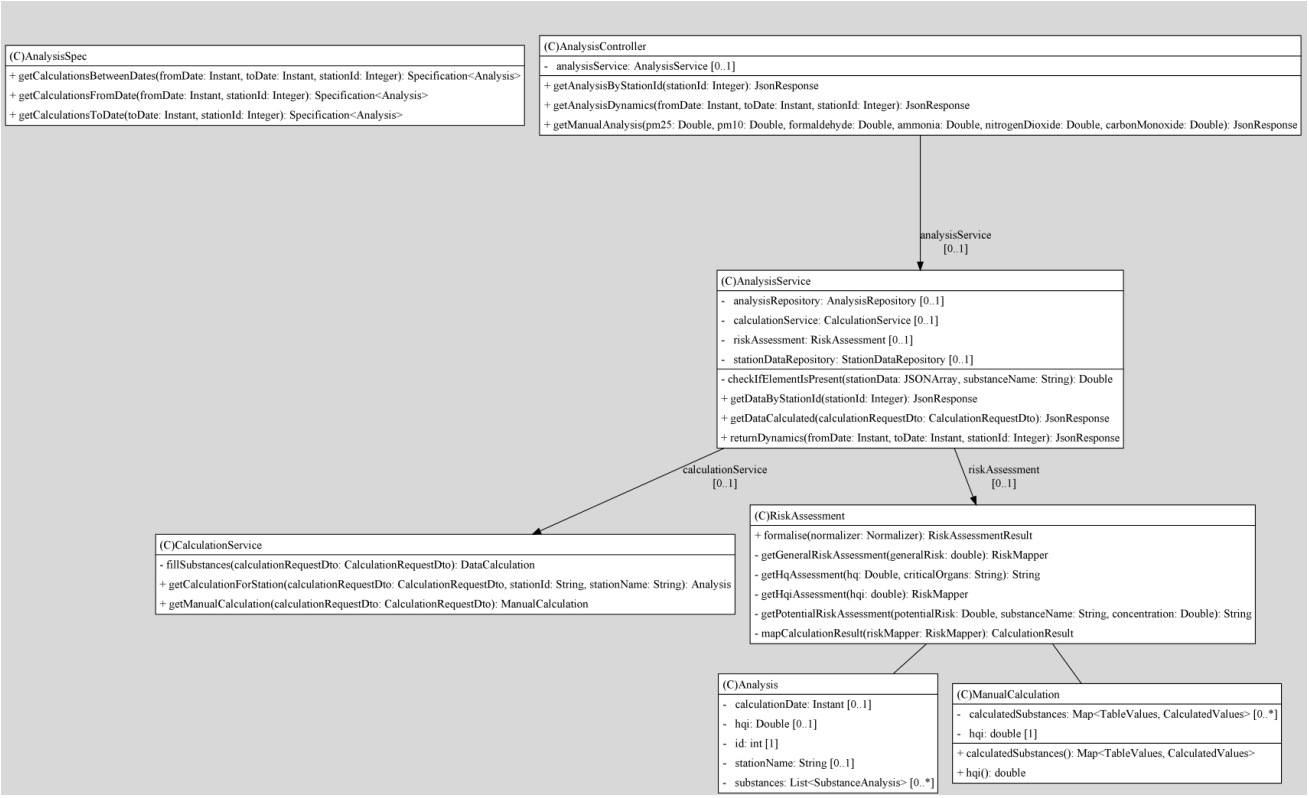


Рисунок 2.21 - Структурна діаграма з точки зору контролеру частина 2

Стосовно динаміки системи, така ж важливо показати процес збору даних (рис. 2.22).

Спочатку проводиться відправка запиту до ресурсу за токеном, якщо інформація присутня, інформація зберігається, якщо ні, програма очікує фіксований проміжок часу та повторює запит. Після збереження інформації програма повертається до стану очікування та теж потроює запит через фіксований проміжок часу.

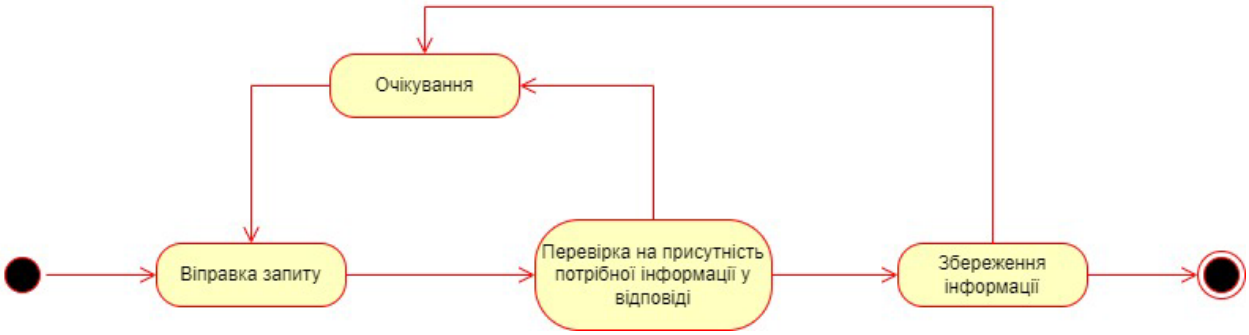


Рисунок 2.22 - Діаграма станів запиту до зовнішнього ресурсу сбіру інформації.

2.2.5 Проектування системи на фізичному рівні

Абстрактне представлення систем за модулями та функціональними зв'язками (рис. 2.23). Контролер викликає сервіс, який в свою чергу займається розподілом задач для калькулятора, нормалізатору та репозиторію в залежності від конкретного запиту до контролеру.

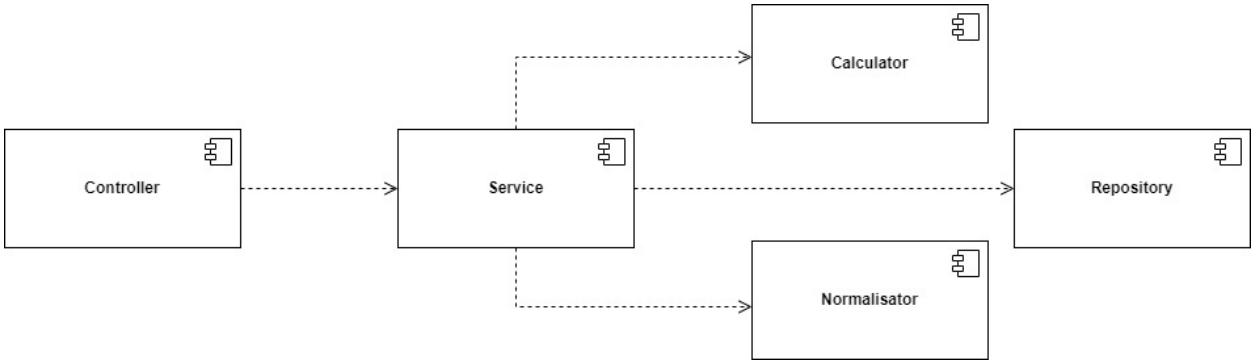


Рисунок 2.23 - Абстрагована діаграма артефактів

2.2.6 Проектування бази даних

Для збереження усіх даних була обрана реляційна база даних MySql. Тому що вона достатньо проста у використанні та забезпечує весь потрібний функціонал для вирішення задач, які ставить пере собою цей проект.

База даних має всього три таблиці. Основна таблиця для зберігання розрахованих даних, вона прив'язана до станції сканування, для збереження інформації про речовини для кожної станції була створена додаткова таблиця яка через ідентифікатори зв'язана з таблицею розрахунків по станціях. Та третя таблиця створена для збереження історичних даних, які були зібрані з API постачальника даних, основною особливістю цієї таблиці є те що вона зберігає отриманий json у базу даних та індексує дані за датою сканування.

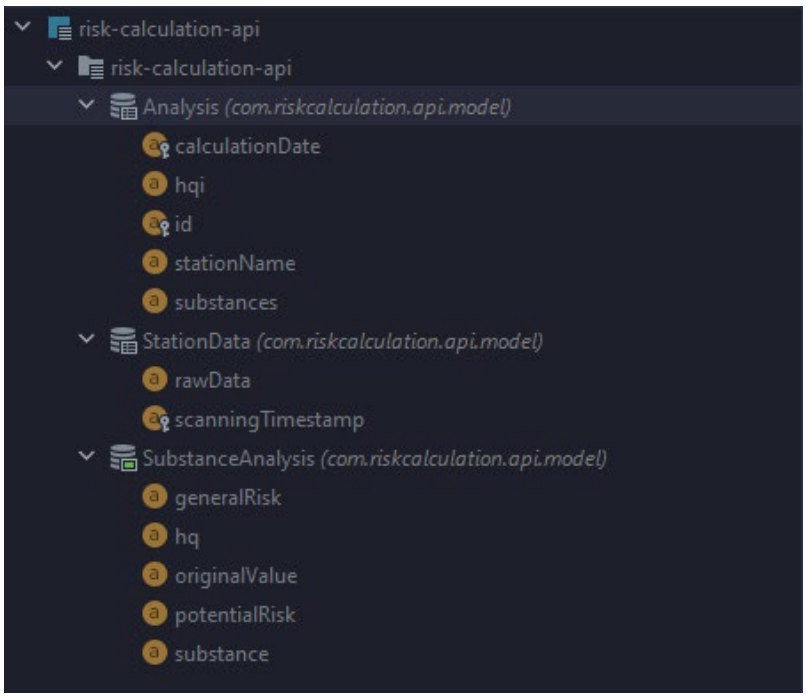


Рисунок 2.24 - Збереження даних в базі даних в рамках моделей програми

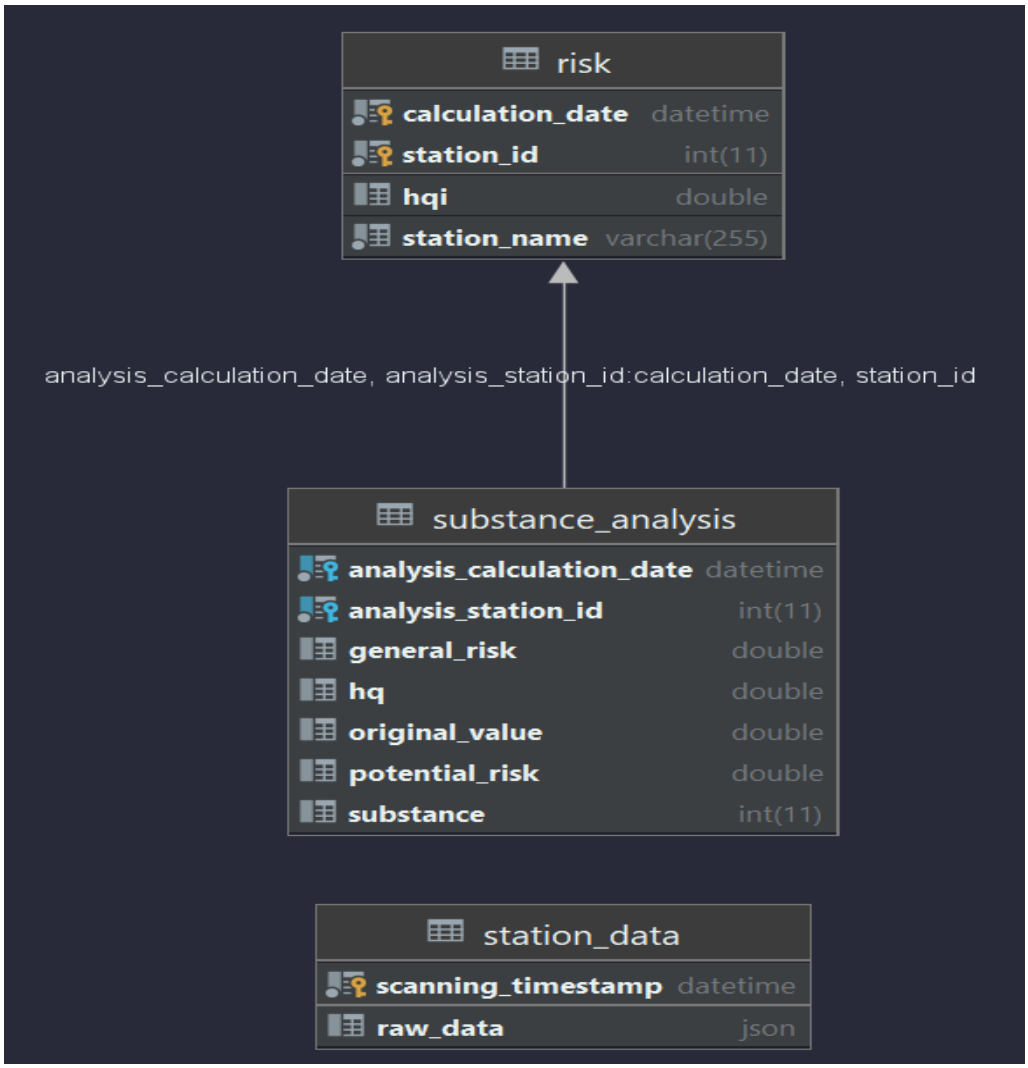


Рисунок 2.25 - Логічна схема бази даних

Висновки до розділу 2

Під час етапу проектування були визначені потреби до функціональності програми, проведений огляд процесів та поведінки системи. Сформована архітектура системи та її компонентів, логіка побудови модулів, документування залежностей та функціоналу додатку. Крім того, проведене проектування бази даних та системи на фізичному рівні. Враховуючи всі ці аспекти, етап проектування є критичним для успішної реалізації програмного продукту, оскільки він визначає фундаментальні аспекти системи та забезпечує чітке розуміння вимог та задач, що повинні бути виконані під час розробки.

3 РОЗРОБКА

3.1.1. Алгоритм, його властивості, засоби описування

Основна ідея алгоритму проста як і завжди:

1. Отримати початкові дані;
2. Перетворити початкові дані в результати (вихідні дані);
3. Повернути результати.

У першому пункті мається на увазі отримання початкових даних для ініціації роботи алгоритму, вже підготовлені та структуровані данні постачаються до алгоритму з бази даних через логічний фасад для проведення операцій з базою даних, в цьому випадку – пошук.

В другому пункті, дані типізуються за їх початковим значенням, та зіставляються з їх табличним еквівалентом. Опис формул див. Додаток В. На приклад, для першого етапу обчислення, концентрація формальдегіду у повітрі буде зіставлена з його табличним еквівалентом, далі проводяться обчислення за формулою HQi для кожної речовини. Далі HQi підсумовуються між собою для отримання загального HQ. Окрім того, проводиться обчислення загального ризику та потенціального. Для обчислення обох ризиків потребується концентрація речовини та її клас небезпеки, від класу небезпеки залежить табличний еквівалент додаткових змінних для обчислення за формулою. Далі проводяться відповідні обчислення ризиків за формулами. Після чого обчислення завершено.

І на третьому етапі, алгоритм обчислення повертає результати, які пізніше будуть збережені у бд.

Характеристики алгоритму:

Прояв детермінованості полягає в тому, що результат обчислень завжди однозначний, це hq показники за концентраціями отриманих хімічних речовин та два обчислених ризики: потенціальний та загальний.

Алгоритм розділений на послідовні кроки: отримання речовин, розрахунок HQi для кожної, розрахунок загального HQ та розрахунок ризиків. Тому цей алгоритм е дискретним.

До того ж алгоритм є зрозумілим, тому що слідує логічній послідовності формул.

Розв'язок здобувається через 4 кроків, що каже про результативність алгоритму.

Алгоритм гнучкий, незалежний та здатен вирішити усі види задач даного типу. Таким чином, цей алгоритм можна характеризувати як масовий.

Алгоритм представлений на алгоритмічній мові програмування Algol68:

Функція обчислення загального ризику:

```
PROC generalRisk = (INT arrayIndex, INT hazardClass, REAL tableValue)VOID:
BEGIN
IF substances.get(arrayIndex) # 0 THEN
[REAL] kz, b;
IF hazardClass = 1 THEN
kz := 7.5;
b := 2.35;
ELIF hazardClass = 2 THEN
kz := 6.0;
b := 1.28;
ELIF hazardClass = 3 THEN
kz := 4.5;
b := 1.0;
ELIF hazardClass = 4 THEN
kz := 3.0;
b := 0.87;
ELSE
kz := 0;
b := 0;
FI
   generalRiskArray.add(1 - exp(log(0.84) * (substances.get(arrayIndex) / tableValue) ** b / kz));
```

```
ELSE
   generalRiskArray.add(NULL);
FI

END;
```

Функція обчислення потенційного ризику:

```
PROC potentialRisk = (INT arrayIndex, INT hazardClass, REAL tableValue)VOID:
BEGIN
IF substances.get(arrayIndex) # 0 THEN
[REAL] kz, n;
IF hazardClass = 1 THEN
kz := 7.5;
n := 2.4;
ELIF hazardClass = 2 THEN
kz := 6.0;
n := 1.31;
ELIF hazardClass = 3 THEN
kz := 4.5;
n := 1.0;
ELIF hazardClass = 4 THEN
kz := 3.0;
n := 0.86;
ELSE
kz := 0;
n := 0;
FI
   potentialRiskArray.add(1 - exp(pow(abs(log(0.84) / (tableValue * kz)) *
substances.get(arrayIndex) * 70, n)) * (-1));
ELSE
```

```
    potentialRiskArray.add(NULL);
FI


END;
```

Функція обчислення HQi для кожної речовини:

```
PROC hqFormula = (INT arrayIndex, REAL tableValue)VOID:
BEGIN
IF substances.get(arrayIndex) # 0 THEN
substances.set(arrayIndex, substances.get(arrayIndex) / tableValue);
FI
END;
```

3.1.2. Графічні засоби описування алгоритмічних структур

Загальне представлення алгоритму.

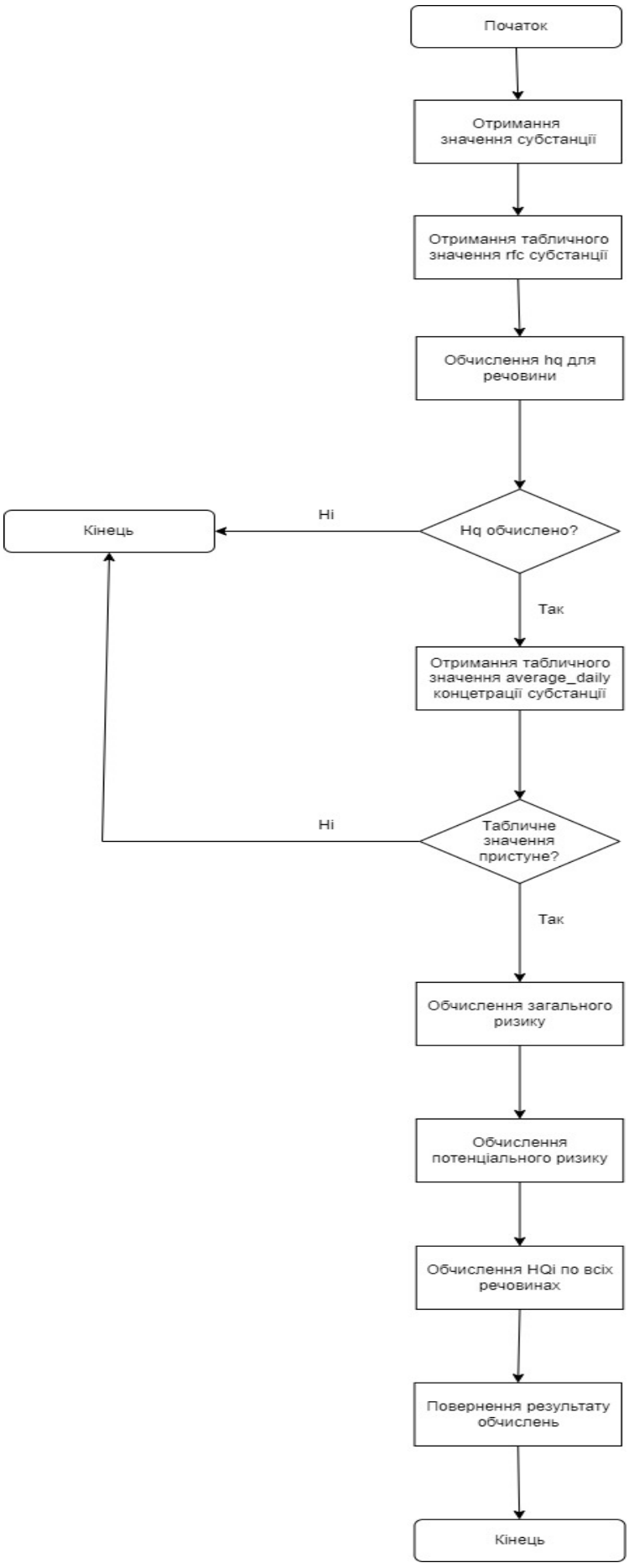


Рисунок 3.1 - Блок-схема алгоритму обчислення ризиків

3.1.3. Види алгоритмічних структур

В основному можна описати цей алгоритм, як послідовний, з елементами розгалуження.

Послідовний алгоритм з елементами розгалуження є одним з основних типів алгоритмів, який використовується в програмуванні. Він дозволяє виконувати послідовні дії з можливістю вибору різних шляхів в залежності від умов.

Основна особливість послідовного алгоритму з елементами розгалуження полягає в тому, що він дозволяє виконувати певні дії в залежності від певних умов. Зазвичай ці умови формулюються у формі логічних виразів, які можуть бути істинними або хибними. Залежно від результату перевірки цих умов, виконується відповідний блок коду.

Переваги послідовного алгоритму з елементами розгалуження включають:

1. Гнучкість: Алгоритм може реагувати на різні ситуації і вибирати відповідні дії в залежності від умов. Це дозволяє програмі бути адаптивною і здатною здійснювати різні види обробки даних або взаємодії з користувачем.
2. Ефективність: За допомогою розгалужень можна уникнути зайвих обчислень або виконання непотрібних дій, оскільки алгоритм вибирає лише необхідні шляхи в залежності від умов. Це дозволяє зберігати ресурси системи і покращувати час виконання програми.
3. Читабельність: Послідовний алгоритм з елементами розгалуження зазвичай легко зрозуміти і читати. Він використовує стандартні конструкції, такі як умовні оператори (if-else, switch-case) або цикли, що дозволяють програмістам легко сприймати логіку алгоритму.
4. Надійність: Застосування розгалужень дозволяє обробляти різні ситуації та виявляти помилки. Наприклад, можна використовувати блоки try-catch для обробки виняткових ситуацій і забезпечувати стабільну роботу програми.

Хоча послідовний алгоритм з елементами розгалуження має свої переваги, в деяких випадках він може стати складним у керуванні і підтримці. Занадто багато

вкладених розгалужень може призвести до складного коду і збільшити ймовірність помилок. У таких випадках можуть застосовуватися інші типи алгоритмів, такі як ітеративні або рекурсивні, для більш ефективного управління потоком програми.

3.2. Метод покрокової деталізації

Опис процесу деталізації:

1. Отримання даних
    a. Отримання json файлу
        i. Відправка запиту
        ii. Опрацювання запиту
    b. Зчитування файлу
        i. Обробка та форматування
    c. Збір потрібної інформації
        i. Валідація файлу
        ii. Перевірка на присутність необхідних параметрів
    d. Збереження інформації
        i. Нормалізація даних
        ii. Зіставлення параметрів
        iii. Збереження
2. Обчислення даних
    a. Розрахунок HQi для кожної отриманої речовини
        i. Зіставлення результату обчислення з текстовим еквівалентом
    b. Розрахунок HQ по всіх речовинах
        i. Отримання значення та табличного значення речовини
        ii. Розрахунок за формулою
        iii. Повернення результату
    c. Розрахунок загального ризику
        i. Отримання значення та табличних значень речовини
        ii. Розрахунок за формулою

iii. Повернення результату

d. Розрахунок потенційного ризику

i. Отримання значення та табличних значень речовини

ii. Розрахунок за формулою

iii. Повернення результату

e. Збереження результатів

3. Повернення даних

a. Нормалізація даних

i. Підготовка інформації для кінцевого споживача

ii. Додаткове пост-опрацювання

b. Форматування даних

i. Кінцеве зіставлення даних для повернення відповіді

c. Відправка даних

i. Повернення коду відповіді

ii. Повернення відповіді

Дерево покрокової деталізації:

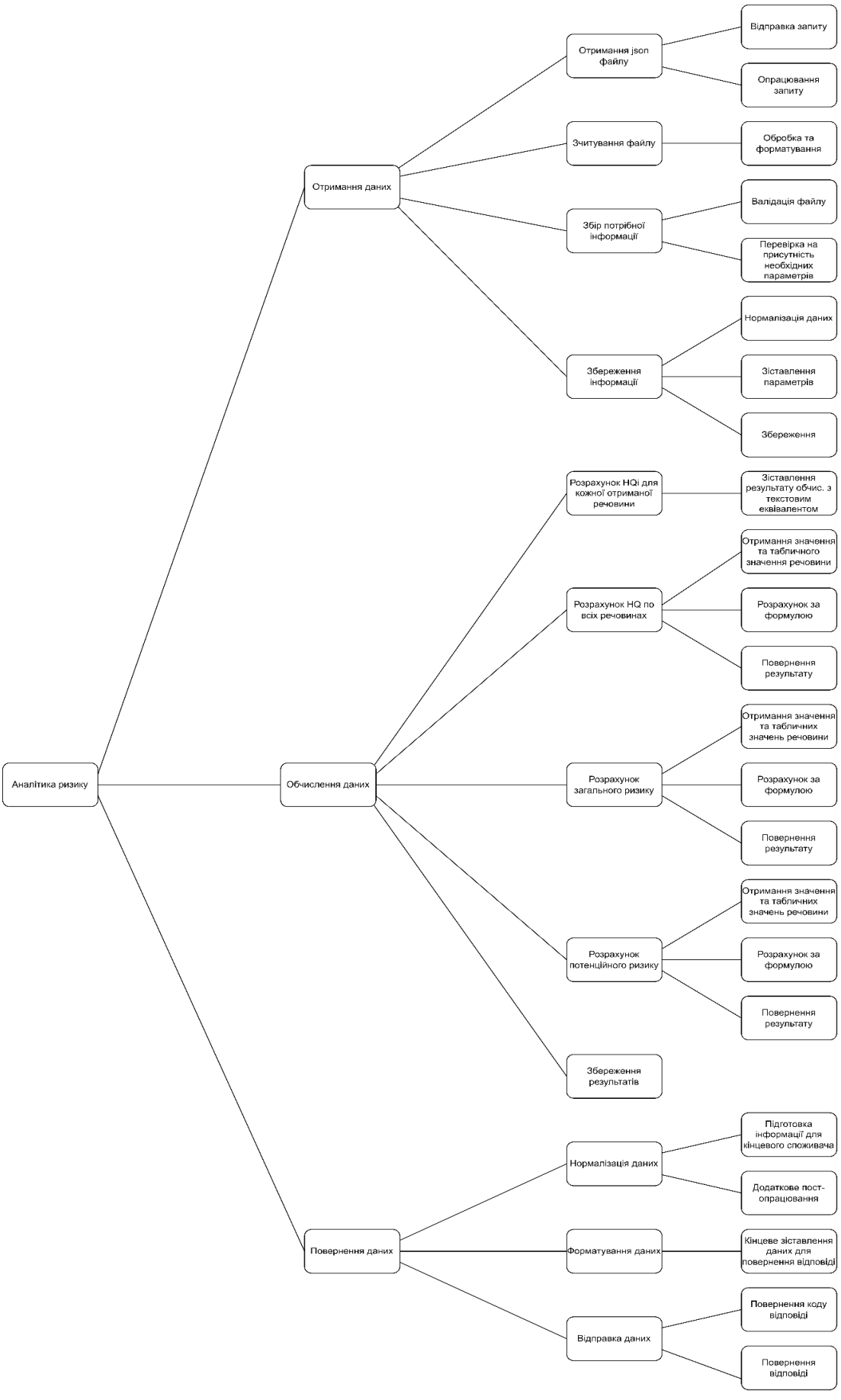


Рисунок 3.2 - Дерево покрокової деталізації

Висновки до розділу 3

В цьому розділі був проведений розбір алгоритму, який потрібно реалізовувати його властивостей та характеристик. Поставлена для розробника задача була розбита на під-задачі для покращення якості та пришвидшення процесу розробки програмного забезпечення.

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Юніт тестування:

Код тестів див. додаток А.

На меті було тестування лише правильності обчислення даних калькуляторами, бо інші компоненти мають достатньо засобів для запевнення правильності роботи системи.

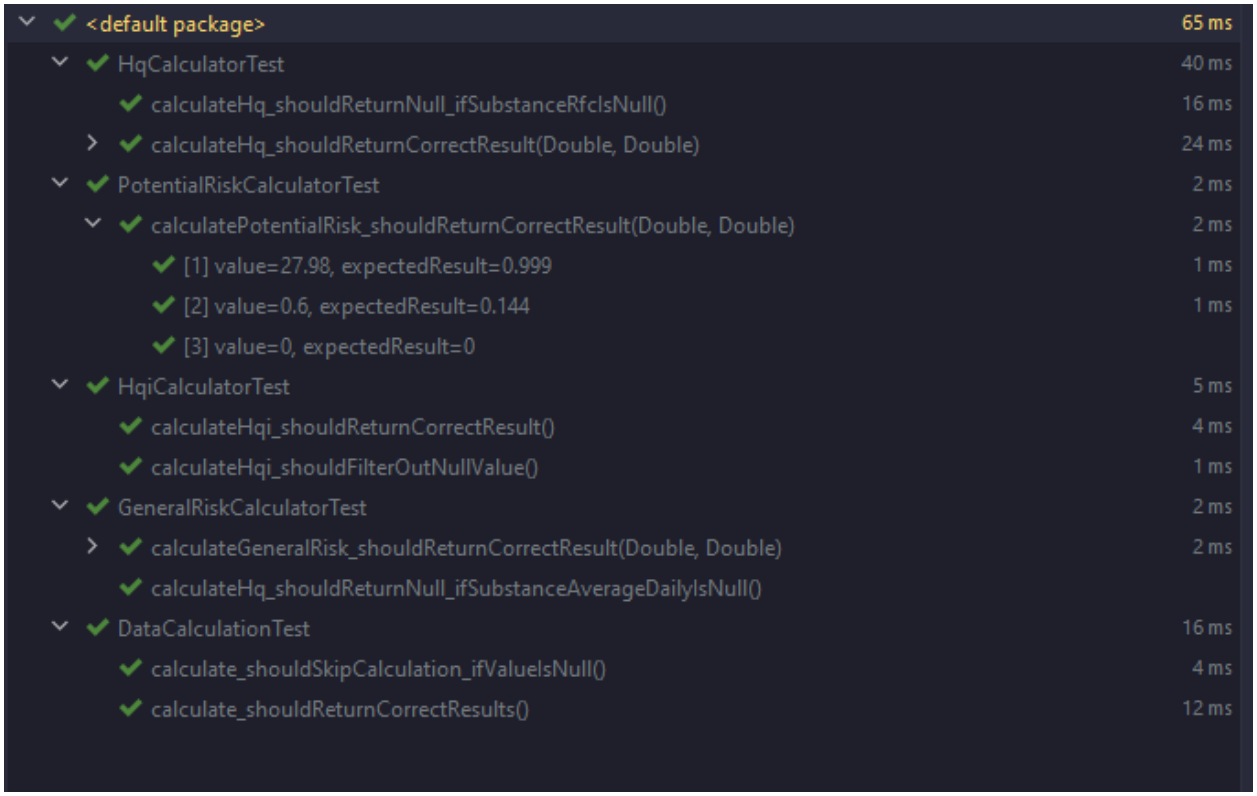


Рисунок 4.1 - Структура тестів

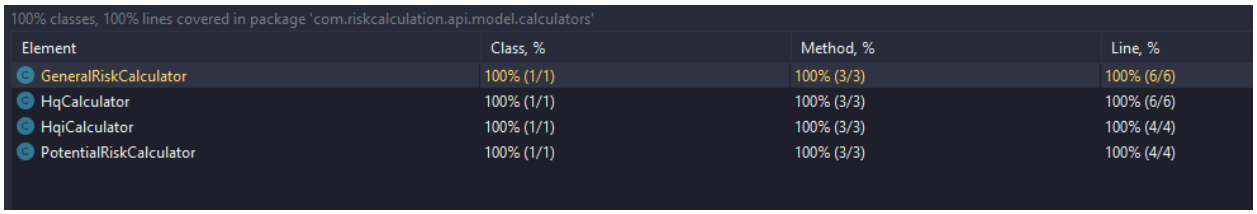
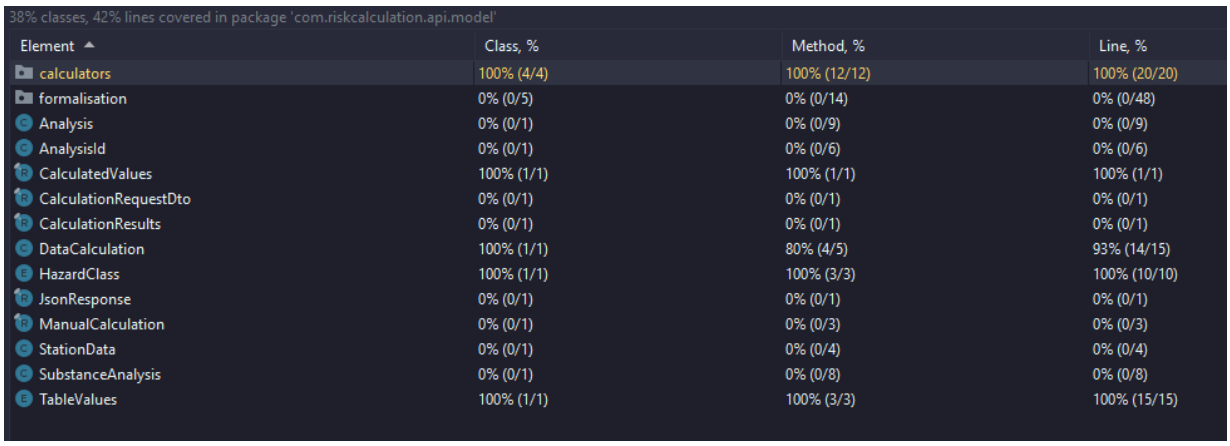


Рисунок 4.2 - Покриття тестів для класів



38% classes, 42% lines covered in package 'com.riskcalculation.api.model'

Element	Class, %	Method, %	Line, %
calculators	100% (4/4)	100% (12/12)	100% (20/20)
formalisation	0% (0/5)	0% (0/14)	0% (0/48)
Analysis	0% (0/1)	0% (0/9)	0% (0/9)
AnalysisId	0% (0/1)	0% (0/6)	0% (0/6)
CalculatedValues	100% (1/1)	100% (1/1)	100% (1/1)
CalculationRequestDto	0% (0/1)	0% (0/1)	0% (0/1)
CalculationResults	0% (0/1)	0% (0/1)	0% (0/1)
DataCalculation	100% (1/1)	80% (4/5)	93% (14/15)
HazardClass	100% (1/1)	100% (3/3)	100% (10/10)
JsonResponse	0% (0/1)	0% (0/1)	0% (0/1)
ManualCalculation	0% (0/1)	0% (0/3)	0% (0/3)
StationData	0% (0/1)	0% (0/4)	0% (0/4)
SubstanceAnalysis	0% (0/1)	0% (0/8)	0% (0/8)
TableValues	100% (1/1)	100% (3/3)	100% (15/15)

Рисунок 4.3 - Покриття тестів в рамках модуля моделі

Як ми можемо бачити тести надають очікуване покриття коду.

Тестування API:

Postman є популярним інструментом для тестування та взаємодії з API. Він надає зручний інтерфейс, який дозволяє створювати, надсилати та тестувати HTTP запити до API, а також аналізувати й отримувати відповіді з сервера.

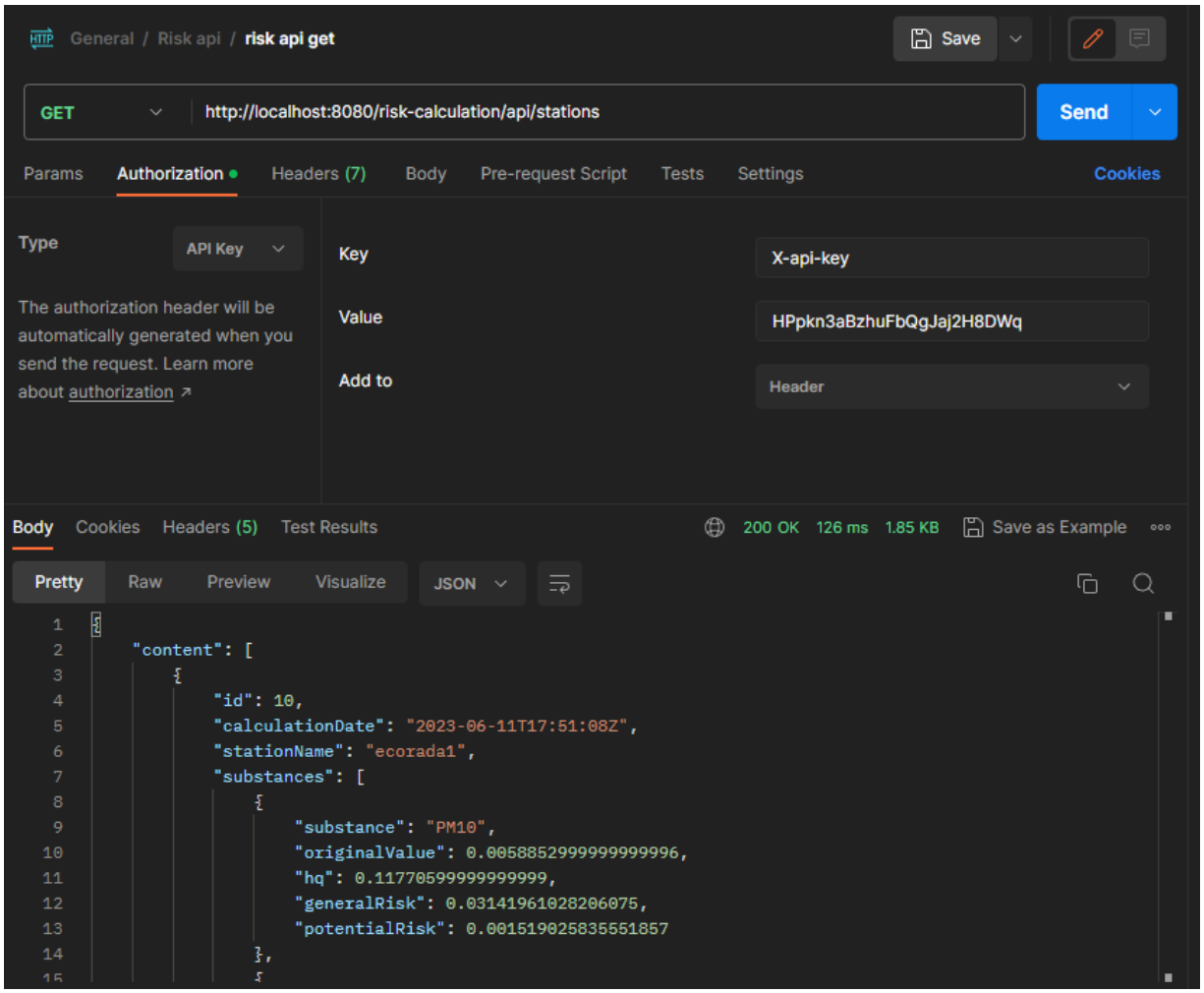


Рисунок 4.4 - Тестування запиту на отримання розрахованих даних по всіх станціях

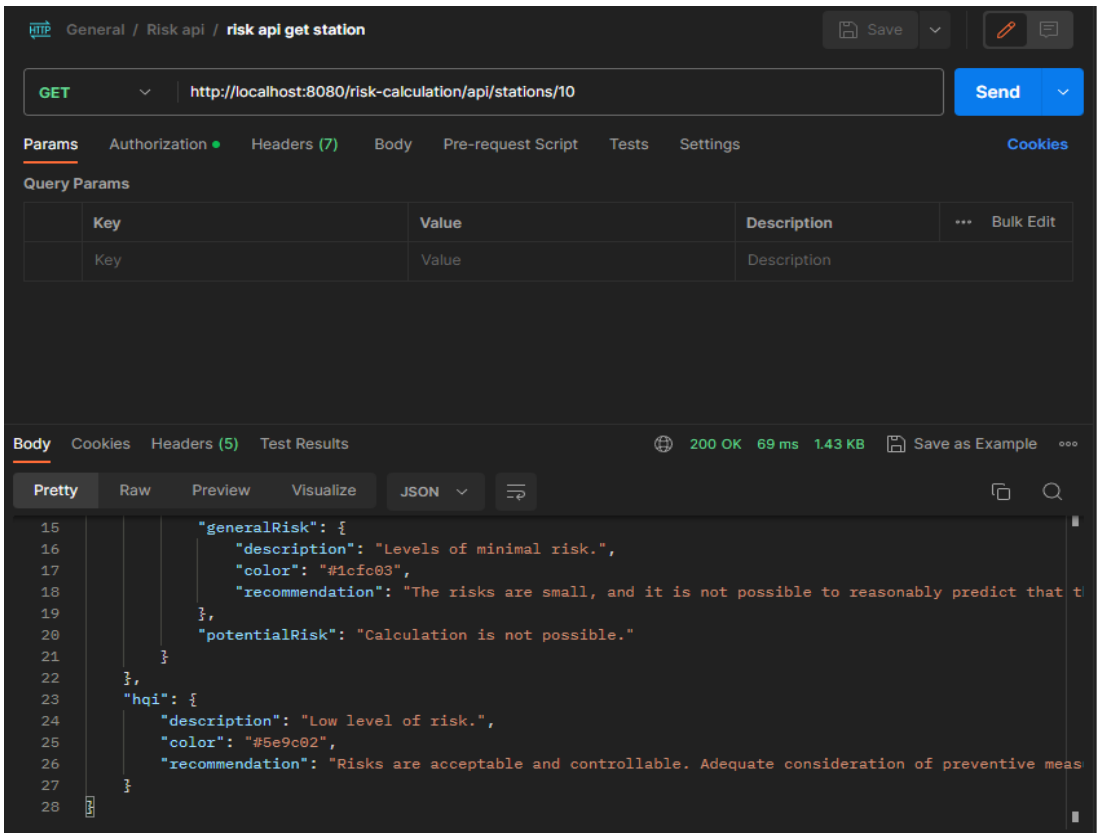


Рисунок 4.5 - Тестування запиту на отримання аналітики по станції

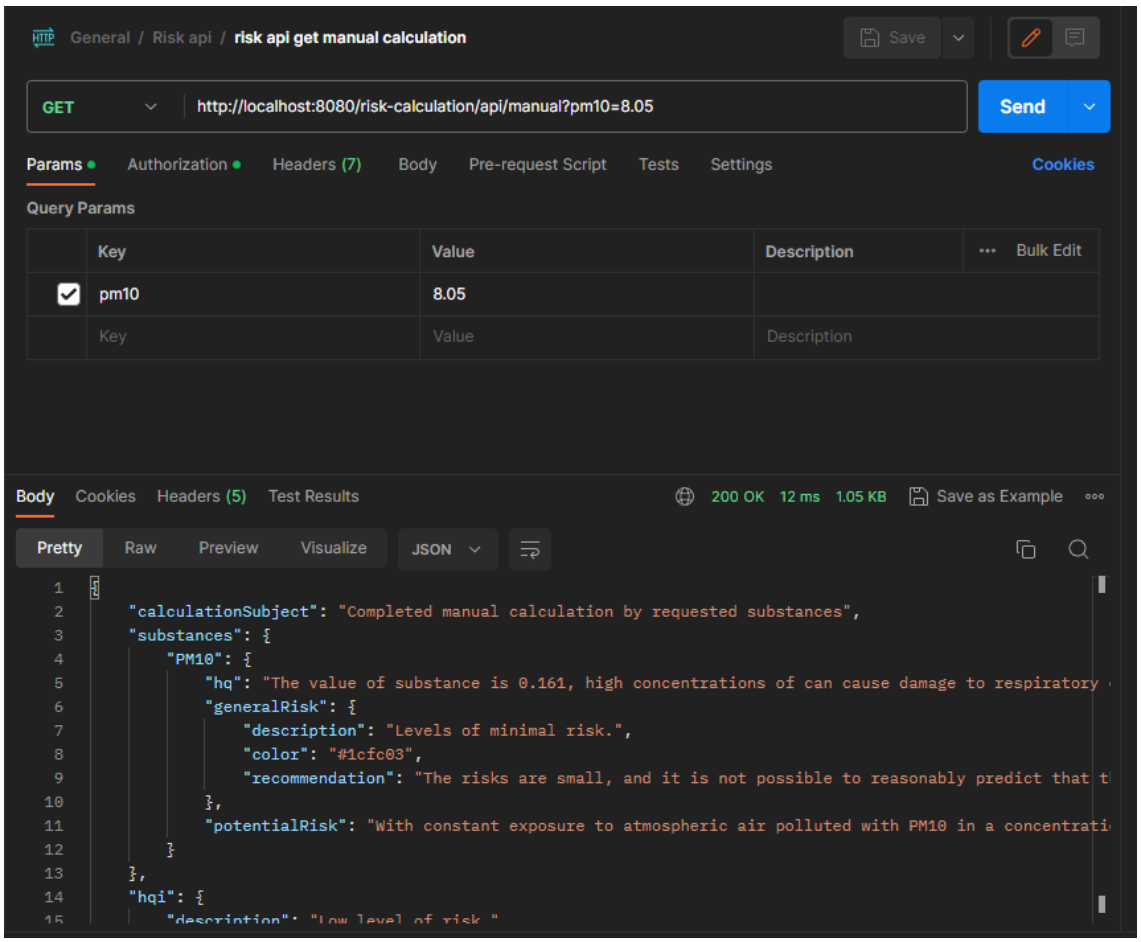


Рисунок 4.6 - Тестування запиту на отримання аналітики за мануально введеними речовининами

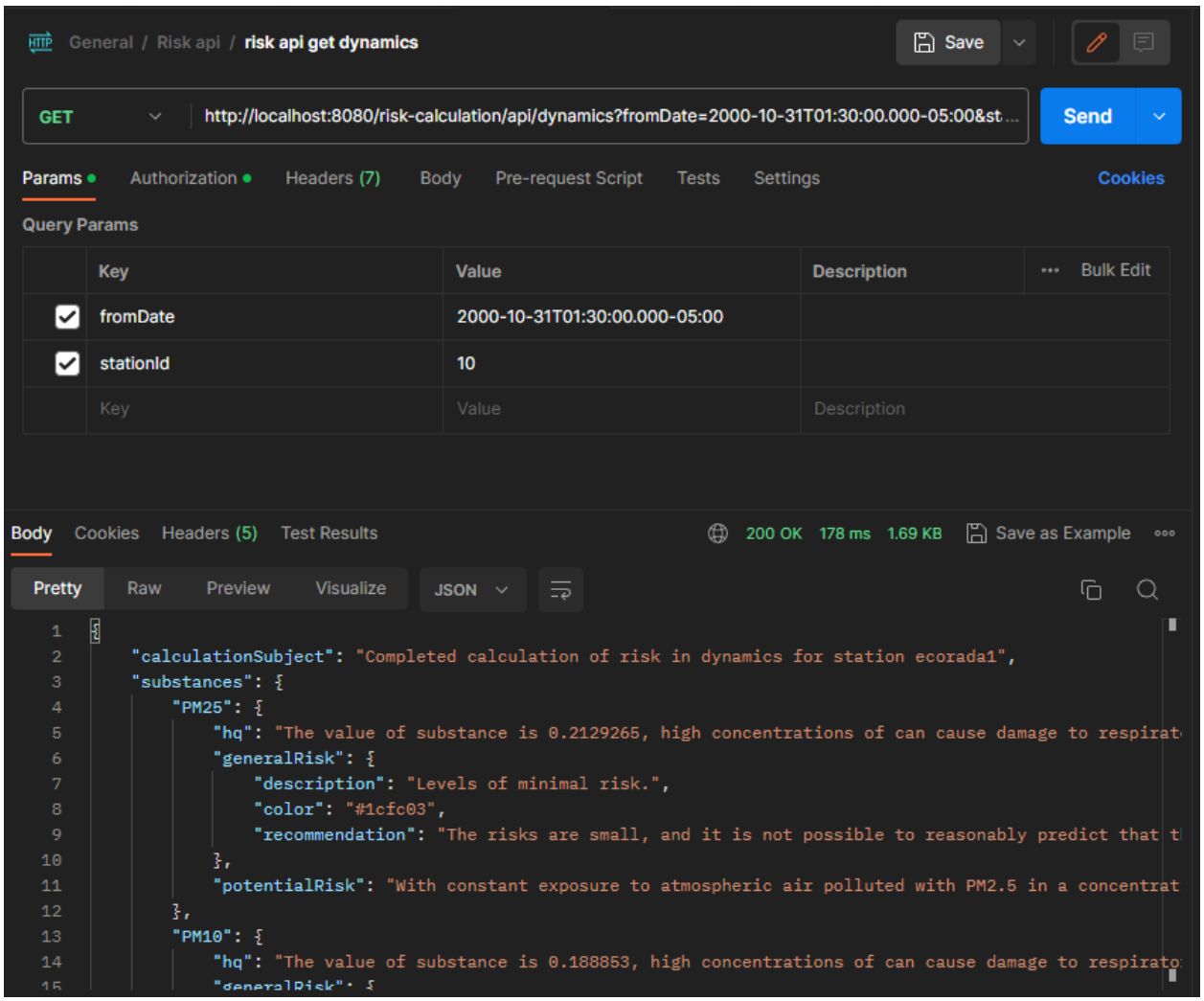


Рисунок 4.7 - Тестування запиту на отримання динаміки ризику по станції

Налагодження:

Окрім звичних способів налагодження: за допомогою компілятора, засобів IDE, синтаксичних аналізаторів, профайлерів, консолі налагодження та інших звичних для розробника методів, були використані додаткові методи для діагностики статусу роботоздатності та надійності розроблюваної системи. Це також пояснюється тим, що це контейнеризрванний додаток, тоб то має своє ізольоване середовище, це може ускладнювати процесс тестування та відладки в цілому.

Моніторинг та налагодження контейнеризованих додатків і сервісів у Docker є важливими аспектами ефективної експлуатації та забезпечення стабільності інфраструктури. Для цього можна використовувати різноманітні інструменти та практики. В даному випадку були використані такі методи:

1. Логування: Контейнеризація додатків використовує ізольовані середовища, тому важливо забезпечити збір та аналіз логів з кожного контейнера.

2. Метрики контейнерів: Docker має вбудовану підтримку для збору метрик процесора, пам'яті, мережі та інших ресурсів з контейнерів.

3. Огляд подій: Docker надає можливість переглядати та моніторити події, що стосуються контейнерів. Це може включати створення, запуск, зупинку, помилки тощо.

4. Автоматизоване тестування: Важливо мати набір автоматизованих тестів для перевірки працездатності та стабільності контейнерів та сервісів. Це може включати функціональні тести, навантажувальні тести, тести безпеки тощо.

5. Огляд мережі: Docker надає можливість налаштування мережі для контейнерів. Важливо відстежувати мережевий трафік та налаштування, щоб виявити проблеми з комунікацією, обмеженнями пропускної здатності або конфліктами портів.

Логування:

Логування подій грає важливу роль у тестуванні та налагодженні API додатків. Воно дозволяє збирати, зберігати та аналізувати інформацію про події, які відбуваються під час виконання програмного коду.

На приклад:

1. Відстеження виконання: Логи можуть містити повідомлення про кожну важливу дію або подію, яка відбувається в API. Це допомагає розуміти послідовність подій і виявляти помилки або неправильну поведінку.

2. Виявлення помилок: Логи можуть включати повідомлення про помилки або винятки, які виникають під час виконання API. Це дозволяє швидко виявляти проблеми та знаходити їх причину.

3. Моніторинг продуктивності: Логи можуть містити інформацію про час виконання окремих операцій або запитів до API. Це допомагає виявляти місця затримок та покращувати продуктивність додатку.

4. Відладка: Під час налагодження API розробники можуть розміщувати додаткові логи, щоб відстежувати значення змінних, контролювати виконання певних частин коду та виявляти причини неправильної роботи.

5. Аудит дій користувачів: Логи можуть бути корисними для аудиту дій користувачів API, щоб виявляти недозволену поведінку або вразливості системи.

Загалом, логування є потужним інструментом, який допомагає виявляти, аналізувати та вирішувати проблеми в API додатку, полегшує процес тестування та налагодження, а також покращує загальну якість продукту. Тому логуванню було приділено досить багато уваги в процесі розробки цього проекту.

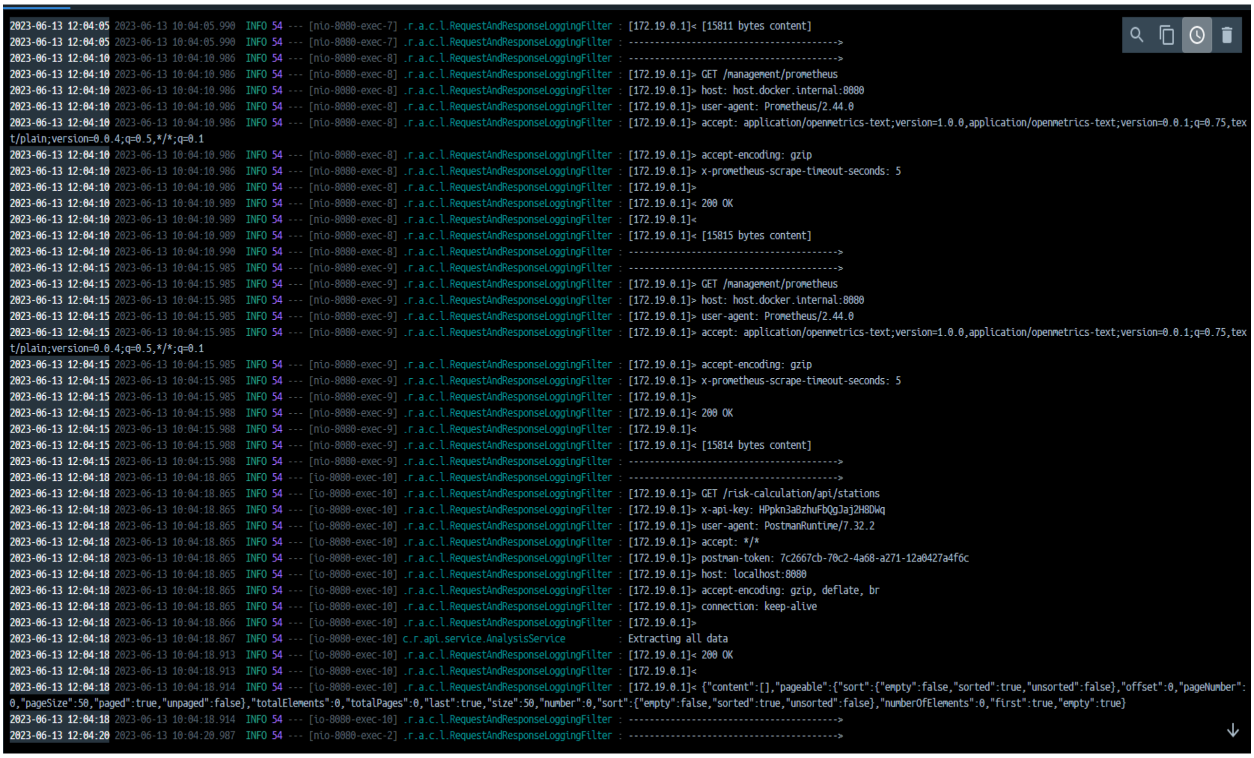


Рисунок 4.8 - Приклад логування у програмі

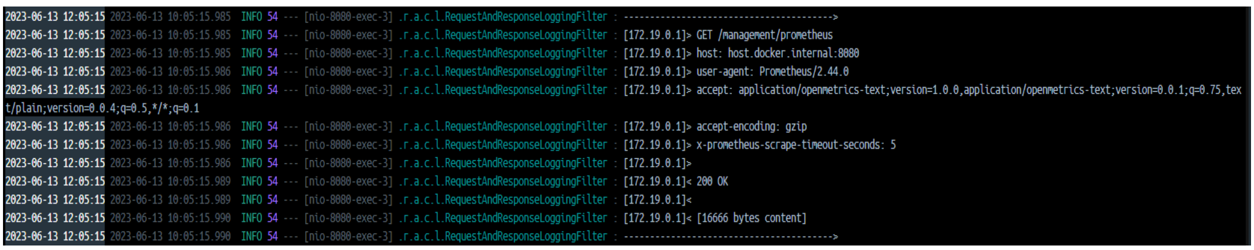


Рисунок 4.9 - Частний випадок логування запиту

Метрики та моніторинг:

Крім того для збору характеристик та статусу додатку був використаний Spring Actuator - це розширення фреймворку Spring Boot, яке надає можливість налагоджувати, моніторити та адмініструвати Java додаток. Він надає HTTP-ендпоінти, за допомогою яких можна отримати інформацію про стан додатку, виконати діагностику, збір метрик та багато іншого. Використання метрик Actuator дозволяє отримати цінну інформацію про продуктивність та ресурси API додатку, що сприяє ефективному налагодженню його роботи, виявленню проблем та оптимізації для досягнення найкращої продуктивності та надійності. Використання /actuator/health дозволяє здійснювати моніторинг стану додатку в реальному часі. Запит до цього ендпоінта повертає структуровану відповідь, яка містить інформацію про стан різних компонентів системи, таких як бази даних, зовнішні сервіси, пам'ять та інші. Це дозволяє оперативно виявляти проблеми, такі як недоступність джерел даних або несправності сервісів, та приймати відповідні заходи для їх вирішення.

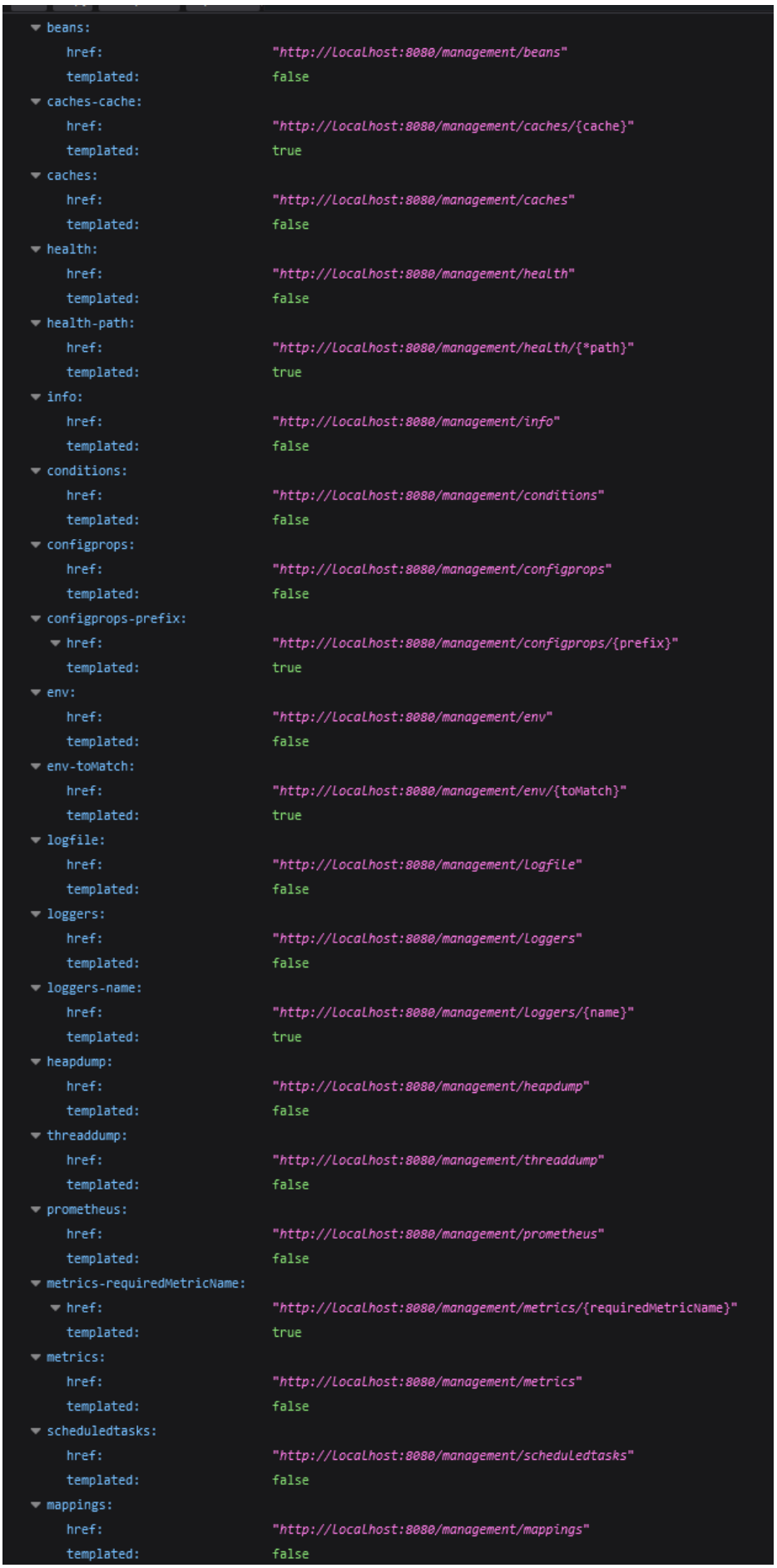


Рисунок 4.10 - Усі ендпоінти у actuator

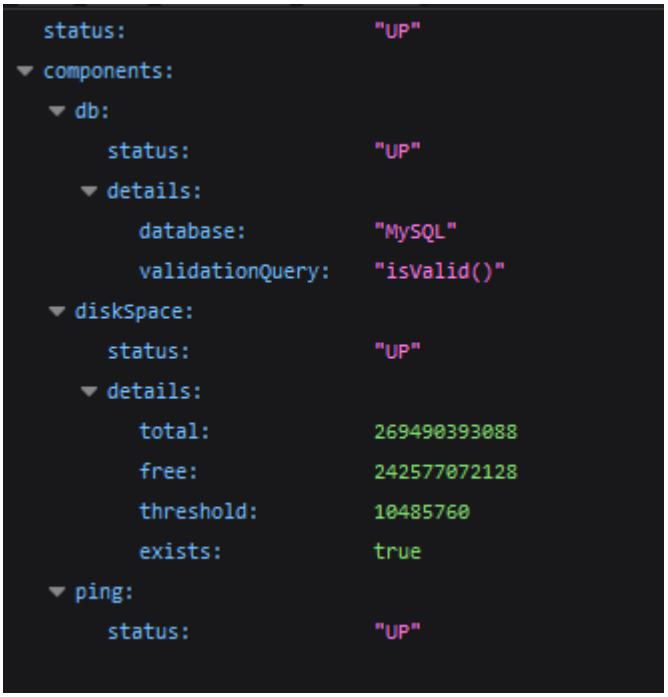


Рисунок 4.11 - Ендпоінт actuator/health

Використання Prometheus та Grafana для моніторингу та налагодження Spring Boot додатку є дуже популярним варіантом. Prometheus - це система моніторингу з відкритим вихідним кодом, а Grafana - це інструмент для візуалізації та аналізу даних з різних джерел. І така комбінація була використана у цьому проекті для загального моніторингу роботи програми та її компонентів.

```
# HELP disk_total_bytes Total space for path
# TYPE disk_total_bytes gauge
disk_total_bytes{path="/risk-calculation-api/.",} 2.69490393088E11
# HELP executor_pool_max_threads The maximum allowed number of threads in the pool
# TYPE executor_pool_max_threads gauge
executor_pool_max_threads{name="applicationTaskExecutor",} 2.14748364 7E9
executor_pool_max_threads{name="taskScheduler",} 2.14748364 7E9
# HELP process_cpu_usage The "recent cpu usage" for the Java Virtual Machine process
# TYPE process_cpu_usage gauge
process_cpu_usage 4.3535045711797995E-4
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_count_buffers{id="mapped",} 0.0
jvm_buffer_count_buffers{id="direct",} 11.0
# HELP jvm_memory_usage_after_gc_percent The percentage of long-lived heap pool used after the last GC event, in the range [0..1]
# TYPE jvm_memory_usage_after_gc_percent gauge
jvm_memory_usage_after_gc_percent{area="heap",pool="long-lived",} 0.010438254164678753
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual machine has started execution
# TYPE jvm_classes_unloaded_classes_total counter
jvm_classes_unloaded_classes_total 0.0
# HELP tomcat_sessions_rejected_sessions_total
# TYPE tomcat_sessions_rejected_sessions_total counter
tomcat_sessions_rejected_sessions_total 0.0
# HELP jvm_gc_overhead_percent An approximation of the percent of CPU time used by GC activities over the last lookback period or since monitoring began, whichever is shorter, in the range [0..1]
# TYPE jvm_gc_overhead_percent gauge
jvm_gc_overhead_percent 2.6378402001268745E-4
# HELP system_cpu_count The number of processors available to the Java virtual machine
# TYPE system_cpu_count gauge
system_cpu_count 12.0
# HELP process_files_max_files The maximum file descriptor count
# TYPE process_files_max_files gauge
process_files_max_files 1048576.0
# HELP system_cpu_usage The "recent cpu usage" of the system the application is running in
# TYPE system_cpu_usage gauge
system_cpu_usage 0.00130605137135394
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to use
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 2097152.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 9.6468992E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 6.8091904E7
jvm_memory_committed_bytes{area="nonheap",id="CodeCache",} 1.409024E7
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 1.59383552E8
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 9895936.0
# HELP hikaricp_connections_timeout_total Connection timeout total count
# TYPE hikaricp_connections_timeout_total counter
hikaricp_connections_timeout_total{pool="HikariPool-1",} 0.0
# HELP tomcat_sessions_active_current_sessions
# TYPE tomcat_sessions_active_current_sessions gauge
tomcat_sessions_active_current_sessions 0.0
# HELP logback_events_total Number of error level events that made it to the logs
# TYPE logback_events_total counter
logback_events_total{level="warn",} 1.0
logback_events_total{level="debug",} 0.0
logback_events_total{level="error",} 0.0
logback_events_total{level="trace",} 0.0
logback_events_total{level="info",} 407.0
# HELP process_files_open_files The open file descriptor count
# TYPE process_files_open_files gauge
process_files_open_files 127.0
# HELP hikaricp_connections_pending Pending threads
# TYPE hikaricp_connections_pending gauge
hikaricp_connections_pending{pool="HikariPool-1",} 0.0
# HELP system_load_average_1m The sum of the number of runnable entities queued to available processors and the number of runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 0.13
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.686650282357E9
# HELP hikaricp_connections_acquire_seconds Connection acquire time
# TYPE hikaricp_connections_acquire_seconds summary
hikaricp_connections_acquire_seconds_count{pool="HikariPool-1",} 5.0
hikaricp_connections_acquire_seconds_sum{pool="HikariPool-1",} 0.001594071
# HELP hikaricp_connections_acquire_seconds_max Connection acquire time
# TYPE hikaricp_connections_acquire_seconds_max gauge
hikaricp_connections_acquire_seconds_max{pool="HikariPool-1",} 5.62018E-4
# HELP hikaricp_connections_active Active connections
# TYPE hikaricp_connections_active gauge
hikaricp_connections_active{pool="HikariPool-1",} 0.0
# HELP disk_free_bytes Usable space for path
# TYPE disk_free_bytes gauge
disk_free_bytes{path="/risk-calculation-api/.",} 2.42576838656E11
# HELP jvm_threads_live_threads The current number of live threads including both daemon and non-daemon threads
# TYPE jvm_threads_live_threads gauge
```

Рисунок 4.12 - Метрики у Prometheus

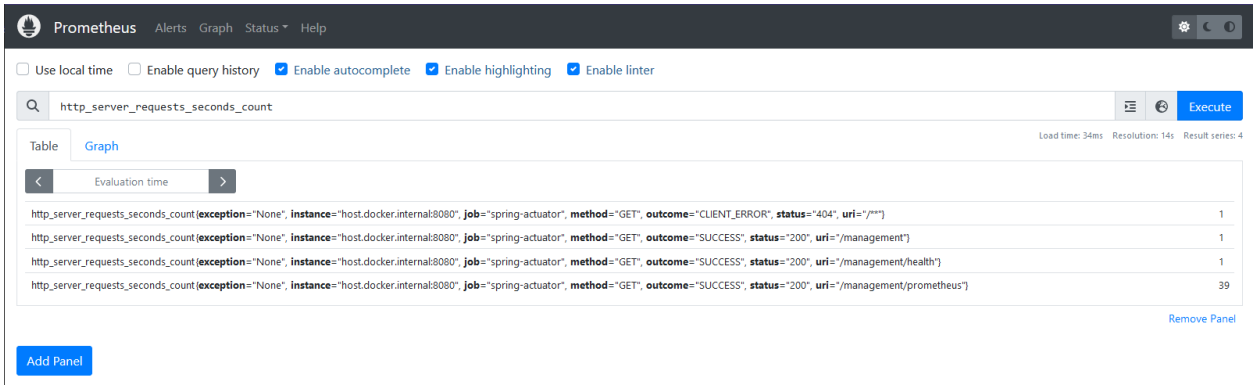


Рисунок 4.13 - Візуальний інтерфейс Prometheus, відображеня даних за метрикою

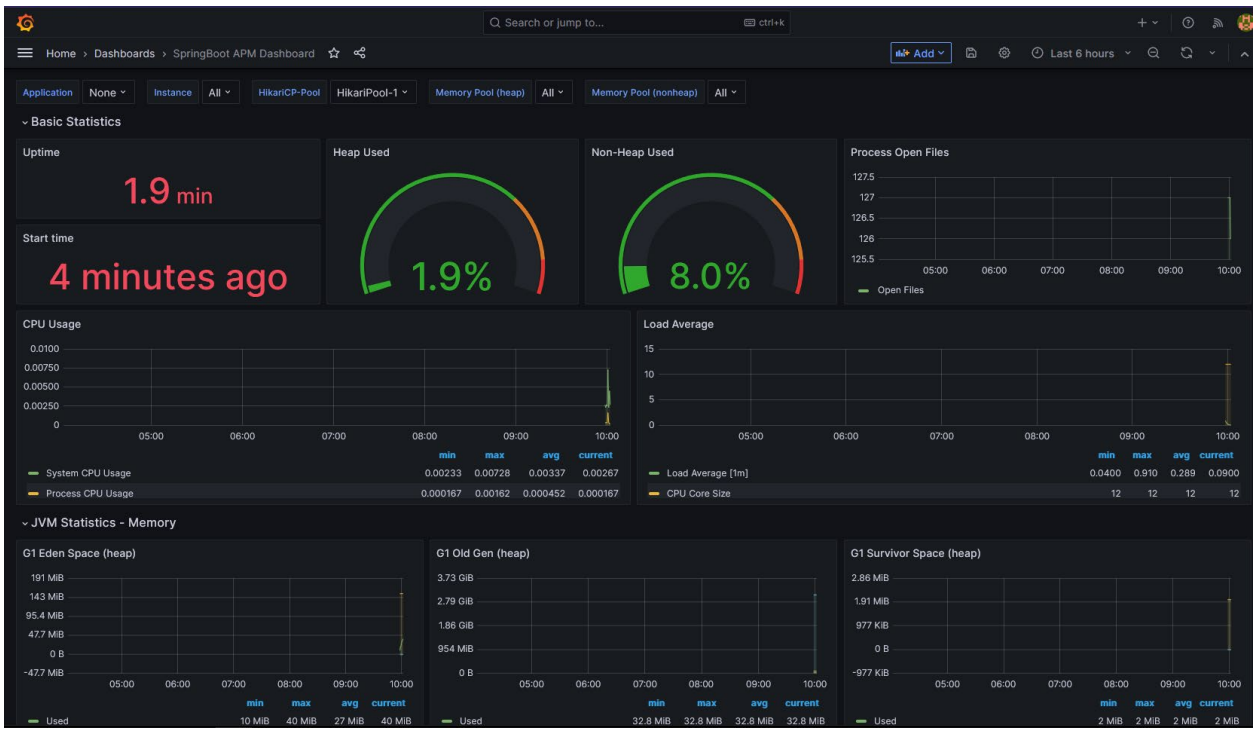


Рисунок 4.14 - Візуальний інтерфейс моніторингу у Grafana

Що стосується інструментів відстеження та огляду від Docker, то для поставленої задачі нативного інструментарію Docker було достатньо.

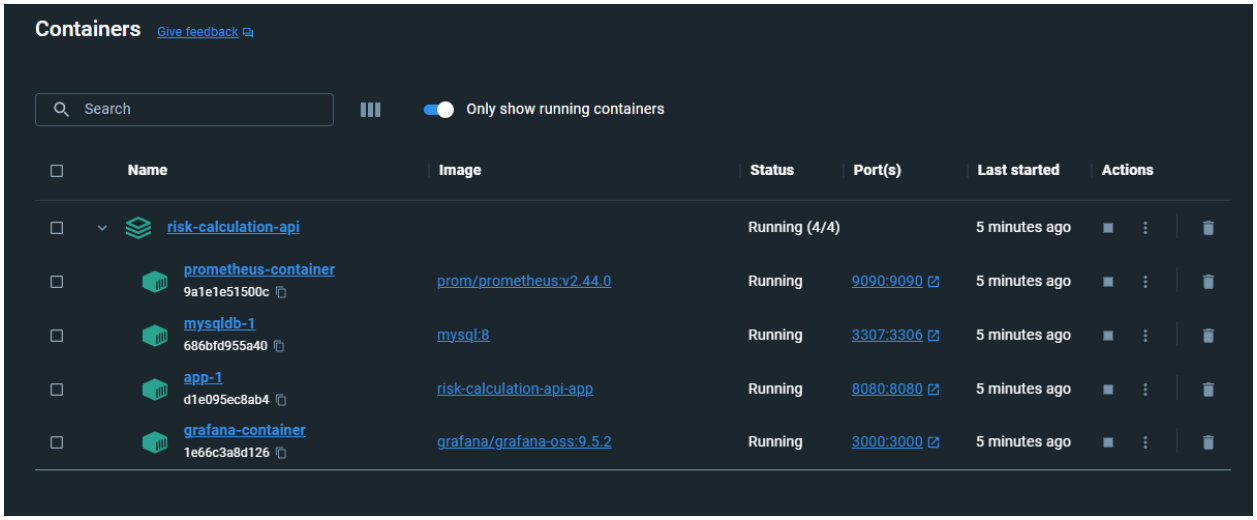


Рисунок 4.15 - Список запущених контейнерів у Docker та їх статус

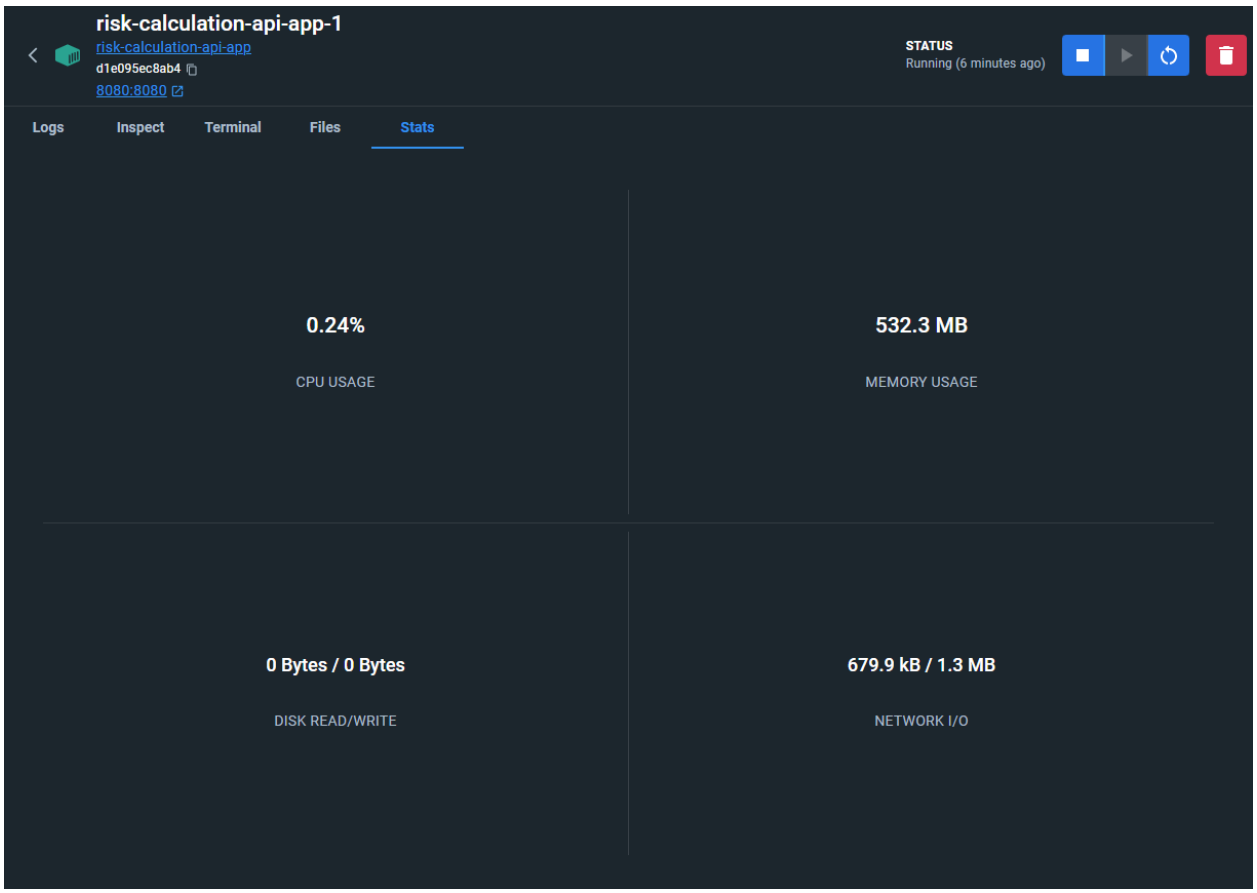


Рисунок 4.16 - Аналітика по використанню ресурсів контейнеру додатку

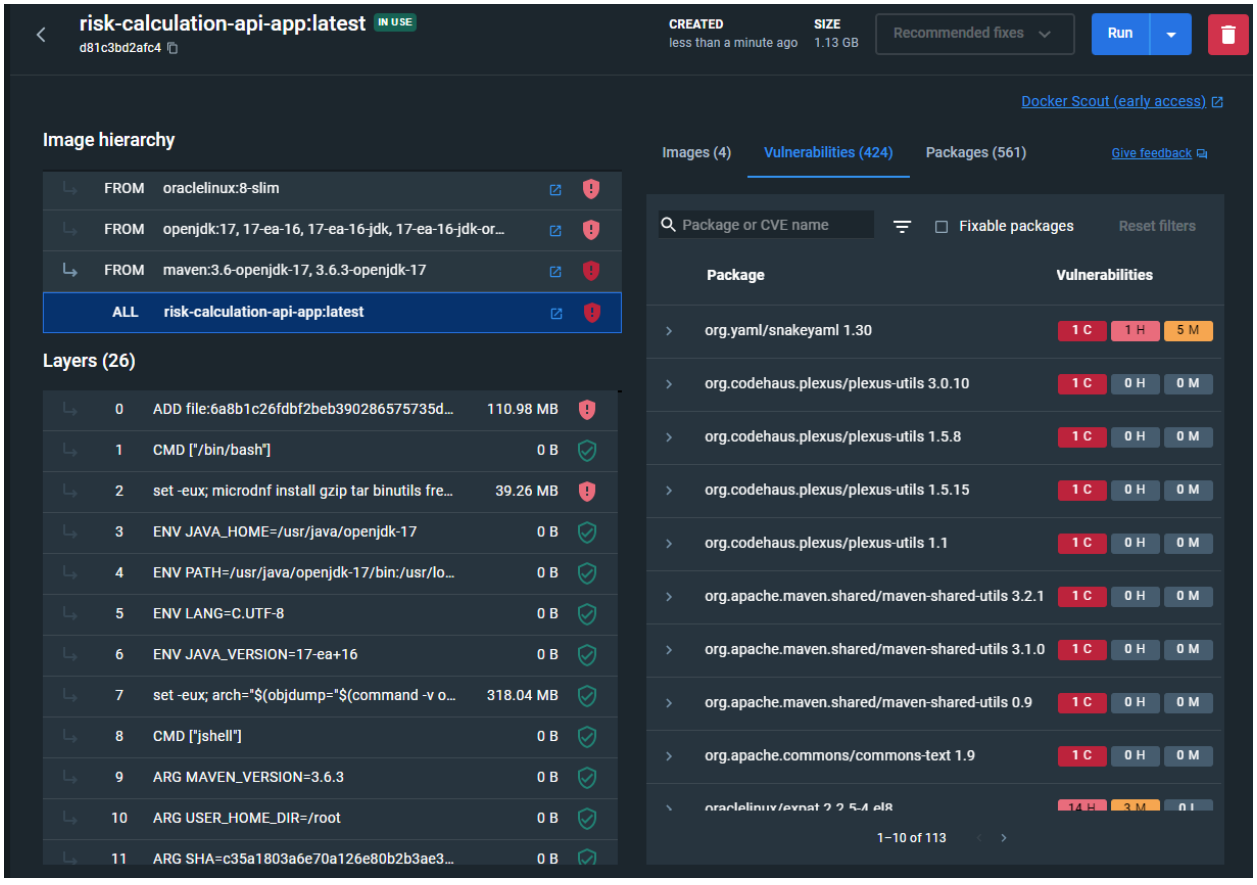


Рисунок 4.17 - Аналітика вузьких місць та вразливостей контейнеру додатку

Висновки до розділу 4

Під час процесу тестування та влагодження додатку, був використаний ряд методик та засобів для виявлення недоліків у системі з точки зору правильності роботи алгоритмів обчислення, безпеки, надійності, перевірки роботи функціоналу та загального моніторингу за станом додатку і його сервісів.

ВИСНОВКИ

В результаті виконання даної дипломної роботи була проведена розробка програми для обрахування ризику для здоров'я людини на основі показників повітря, поданих мануально або зібраних з API від eco-city.org.ua. Концепт подібного додатку взагалі не має альтернатив, тому такий інструмент дійсно потрібен для подальшого розвитку сфери екологічного аналізу ризиків для населення. Швидке та якісне інформування громадськості про стан навколишнього середовища допоможе покращити контролювання та запобігання потенціально-небезпечних для населення ситуацій, а також збереженню природних ресурсів та здоров'я населення. І дозволить оперативно виявляти проблемні ділянки та вживати заходів для їх усунення.

Проект розроблявся з метою використання програмного забезпечення як незалежного сервісу, з можливістю подальшого обслуговування, покращення та розширення функціоналу. Тому висока гнучкість, простота використання та інтеграції, стабільність – це властивості, яким було приділено багато уваги, що б забезпечити якість розробленого програмного комплексу.

Для спрощення експлуатації та розгортанні програмного забезпечення, була використана технологія контейнеризації. Зручність та простота технологій контейнеризації, таких як Docker, мають велике значення з точки зору експлуатації. Контейнеризація надає ізольоване середовище, де програмне забезпечення та його залежності можуть працювати відокремлено, запобігаючи конфліктам та забезпечуючи стабільність системи. Контейнери також забезпечують переносимість, що означає, що програмне забезпечення може бути легко розгорнуто на різних серверах без додаткових налаштувань. Завдяки швидкому розгортанню та масштабуванню контейнерів можна миттєво збільшити або зменшити обсяг оброблюваного навантаження. Керування контейнерами є простим завдяки інструментам, таким як Docker Compose або Kubernetes. Використання контейнеризації дозволяє ефективно використовувати ресурси серверів та спрощує розгортання, керування та підтримку програмного забезпечення.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Громадський моніторинг EcoCity якості повітря. ГО “Фрі Ардуіно” //https://eco-city.org.ua/.

2. Програма експрес-оцінки ризиків здоров'ю населення. ПРОЄКТ Інститут сталих інновацій //https://npo-kad.ru/activity/orzn/ekspress-otsenka/.

3. Copernicus. ПРОЄКТ //https://www.copernicus.eu/en.

4. World Air Quality Index project. ПРОЄКТ //https://aqicn.org/city/dnipro/ru/.

5. Sh**t! I Smoke. ПРОЄКТ // https://github.com/shootismoke/mobile-app.

6. Airq software tool for health risk assessment of air pollution. ПРОЄКТ World Health Organization //https://www.euro.who.int/en/health-topics/environment-and-health/air-quality/activities/airq-software-tool-for-health-risk-assessment-of-air -pollution.

7. Мартін, Роберт С. Чистий код: створення, аналіз та рефакторинг / Роберт С. Мартін ; пер. з англ. С. Гавриш ; ред. О. Матяшовський. - Київ : Вид-во "Дім Слова", 2011. - 464 с. : іл. - (Серія "Професійне програмування").

8. Моніторинг довкілля. Навч. посібник: у 2-х ч. – К.: Вид-во Європ. унту, 2007. – Ч.1. – 273с.

9. Боголюбов В. М., Клименко М. О., Мокін В. Б. Моніторинг довкілля / Вінниця: ВНТУ, 2010. 232 с.

10. Крайнюков О. М. Моніторинг довкілля / Харків: ХНУ ім. В. Н. Каразіна, 2009. 176 с.

11. Реформа державної системи моніторингу довкілля. Офіційний сайт Міністерства захисту довкілля та природних ресурсів України: https://mepr.gov.ua/topics/novyny/ekologichnyj-monitoryng-novyny/.

12. Екологічний моніторинг війни проти України та стратегія відновлення. ПРОЄКТ ОБСЄ //https://www.osce.org/uk/node/542988.

13. Encyclopedia of Atmospheric Sciences. 2nd ed. Eds: J.A. Curry, J.A. Pyle. – London: Academic Press, 2015. – 2998 p.

14. Matsyura, A.V. (2004). Review of computer programs for biological and ecological research. Part I. Applcation of Oriana in avian research. Zapobidna sprava v Ukraini, 10(1–2), 98–99.

15. Kazmirchuk, V.O., Savrun, B.Y., & Tsybulia S.A. (2015). The system of ecological support of the Armed Forces of Ukraine, ways and directions of its transformation into the system of ecological safety management. Military-Technical Collection, (13), 120-126. doi: https://doi.org/10.33577/2312-4458.13.2015.120-126.

16. Matsyura, A.V. (2005). Species Diversity and Richness and its application for the analysis of communities structure. Vestnik Dnepropetrovskogo natsionalnogo universiteta, Seria Biologia, Ecologia, 13(2), 124– 128.

17. Matsyura, A.V. & Matsyura, M.V. (2006a). Mathematical methods of population size estimation on the base of banding and capture-recapture data. Naukovi zapiski Ternopilskoho natsionalnogo pedagogichnogo universitet, 1(28), 73–76.

18. Batschelet E. Statistical methods for the analysis of problems in animal orientation and certain biological rhythms / E. Batschelet // Am. Inst. Sci., Washington, 1985. – 158 p.

19. Begon M. Abuses of mathematical techniques in Ecology: applications of Jolly`s capture–recapture method / M. Begon // Oikos. – 1983. – Vol. 40. – P. 155–158.

20. Matsyura, A.V & Matsyura, M.V. (2006b). Review of computer programs for biological and ecological research. Part III. Appilcation of Axis for the circular data estimation. Zapovidna sprava v Ukraini. 12(10, 83–85.

21. Cormack R.M. Log–linear models for capture–recapture / R.M. Cormack // Biometrics. – 1989. – Vol. 45(2). – P. 395–414.

22. Batschelet E. Circular statistics in biology / E. Batschelet. – Academic Press, London. – 1981. – 272 p.

23. Engeman R.M. A comparison of plotless density estimators using Monte Carlo simulation / R.M. Engeman, R.T. Sugihara, L.F. Pank // Ecology. – 1994. – Vol. 75(6). – P. 1769–1779.

24. Crain B.R. Nonparametric estimation of population density for line transect sampling using Fourier series / B.R. Crain, K.P. Burnham, D.R. Anderson // Biometrical Journal. – 1989. – Vol. 21. – P. 731–748.

25. Ensign W.E. Use of line transect methods of estimate abundance of benthic stream fishes / W.E. Ensign, P.L. Angermeier, C.A. Dolloff // Can. J. Fish. Aq. Sci. – 1995. – Vol. 52(1). – P.213–222.

26. Gadagkar R. An undesirable property of Hill`s index N2 / R. Gadagkar // Oecologia. – 2005. – Vol. 80. – P. 140–141.

27. Hutchinson G.E. A Theoretical Ecological Model of Size Distribution / G.E. Hutchinson, R.H. MacArthur // American Nature. – 1989. – Vol. 93. – P. 117–125.

28. Geisser P.H. Topics in route–regression analysis / P.H. Geisser, J.R. Sauer // Survey designs and statistical methods for the estimation of avian populations trends. – Washington: U.S. Fish and Wildlife service. – 1990. – P. 85–97.

29. Montgomery D.C. Forecasting and time series analysis (2nd Ed.) / Montgomery D.C., Johnson L.A., Gardiner J.S. – New York: McGraw–Hill, 1990. – P. 56–70.

30. Guisan A. Predictive habitat distribution models in ecology / A. Guisan, N.E. Zimmermann // Ecological Modelling. – 2000. – Vol. 135. – P. 147–186.

31. STATISTICA. Electronic manual. – StatSoft, Inc.: Bedford, 2002. SYSTAT. – SYSTAT, Inc.: Evanston, 1989. – P. 55–57.

32. Hurvich C.M. Regression and time series model selection in small samples / C.M. Hurvich, C.L. Tsai // Biometrika. – 1989. – Vol. 76. – P. 297–307.

33. Jackson D.A. Similarity coefficients: measures of co–occurence and association or simply measures of occurrence? / D.A. Jackson, K.M. Somers, H.H. Harvey // Amer. Nat. – 1989. – Vol. 133. – P.436–453.

34. Southwell C. Evaluation of analytical procedures for density estimation from line–transect data: Data grouping, data truncation and the unit of analysis / C. Southwell, K. Weaver // Wildlife Research. – 1999. – 20(4). – P. 433–444.

35. Ter Braak C.J.F. Analysis of monitoring data with many missing values: which method? // The European Union and Biodiversity [W. Hagemeijer] / C.J.F. Ter Braak, A.J. van Strien, R. Meijer. – Brussels: Friends of the Earth & EEB, 1998. – 76 p.

36. Van Dop H., Frans T.M. Nieuwstadt // Atmospheric Turbulence and Air Pollution Modelling. – Dordrecht: D. Reider Publishing Company, 1981. – 358 p.

37. The comparison of Lagrangian and Gaussian models in predicting of air pollution emission using experimental study, a case study: Ammonia emission / M. Asadi, G. Asadollahfardi, H. Fakhraee, M. Mirmohammadi // Environmental Modeling & Assessment. – 2017. – V. 22. – P. 27–36.

38. Pasquill F., Smith F.B. Atmospheric diffusion. 3rd ed. – Chichester: Ellis Horwood Ltd., 1983. – 429 p.

39. Wark K., Warner C.F., Davis W.T. Air Pollution. Its Origin and Control. 3rd ed. – Menlo Park: Addison Wesley Longman Inc., 1998. – 573 p.

40. Characterization of solid airborne particles deposited in snow in the vicinity of urban fossil fuel thermal power plant (Western Siberia) / A.V. Talovskaya, E.G. Yazikov, E.A. Filimonenko, J.C. Lata, J. Kim, T.S. Shakhova // Environmental Technology. – 17 September 2018. – V 39. – Iss. 18. – P. 2288–2303.

41. Tonhasca A. Diversity indices in the analysis of biological communities / A. Tonhasca // Cienc. et cult. – 1994. – Vol. 3. – P. 138–140.

42. Fluorine concentration in snow cover within the impact area of aluminium production plant (Krasnoyarsk city) and coal and gas fired power plant (Tomsk city) / A.V. Talovskaya, N.A. Osipova, S.A. Polikanova, N.P. Samokhina, E.A. Filimonenko, E.G. Yazikov, I.A. Matveenko // IOP Conference Series: Earth and Environmental Science. – 2015 – V. 27. – P. 012043–1–012043–6.

43. Mandurino C., Vestrucci P. Using meteorological data to model pollutant dispersion in the atmosphere // Environmental Model ling & Software. – 2009. – V. 24. – Iss. 2. – P. 270–278.

44. Duynkerke P.G. Application of the E turbulence closure model to the neutral and stable atmospheric boundary layer // Journal of the Atmospheric Sciences. – 1988. – V. 45. – № 5. – P. 865–880.

45. Arya S. Pal Air Pollution Meteorology and Dispersion. 1st ed. – Oxford: Oxford University Press. 1998. – 320 p.

46. Safety assessment for facilities and activities. International Atomic Energy Agency Safety Standards Series No. GSR. Part 4 (Rev. 1). Vienna, IAEA, 2016. 63 p.

47. Fractional derivative models for atmospheric dispersion of pollutants / A.G.O. Goulart, M.J. Lazo, J.M.S. Suarez, D.M. Moreira // Physica A: Statistical Mechanics and its Applications. – 2017. – V. 477. – P. 9–19.

48. Spring Framework Documentation. spring: https://docs.spring.io/spring-framework/docs/current/reference/html/

49. Hibernate Documentation. Hibernate: https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html

50. Oracle. "Java EE Documentation." oracle: https://docs.oracle.com/javaee/

51. A. S. Torres, "Building Web Applications with Java Persistence API," International Journal of Software Engineering and Its Applications, vol. 10, no. 5, pp. 235-248, 2016.

52. E. M. Schijff, "Web Applications in Java: A Developer's Guide," O'Reilly Media, 2009.

53. M. Sikora, "Java Web Development with Spring and Hibernate," Packt Publishing, 2019.

54. S. Haloi, "Java Web Services: Up and Running," O'Reilly Media, 2013.

55. Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John M. Design Patterns: Elements of Reusable Object-Oriented Software. 1: Addison-Wesley Professional, 1994.

56. Freeman, Eric, Freeman, Elisabeth, Bates, Bert and Sierra, Kathy. Head First Design Patterns. 1 Edited by Mike Loukides.: O'Reilly Media, 2004.

57. Walls, Craig. Spring in Action. Shelter Island: Manning, 2011.

58. Dinesh Rajput, Spring 5 Design Patterns: Master Efficient Application Development with Patterns Such as Proxy, Singleton, the Template Method, and More, Packt Publishing, 2017, c – 396.

59. Rajput, Dinesh., R V, Rajesh. Building Microservices with Spring: Master Design Patterns of the Spring Framework to Build Smart, Efficient Microservices. United Kingdom: Packt Publishing, 2018.

ДОДАТК

ЗАТВЕРДЖЕНО

1116130.01291-02-ЛЗ

Специфікація

ЗАСТОСУНОК ДЛЯ ЗБОРУ ДАНИХ ПРО НАКОЛИШНЄ СЕРЕДОВИЩЕ ТА АНАЛІЗУ ЙОГО ПОТОЧНОГО СТАНУ

Специфікація

Листів 2

1116130.01291-02-ЛЗ

2023

Позначення	Найменування	Примітка
1116130.01291-01-ЛЗ	Лист затвердження	
1116130.01291-01	Технічне завдання	
1116130.01291-02-ЛЗ	Лист затвердження	
1116130.01291-02	Специфікація	
1116130.01291-01 12 01-ЛЗ	Лист затвердження	
1116130. 01291-01 12 01	Текст програми	
1116130. 01291-01 13 01-ЛЗ	Лист затвердження	
1116130. 01291-01 13 01	Опис програми	