ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Веб-система для автоматизації процесів надання пільгової продукції на основі відкритих API та баз даних»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ Богдан Бакал
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Богдан БАКАЛ
група КНДМ-61

Керівник: Юлія БЕРЕЗОВСЬКА
науковий ступінь, вчене звання д.к.к.н., доктор філософії

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бакалу Богдану Олександровичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Веб-система для автоматизації процесів надання пільгової продукції на основі відкритих API та баз даних»

керівник кваліфікаційної роботи Юлія БЕРЕЗОВСЬКА д.к.к.н., доктор філософії,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи інтеграції зовнішніх сервісів через відкриті API, принципи захисту даних, основи автоматизації управління доступом.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1. Дослідження методів статистичного аналізу та методів візуалізації даних за допомогою програмного забезпечення
   4.2. Проектування прототипу системи та підбір технологій для її реалізації.
   4.3. Розробка прототипу системи візуалізації та статистичного аналізу даних.

5. Перелік графічного матеріалу: *презентація*
 5.1 Основні методи візуалізації даних за допомогою діаграм та таблиць.
 5.2 Підходи до проектування прототипу системи.
 5.3 Приклади програмного коду.
 5.4 Головні компоненти створеної системи.
6. Дата видачі завдання«16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.24	виконано
2	Аналіз методів візуалізації та статистичного аналізу	30.09-11.10.24	виконано
3	Проектування та підбір технологій реалізації	14.10-25.10.24	виконано
4	Розробка серверної частини	28.10-08.11.24	виконано
5	Розробка клієнтської частини	11.11-22.12.24	виконано
6	Головні компоненти створеної системи	25.11-29.11.24	виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-06.12.24	виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.24	виконано

Здобувач вищої освіти _____ Богдан БАКАЛ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Юлія БЕРЕЗОВСЬКА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 102 стор., 38 рис., 25 джерел.

Наукове завдання – розробка веб-системи для автоматизації процесів надання пільгової продукції, що включає інтеграцію з відкритими API та базами даних для контролю доступу та управління правами користувачів.

Мета роботи – підвищити ефективність процесу надання пільгової продукції шляхом розробки веб-системи, що автоматизує перевірку прав доступу та обробку запитів через API, забезпечуючи точність, швидкість та прозорість процесів.

Об'єкт дослідження – процес автоматизації надання пільгової продукції з використанням відкритих API та баз даних для ефективного контролю доступу та управління правами користувачів.

Предмет дослідження – технології та методи інтеграції веб-систем з відкритими API та базами даних для автоматизації процесів надання пільгової продукції, включаючи механізми авторизації, перевірки прав доступу та обробки запитів у реальному часі.

Методи дослідження – аналіз та узагальнення методів автоматизації процесів через інтеграцію API, методи розробки веб-систем для управління правами доступу, алгоритмізація обробки запитів та перевірки прав доступу. Також використовувались методи статистичного аналізу для оцінки ефективності розробленої системи.

Короткий зміст роботи: У роботі проведено дослідження сучасних технологій для автоматизації надання пільгової продукції, зокрема застосування відкритих API та баз даних для інтеграції з різними сервісами та управління доступом. Описано, як автоматизація цього процесу може підвищити ефективність, знизити кількість помилок, а також забезпечити точність і актуальність даних у реальному часі.

Розроблено веб-систему, яка складається з двох основних модулів: перший відповідає за інтеграцію з зовнішніми сервісами та перевірку прав доступу користувачів у реальному часі, а другий дозволяє управляти списками осіб, які мають право на отримання пільгової продукції, а також надає можливість додавати або видаляти користувачів.

Система автоматизує процеси видачі пільгової продукції, забезпечує прозорість обліку та мінімізує ризик помилок, пов'язаних із ручним введенням даних. Особливу увагу приділено безпеці даних, зокрема використанню методів аутентифікації та авторизації через ASP.NET Core Identity та JWT для забезпечення захисту особистої інформації користувачів.

Результатом роботи є прототип веб-системи, яка дозволяє автоматизувати процеси надання пільгової продукції, покращити ефективність управління цими процесами та забезпечити точність і прозорість обліку в реальному часі.

Ключові слова: автоматизація процесів, відкриті API, веб-система, пільгові продукти, база даних, контроль доступу, інтеграція, безпека даних, авторизація, реальний час.

ABSTRACT

Text part of the master's qualification work: 102 pages, 39 pictures, 25 sources.

Scientific task is development of a web system for automating the processes of providing discounted products, integrating open APIs and databases for access control and user rights management.

Goal of the work is to improve the efficiency of providing discounted products by developing a web system that automates access rights verification and request processing through APIs, ensuring speed, accuracy, and transparency of the processes..

Object of research – The process of automating the provision of discounted products using open APIs and databases for effective access control and user rights management.

Subject of research – Technologies and methods of integrating web systems with open APIs and databases for automating the process of providing discounted products, including authorization mechanisms, access rights verification, and real-time request processing.

Research methods – Analysis and generalization of methods for process automation via API integration, web system development techniques for access control, algorithmization of request processing and access rights verification. Statistical analysis methods were also used to assess the efficiency of the developed system.

Summary of the work: This work investigates modern technologies for automating the provision of discounted products, specifically the use of open APIs and databases for integration with external services and access control. The paper outlines how automation of this process can enhance efficiency, reduce error rates, and ensure real-time data accuracy and relevance.

A web system was developed, consisting of two main modules: the first integrates with external services and verifies user access rights in real time, while the second manages lists of individuals eligible for discounted products and provides the ability to add or remove users.

The system automates the process of providing discounted products, ensuring transparency in accounting and minimizing the risk of errors related to manual data entry. Special attention was paid to data security, including the use of OAuth 2.0 and JWT for protecting personal user information.

The result of this work is a web system prototype that automates the processes of providing discounted products, improving the efficiency of managing these processes and ensuring real-time accuracy and transparency of data.

Keywords: process automation, open APIs, web system, discounted products, database, access control, integration, data security, authorization, real-time.

ЗМІСТ

ВСТУП

У сучасному суспільстві, де технологічний розвиток відбувається швидкими темпами, помітно зростає попит на автоматизацію та ефективність обробки даних. Особливо це стосується сфер, де точність, достовірність, оперативність та прозорість інформації має велике значення. Наприклад, у процесі надання привілейованих товарів, де необхідний високий ступінь довіри та контролю, швидка обробка та оновлення даних, доступ до актуальної інформації в режимі реального часу та мінімізація помилок при обміні даними між різними системами є ключовими факторами. Одним з найважливіших інструментів для досягнення цих цілей є використання відкритих API (інтерфейсів прикладного програмування). Відкриті API забезпечують ефективну взаємодію між різним програмним забезпеченням та автоматизують обробку і передачу даних між системами. Інтеграція таких API з іншими сервісами та базами даних забезпечує швидкість обміну інформацією, актуальність даних та значно знижує ймовірність помилок, що є важливими факторами в таких сферах, як фінанси, охорона здоров'я та адміністративні послуги. Інтеграція з базами даних не тільки автоматизує обробку інформації, але й забезпечує високий рівень безпеки даних, що є однією з основних вимог до таких систем. Дійсно, в умовах постійно зростаючих ризиків забезпечення безпеки персональних даних та захист їх від несанкціонованого доступу стає ключовим викликом. На цьому тлі була запропонована концепція веб-системи, яка інтегрує зовнішні сервіси через API, контролює доставку привілейованих товарів та ефективно управляє доступом до них. Розроблена система спрямована на підвищення ефективності, точності та прозорості процесу надання пріоритетних послуг, при цьому значно спрощуючи адміністрування та забезпечуючи високий рівень безпеки даних. Система відповідає сучасним вимогам автоматизації та інтеграції технологій, що дозволяє організаціям скоротити адміністративні витрати і значно підвищити якість послуг, що надаються. Метою цієї статті є детальне вивчення технічних аспектів розробки такої системи, її архітектури, принципів обробки даних та безпеки. Вона також має на

меті проаналізувати сучасні підходи до інтеграції відкритих API та баз даних і надати рекомендації щодо впровадження подібних рішень в інших галузях, які потребують швидкої, ефективної та безпечної обробки даних.

1 АНАЛІЗ ІНТЕГРАЦІЇ API ТА БАЗ ДАНИХ ДЛЯ АВТОМАТИЗАЦІЇ

1.1 Аналіз проблем і потреб автоматизації процесу видачі продукції

У багатьох сферах, де громадяни мають право на товари та послуги, існують серйозні проблеми, пов'язані з ручним введенням даних. Цей процес часто передбачає ручну перевірку документів користувачів, внесення інформації до місцевих баз даних і систем обліку та затвердження заявок на отримання продуктів. Основною проблемою цього методу є висока ймовірність людської помилки, коли неточності у введенні даних, наприклад, неправильний ІПН або інші ключові деталі, можуть призвести до відмови у наданні продуктів та надмірного споживання ресурсів. Іншим наслідком є час, необхідний для обробки запитів, оскільки ручна обробка займає набагато більше часу, ніж автоматизована. Крім того, відсутність єдиної системи обліку перешкоджає повному контролю за видачею інструментів та залишками коштів, що призводить до непрозорості всього процесу. Наприклад, у багатьох державних установах дані про права користувачів можуть бути застарілими на момент подання заявки, що ще більше ускладнює своєчасну перевірку права на отримання допомоги.

Існуючі інформаційні системи реєстрації категорій населення зазвичай є закритими і мають багато обмежень у функціонуванні. Інтеграція з іншими системами часто обмежена, а взаємодія з національними реєстрами та базами даних неможлива, що призводить до затримок в оновленні інформації. Крім того, такі системи складні в управлінні і характеризуються значними зусиллями і часом, необхідними для додавання нових користувачів або зміни їхнього статусу. Неможливість працювати в режимі реального часу ще більше ускладнює своєчасну перевірку прав користувачів, що має вирішальне значення для цього процесу. Наприклад, дані про права не оновлюються автоматично, що призводить до видачі продуктів людям, які вже мають на них право, або до відмови тим, хто щойно набув такого

права. Крім того, системи обліку, що використовуються різними організаціями, часто не уніфіковані, що ускладнює інтеграцію і є бар'єром для обміну даними між різними установами.

У багатьох випадках інформація зберігається локально без доступу до централізованої бази даних, що суттєво обмежує прозорість і контроль. Відсутність централізованого контролю також створює умови для потенційного шахрайства, оскільки одна й та сама особа може отримувати продукти багато разів у різних місцях. Така практика не лише знижує ефективність процесу, але й спричиняє додаткові фінансові втрати для організації, що надає пільги. Для вирішення вищезазначених проблем необхідно створити веб-систему, яка б автоматизувала ключові етапи процесу надання пільгових продуктів. Така система має відповідати найсучаснішим вимогам щодо швидкості та точності співставлення даних, а також гарантувати інтеграцію з національними реєстрами через відкриті API. Це дає змогу перевіряти інформацію про користувача в режимі реального часу та значно знижує ризик помилок і затримок. Важливою перевагою використання відкритих API є те, що вони можуть взаємодіяти з різними джерелами даних, включаючи державні бази даних, що містять інформацію про соціальний статус користувачів. Це дозволяє уникнути дублювання даних і забезпечує актуальність інформації.

Простота адміністрування також важлива, оскільки дозволяє системним адміністраторам швидко додавати, змінювати або видаляти користувачів і керувати правами доступу. Прозорість обліку - ще один важливий фактор. Всі дії в системі повинні реєструватися, щоб можна було уникнути шахрайства і повністю контролювати процес дистрибуції продукції. Ще одним важливим елементом є захист даних. Системи повинні відповідати сучасним вимогам інформаційної безпеки, таким як шифрування даних і багаторівнева автентифікація користувачів. Це особливо важливо, коли персональні дані є конфіденційними і потребують захисту від несанкціонованого доступу. Запропонована система також повинна мати гнучку архітектуру, яка може адаптуватися до різних потреб організації. Наприклад, залежно від специфікацій організації, можна підтримувати різні методи автентифікації ко-

ристувачів, такі як використання QR-кодів, смарт-карт або біометричних даних. Також важливо забезпечити мобільний доступ до системи, що дозволить користувачам отримувати доступ до системи через смартфони та інші портативні пристрої. Все це покращить зручність використання системи як для користувачів, так і для адміністраторів. Таким чином, аналіз проблеми показав необхідність автоматизації процесу пропозиції продуктів шляхом інтеграції відкритих API та баз даних.

Запропонована веб-система дозволить усунути недоліки ручного введення даних, прискорити їх перевірку, підвищити прозорість обліку та забезпечити сучасний рівень інформаційної безпеки. Універсальність та адаптивність такої системи робить її придатною для використання в різних організаціях і забезпечує ефективність та прозорість процесу надання пропозицій.

1.2 Використання відкритих API для інтеграції даних у веб-системах.

У сучасному суспільстві автоматизація даних є однією з основних складових ефективних бізнес-процесів, в яких важливу роль відіграють відкриті API. Відкриті API дозволяють інтегрувати різні системи, бази даних і зовнішні сервіси в єдину екосистему, значно спрощуючи взаємодію між різними компонентами веб-системи. Використання відкритих API забезпечує стандартизований та уніфікований доступ до різних ресурсів, значно скорочуючи час обробки даних і дозволяючи автоматизувати багато процесів Завдяки API немає необхідності вручну обробляти запити або виконувати складну конфігурацію для підключення до окремих сервісів Однією з головних переваг API є можливість доступу до даних у режимі реального часу, що особливо важливо для автоматизації процесів, які потребують постійного оновлення інформації. Це особливо актуально для великих інформаційних систем, яким необхідно швидко і точно обробляти великі обсяги даних, наприклад, інформацію про продукти, клієнтів і запити користувачів і т.д. Оскільки вся інформація обробляється автоматично і миттєво передається між системами,

API дозволяють істотно знизити ймовірність ручного введення даних значно зменшується кількість помилок, які можуть виникнути при ручному введенні даних. API також забезпечують автоматичну синхронізацію даних, що гарантує їхню актуальність у режимі реального часу, що важливо для забезпечення високої точності та ефективності веб-систем. Ще однією важливою особливістю використання відкритих API є можливість інтеграції з різноманітними зовнішніми сервісами, що дозволяє значно розширити функціональність системи: API дозволяють взаємодіяти з платіжними системами, сервісами перевірки даних, службами доставки та іншими інструментами, необхідними для ефективної роботи бізнес-процесів. API дозволяє взаємодіяти з інструментами, необхідними для ефективної роботи бізнес-процесів, такими як платіжні системи, сервіси валідації даних і служби доставки. Це дозволяє створювати більш складні та потужні рішення, які об'єднують різні елементи інфраструктури та надають користувачам зручний інтерфейс для роботи з ними. Наприклад, якщо вам потрібно перевірити статус замовлення або наявність певного товару в режимі реального часу, відкритий

API надає прямий доступ до цих даних без необхідності ручного втручання. Вони також допомагають забезпечити високий рівень безпеки, оскільки можуть стандартизувати способи авторизації та автентифікації користувачів і зовнішніх систем. Це дозволяє створювати ефективні механізми контролю доступу, які обмежують можливості зловмисників і зловмисників. Відкриті API можуть включати такі механізми безпеки, як OAuth, шифрування TLS/SSL, ключі API та токени доступу, що забезпечує високий рівень захисту даних на всіх етапах їх обробки. Оскільки дані часто передаються між різними системами, важливо забезпечити їхню конфіденційність і цілісність, а API забезпечують цей процес за допомогою стандартизованих протоколів безпеки. Крім того, API можуть спростити процес обслуговування та модернізації системи: оскільки API забезпечують чітку структуру взаємодії між компонентами, зміни в одному елементі системи можуть бути внесені без необхідності змінювати всю інфраструктуру. Наприклад, якщо система має нові вимоги до інтеграції з іншими сервісами або потрібно додати нове джерело даних, API можна просто розширити або додати нову кінцеву точку, не втручаючись в

основний код. Це дозволяє розширювати систему та адаптувати її до нових потреб без значних витрат часу та ресурсів.

Технологія відкритих API забезпечує гнучкість і адаптивність системи, дозволяючи їй швидко реагувати на зміни в бізнес-процесах і вимоги зовнішніх користувачів; інтеграція з базами даних через API також є важливим аспектом, оскільки дозволяє централізовано зберігати і обробляти дані. Замість того, щоб кожна система мала власну окрему базу даних і вручну переносила дані між ними, API можна використовувати для створення єдиної точки доступу до всіх необхідних ресурсів. Це спрощує управління базами даних і дозволяє гнучкіше керувати правами доступу:

API може автоматично надавати доступ до потрібних даних на основі ролей користувачів та інших параметрів, що значно підвищує ефективність управління доступом до ресурсів. Завдяки всім цим перевагам відкриті API є потужним інструментом для інтеграції даних та автоматизації бізнес-процесів. Вони зменшують кількість помилок, скорочують час обробки даних і роблять процеси більш прозорими та керованими. Використання API є важливою частиною стратегії цифрової трансформації, забезпечуючи ефективне використання ресурсів і дозволяючи компаніям швидко реагувати на ринкові зміни та надавати користувачам кращі послуги.

1.3 Особливості інтеграції баз даних із зовнішніми сервісами через API.

Інтеграція баз даних із зовнішніми сервісами через API є ключовим елементом забезпечення ефективності сучасних веб-систем. Бази даних традиційно є основним сховищем інформації в більшості організацій, оскільки вони можуть зберігати, категоризувати та обробляти великі обсяги даних. Однак для забезпечення більшої гнучкості, масштабованості та інтерактивності бази даних потребують інтеграції з іншими системами, в тому числі із зовнішніми сервісами. Відкриті API

відіграють важливу роль у цьому процесі, оскільки вони забезпечують безпечний та ефективний обмін інформацією між різними програмними компонентами та гарантують синхронізацію даних у реальному часі. Однією з головних переваг використання API для інтеграції з базами даних є автоматична синхронізація даних між внутрішніми системами та зовнішніми сервісами. Замість того, щоб вручну передавати або імпортувати дані між різними джерелами, API автоматизують цей процес, зменшуючи ймовірність помилок і значно підвищуючи швидкість обробки інформації. Наприклад, API можуть автоматично оновлювати інформацію в базі даних щоразу, коли нові дані стають доступними із зовнішніх джерел, таких як постачальники товарів, платіжні сервіси та інші зовнішні ресурси, допомагаючи підтримувати точність і актуальність даних у режимі реального часу. за допомогою API Інтеграція баз даних також дозволяє виконувати більш складні операції з даними без ручного втручання в процес обробки. Наприклад, за допомогою API можна робити запити до бази даних для отримання необхідної інформації, обробляти її та передавати в інші системи або зовнішні сервіси без зайвих затримок. Це дозволяє створювати більш складну бізнес-логіку, наприклад, автоматизовану доставку продукції, управління складом, перевірку доступу користувачів та обробку замовлень.

Важливо, що API дозволяє виконувати ці операції швидко і безпечно, завдяки використанню стандартів і протоколів, які регулюють доступ до даних і гарантують їх захист Ще однією важливою перевагою інтеграції баз даних за допомогою API є можливість розширювати і адаптувати систему, забезпечують зв'язок між системами без значних змін у внутрішній архітектурі, дозволяючи розширювати систему, додаючи нові зовнішні сервіси або змінюючи джерела даних, не витрачаючи при цьому багато часу і ресурсів. Наприклад, можна підключити нові зовнішні сервіси для аутентифікації користувачів або додати нові платіжні шлюзи без зміни структури бази даних або базової логіки веб-системи. Це особливо важливо для організацій, які працюють у швидкозмінному середовищі і повинні бути готовими до інтеграції з новими сервісами та джерелами даних. інтеграція баз даних із зовнішніми сервісами через API також важлива з точки зору безпеки даних. за допомогою API ви можете використовувати OAuth, API-ключі, токени, SSL-з'єднання токени,

SSL-з'єднання та різні інші методи автентифікації та авторизації, щоб гарантувати, що тільки авторизовані користувачі та системи можуть отримати доступ до даних. Це мінімізує ризик несанкціонованого доступу до конфіденційної інформації та забезпечує цілісність даних під час передачі даних між різними системами.

Оскільки багато зовнішніх сервісів обробляють конфіденційну інформацію, наприклад, фінансові або медичні дані, дуже важливо забезпечити належний рівень безпеки під час інтеграції. Крім того, інтеграція через API може значно спростити процес моніторингу та обслуговування систем: API надають стандартизований спосіб відстеження виконання запитів і можливість аудиту взаємодії між системами. Це дозволяє швидко виявляти потенційні проблеми в системі, такі як збій в обміні даними або збої в роботі зовнішніх сервісів, і вирішувати їх швидко і без затримок. Такі механізми зворотного зв'язку можуть значно підвищити стабільність і надійність всієї інфраструктури.

Таким чином, інтеграція баз даних із зовнішніми сервісами через API є важливим елементом сучасних автоматизованих систем. Це дозволяє швидко, безпечно та ефективно обмінюватися даними, інтегрувати різні джерела інформації, розширювати систему та зменшувати ризики, пов'язані з ручним введенням та передачею даних. Така інтеграція дозволяє організаціям швидко реагувати на нові вимоги ринку, покращувати якість обслуговування та зменшувати витрати на обробку даних.

1.4 Аналіз стандартів та протоколів взаємодії з відкритими API

Аналіз стандартів і протоколів взаємодії з відкритими API є важливою частиною будь-якої веб-системи, яка використовує ці технології для інтеграції з іншими системами та зовнішніми сервісами. Протоколи взаємодії та стандарти API визначають правила передачі даних між системами і забезпечують узгодженість, безпеку та ефективність обміну інформацією. Оскільки відкриті API забезпечують зв'язок між різними технологічними стеками та інфраструктурами, важливо розуміти, які

стандарти та протоколи використовуються для забезпечення коректної та безпечної роботи цих API. Одним з основних стандартів, що визначає структуру запитів і відповідей між клієнтом і сервером у відкритих API, є Representational State Transfer (REST) REST базується на принципах простоти, масштабованості та гнучкості для розробки веб-сервісів. REST є одним з найпопулярніших підходів до створення API, оскільки він простий в реалізації та ефективний при роботі з великими обсягами даних,

REST API ідеально підходять для інтеграції з різноманітними зовнішніми системами та базами даних, оскільки вони можуть легко адаптуватися до змін. Іншим важливим стандартом є GraphQL, альтернатива REST, яка набирає популярності, оскільки зменшує кількість запитів до сервера і дозволяє тонко налаштовувати обробку запитів для отримання лише необхідних даних. Клієнт має змогу самостійно визначати дані запиту і може визначати поля та об'єкти, необхідні в запиті. Це значно зменшує навантаження на сервер і обсяг даних, що передаються. Це дуже важливо при роботі з великими базами даних або в ситуаціях, коли ресурси обмежені. Для деяких веб-систем, що мають справу зі складними даними, GraphQL може бути більш ефективним варіантом, ніж традиційний REST Ще одним важливим аспектом взаємодії з API є використання для передачі даних стандарту JSON (JavaScript Object Notation), JSON - це легкий текстовий формат для зберігання і передачі структурованих даних і є основним форматом для обміну інформацією між клієнтом і сервером в більшості сучасних веб-систем. мовами програмування, що робить його універсальним вибором в якості API.

Завдяки цим характеристикам JSON став стандартом де-факто для більшості API, включаючи RESTful API, і основою для серіалізації даних, що надсилаються через мережу. Ще одним важливим аспектом взаємодії з відкритими API є використання протоколу OAuth (Open Authorisation), який забезпечує безпечний доступ до API OAuth - це стандарт аутентифікації та авторизації, який дозволяє стороннім додаткам використовувати ім'я користувача та пароль від Він дозволяє стороннім додаткам мати обмежений доступ до ресурсів користувача без необхідності вказу-

вати ім'я користувача та пароль. Замість цього користувач надає дозвіл додатку через спеціальний механізм авторизації, а додаток отримує токен доступу, який може бути використаний для запитів до API. OAuth значно підвищує рівень безпеки при роботі з конфіденційною інформацією та дозволяє використовувати різні Інтеграція між сервісами

Протокол HTTPS - ще один важливий аспект взаємодії з відкритими API. Він забезпечує безпечне з'єднання між клієнтом і сервером за допомогою шифрування SSL/TLS і гарантує конфіденційність і цілісність даних, що передаються. Оскільки відкриті API часто передають конфіденційну інформацію, таку як дані користувачів або фінансові транзакції, використання HTTPS є необхідним для захисту від таких атак, як man-in-the-middle (MITM), що дозволяє зловмисникам перехоплювати і модифікувати дані, які передаються між клієнтом і сервером. Не менш важливим є використання стандартів для управління помилками та несправностями API. Важливо, щоб API не тільки правильно обробляв запит, але й коректно повідомляв про будь-які помилки або збої, які можуть виникнути під час взаємодії. Для цього розробники API використовують спеціальні HTTP-коди статусу, які дають чітке розуміння того, що відбувається із запитом. Наприклад, код 200 означає, що запит був успішно оброблений, 400 вказує на помилку на стороні клієнта, а 500 - на помилку на стороні сервера. Крім того, API повинні надавати чіткі повідомлення про помилки в зрозумілому форматі, щоб проблеми можна було швидко діагностувати і виправити; інтеграція API з різними сервісами також вимагає чітко визначених політик доступу та обмежень. Для цього використовуються протоколи, що обмежують кількість запитів до API протягом певного періоду часу (наприклад, ліміти швидкості API). Ці ліміти запобігають перевантаженню сервера і забезпечують рівний доступ до ресурсів для всіх користувачів. Наприклад, протокол RESTful API дозволяє встановлювати ліміти на кількість запитів від одного користувача або на кількість запитів для певної операції, що дозволяє уникнути зловживань і підтримувати стабільну роботу системи навіть при високих навантаженнях.

Таким чином, стандарти і протоколи взаємодії з відкритими API мають вирішальне значення для забезпечення ефективності, безпеки і масштабованості

веб-систем: такі стандарти, як REST, GraphQL, JSON, OAuth і HTTPS, а також відповідне управління помилками і обмеження доступу. Їх використання дозволяє створювати надійні, безпечні та ефективні API, здатні інтегрувати розрізнені системи і забезпечувати якісний обмін даними між клієнтами і серверами. Суворе дотримання цих стандартів гарантує, що взаємодія між системами буде стабільною, надійною та безпечною.

1.5 Переваги та недоліки використання API в автоматизації бізнес-процесів

Хоча використання відкритих API для автоматизації бізнес-процесів має багато переваг, існують також проблеми, які необхідно враховувати при розробці та інтеграції систем. Однією з головних переваг є можливість спростити взаємодію між різними системами та ефективно інтегруватися з іншими платформами. Завдяки стандартизованому доступу до даних, відкриті API дозволяють компаніям швидко обмінюватися інформацією із зовнішніми сервісами, такими як платіжні системи, служби доставки та верифікації даних користувачів, без необхідності створювати власні інтерфейси та інтеграції. Ще однією важливою перевагою є гнучкість, яку надає використання API, що дозволяє розширювати систему та адаптувати її до нових вимог без необхідності змінювати весь програмний код. Наприклад, якщо додається нове джерело даних або змінюється логіка обробки запитів, можна модифікувати та впроваджувати лише частину системи, не впливаючи на інші компоненти. Це значно зменшує витрати на обслуговування та підвищує адаптивність.

Однак використання відкритих API має свої недоліки. Одна з найбільших проблем - безпека. Відкриті API збільшують кількість точок доступу до системи, що підвищує потенціал для атак; якщо API не захищений належним чином, існує ризик витоку даних або несанкціонованого доступу до критично важливої інфор-

мації. Щоб мінімізувати ці ризики, слід використовувати сучасні методи шифрування, автентифікації та авторизації, такі як OAuth, SSL/TLS та інші протоколи безпеки. Ще один аспект, який слід враховувати, - це залежність від зовнішніх сервісів та партнерів по API. Якщо зовнішні сервіси мають проблеми з доступністю або стабільністю, це може вплинути на роботу системи. Тому важливо передбачити резервування та механізми обробки помилок, щоб система могла ефективно працювати в разі тимчасових збоїв. Загалом, використання відкритих API може значно підвищити ефективність автоматизації процесів, але важливо ретельно планувати архітектуру таких систем і враховувати можливі ризики та способи їх мінімізації.

1.6 Взаємодія між різними типами API та їх вибір залежно від завдань

Вибір типу API для взаємодії з іншими системами залежить від конкретних вимог проекту, типу даних, що обробляються, та розміру інфраструктури. Найпоширенішими типами API є REST, SOAP і GraphQL. Кожен з них має свої особливості, переваги та недоліки, які визначають його потенційне використання в різних сценаріях автоматизації бізнес-процесів REST (Representational State Transfer) - найпоширеніший тип API веб-сервісів. В його основі лежать принципи простоти, масштабованості та гнучкості: REST API маніпулює ресурсами за допомогою стандартних методів HTTP (GET, POST, PUT, DELETE). Основними перевагами REST є його універсальність і простота реалізації, а також те, що REST API можуть бути Легка інтеграція з різними платформами і мовами програмування: простий доступ до даних, таких як JSON і XML, робить його зручним для інтеграції з веб-додатками і мобільними пристроями REST - це Це найкращий вибір. Однак, коли ви маєте справу з великими обсягами складних або взаємопов'язаних даних, REST може бути неефективним, оскільки для отримання всіх необхідних даних потрібно кілька запитів Простий протокол доступу до об'єктів (SOAP) - це, як правило, протокол, заснований на XML, використовується для більш складної та формальної інтеграції; SOAP забезпечує суворі правила для структурування запитів та відповідей і

підтримує більш складні механізми безпеки та транзакцій; SOAP API зазвичай використовуються у високозахищених системах, де важливими є безпека, точність та надійність.

Цей тип API дуже корисний у фінансових, медичних та юридичних системах, де важливо забезпечити гарантовану доставку повідомлень, інтеграцію з іншими бізнес-сервісами та відповідність стандартам. Однак SOAP API складніші, вимагають більше ресурсів для налаштування та підтримки і передають більше даних, ніж REST, що може бути недоліком при роботі з легкими мобільними додатками або високонавантаженими веб-сервісами. і є відносно новим підходом до створення API, який дозволяє клієнтам визначати тип даних, які їм потрібні. На відміну від REST, де кожен запит API може повернути більше даних, ніж потрібно, GraphQL дозволяє запитувати тільки необхідні поля, що значно зменшує обсяг переданих даних і навантаження на сервер. Це робить GraphQL ідеальним для роботи з великими масивами даних і складними структурами даних, такими як інтерфейси мобільних додатків, де важливо мінімізувати обсяг переданих даних. GraphQL також дуже ефективна для інтеграції з багатьма зовнішніми джерелами даних і мікросервісами, оскільки вона може запитувати кілька ресурсів в одному запиті.

Однак цей підхід може бути складним у налаштуванні та реалізації, особливо якщо ви хочете зберегти простоту і сумісність з іншими традиційними API; вибір між REST, SOAP або GraphQL залежить від багатьох факторів, включаючи складність запиту, вимоги до безпеки, обсяг і швидкість передачі даних. REST залишається найбільш універсальним і простим у використанні для більшості веб-додатків, але SOAP може бути кращим вибором для більш складних і критичних систем, які вимагають високої надійності і точності. GraphQL, з іншого боку, є найкращим вибором для складних запитів з великими обсягами даних, де важлива оптимізація запитів і зниження навантаження на мережу і сервер. Для кожного конкретного випадку слід детально проаналізувати системні вимоги і вибрати тип API, який найкраще підходить для досягнення поставлених цілей.

1.7 Використання фреймворків аутентифікації при роботі з API

Одним з важливих аспектів забезпечення безпеки систем, що використовують відкриті API, є механізм автентифікації, який гарантує, що тільки авторизовані користувачі або системи можуть отримати доступ до ресурсів і виконувати певні операції. Для реалізації надійної автентифікації в таких системах використовуються різні фреймворки автентифікації, які допомагають організувати цей процес з високим рівнем безпеки та зручності. Залежно від вимог безпеки та специфікацій API-додатків, різні фреймворки можуть бути більш або менш придатними. У сучасних веб-системах для автентифікації API найчастіше використовуються стандартизовані протоколи та фреймворки, такі як OAuth 2.0, OpenID Connect та JWT (JSON Web Tokens).

OAuth 2.0 - найпоширеніший протокол для автентифікації та авторизації у веб-додатках та API. Його основна мета - делегування доступу до ресурсів без надання автентифікаційної інформації (логіна та пароля); OAuth 2.0 дозволяє додаткам отримувати доступ до ресурсів користувача в іншій системі без необхідності передавати автентифікаційну інформацію (наприклад, електронну пошту або наприклад, доступ до файлів на Google Диску). Протокол працює через механізм токенів доступу, які видаються після того, як користувач був авторизований і отримав дозвіл на доступ до ресурсу Основні особливості OAuth 2.0 полягають у тому, що він підтримує кілька типів потоків, з яких користувачі можуть вибрати потік, який найкраще підходить для конкретного сценарію: аутентифікація Потік коду - використовується для веб-додатків, де користувач входить через веб-браузер. Якщо користувач успішно пройшов автентифікацію, сервер автентифікації видає код автентифікації, який потім використовується для отримання токена доступу. Клієнтський потік автентифікації - підходить для серверних додатків і мікросервісів, які керують обома сторонами (клієнтом і сервером) і не вимагають участі кінцевого користувача. Неявний потік - використовується для одноразової автентифікації у веб-

браузерах, таких як мобільні додатки та односторінкові додатки (SPA), де безпечне зберігання токенів доступу ускладнене.

Основною перевагою OAuth 2.0 є його гнучкість і можливість налаштування для різних типів клієнтів і сценаріїв використання, при цьому він забезпечує високий рівень безпеки завдяки механізмам оновлення токенів і обмеженню доступу до ресурсів за часом або обсягом.

OpenID Connect (OIDC) - це протокол, побудований на основі OAuth 2.0, але призначений для автентифікації та авторизації користувачів. Іншими словами, OpenID Connect дозволяє користувачам не тільки отримувати доступ до ресурсів, але й підтверджувати свою особу; OAuth 2.0 додає стандартний набір функцій автентифікації, таких як перехоплення та перевірка інформації про користувача (наприклад, ідентифікатор користувача та електронну пошту). OpenID Connect зазвичай використовується в ситуаціях, коли необхідно перевірити, що певний запит до API надійшов від авторизованого користувача. Протокол також полегшує процес єдиного входу (SSO) для великої кількості веб-додатків і систем, оскільки ідентичність користувача може бути перевірена за допомогою токена ідентичності. Завдяки інтеграції з OAuth 2.0, OpenID Connect підтримує всі його переваги і в той же час надає додаткову функціональність для точної ідентифікації користувача. З цієї причини OpenID Connect особливо корисний в системах, де важливо централізувати аутентифікацію декількох додатків і сервісів.SON Web Tokens (JWT) - це компактний, безпечний і зручний формат для передачі інформації між сторонами у вигляді токенів. JWT часто використовується як метод аутентифікації для систем, що взаємодіють через відкриті API, в тому числі в поєднанні з OAuth 2.0 Токени JWT складаються з трьох частин: заголовка, корисного навантаження і підпису, що полегшує передачу через HTTP-запити JWT дозволяє токену зберігати аутентифікаційні дані (наприклад, ідентифікатор користувача), а також додаткову інформацію про права доступу, термін дії токена та інші параметри. Цей механізм забезпечує безпеку за допомогою криптографічних підписів, які можуть перевірити цілісність даних і запобігти їх модифікації під час передачі.JWT дозволяє генерувати токен один раз і використовувати його для декількох запитів до закінчення терміну дії,

що корисно для декількох серверів і архітектури мікросервісів для запитів. Це корисно для обробки запитів на декількох серверах або в мікросервісній архітектурі. Це зменшує навантаження на сервер, усуваючи необхідність щоразу автентифікувати користувачів.

Таким чином, вибір фреймворку для аутентифікації є ключовим етапом у забезпеченні безпеки та ефективності взаємодії між системами через API. Залежно від потреб організації, можна вибирати між різними стандартами, такими як OAuth 2.0, OpenID Connect або JWT, що дозволяє задовольнити вимоги до безпеки, масштабованості і зручності інтеграції.

1.8 Управління сесіями та контроль доступу в відкритих API

Одним з найважливіших аспектів роботи з відкритими API є правильне управління сесіями та ефективний контроль доступу, оскільки ці процеси безпосередньо впливають на безпеку системи та точність обробки запитів. Управління сеансами означає контроль того, які користувачі та системи можуть отримати доступ до певних ресурсів під час сеансу в рамках взаємодії з API. Важливо, щоб сеанси були належним чином захищені та обмежені, щоб мінімізувати ризик несанкціонованого доступу. Контроль доступу - це процес обмеження доступу до ресурсів на основі дозволів користувача або клієнта, які зазвичай визначаються автентифікацією та авторизацією. Управління сеансами в контексті відкритих API передбачає зберігання та перевірку стану сеансу користувача через взаємодію з сервером API.

Це гарантує, що користувач правильно ідентифікований і що права доступу до певних ресурсів у сесії підтримуються. Одним з найпоширеніших методів управління сесіями є використання маркерів доступу, таких як JSON Web Tokens (JWT) або маркери доступу OAuth 2.0. Токени доступу мають обмежений термін дії і зберігають інформацію про особу та права користувача. При використанні токена доступу кожен запит до API повинен супроводжуватися дійсним токеном в

заголовку HTTP-запиту або параметрі URL-адреси. Сервер перевіряє дійсність токена перед обробкою запиту. Це дозволяє постачальникам послуг надавати безпечний доступ до ресурсів і гарантувати, що кожен запит здійснюється в межах повноважень користувача.

Для забезпечення більшої безпеки багато систем обирають стратегію короткострокових токенів, термін дії яких автоматично закінчується через певний проміжок часу або після виконання певної дії. Це обмежує можливість використання старого токена після закінчення сеансу або в разі підозрілої дії. При необхідності сервер може видати новий токен, щоб оновити токен і продовжити сеанс без повторної аутентифікації користувача. Іншим важливим аспектом є підтримка сесій, які охоплюють кілька пристроїв і сеансів. У багатьох сучасних системах користувачі можуть підключатися через різні пристрої (наприклад, мобільні телефони, ноутбуки, планшети).

Це вимагає підтримки сесій в рамках одного облікового запису на різних пристроях і додаткових механізмів безпеки, таких як двофакторна автентифікація (2FA) і автентифікація на основі місцезнаходження.

Контроль доступу визначає, хто і які ресурси може використовувати в системі. Це є важливим аспектом забезпечення безпеки і правильного функціонування будь-якої системи, яка працює з відкритими API. Основними типами контролю доступу є:

1. Мандатний контроль доступу (Mandatory Access Control, MAC) — це жорсткий контроль доступу, при якому користувачі та процеси можуть доступати тільки ті ресурси, до яких вони явно мають доступ згідно з політиками безпеки, що встановлені адміністраторами. Зазвичай використовується в системах з високими вимогами до безпеки, таких як військові або урядові інституції.
2. Рольовий контроль доступу (Role-Based Access Control, RBAC) — один із найпоширеніших методів, коли права доступу до ресурсів визначаються на основі ролей користувачів. Наприклад, адміністратор

може мати повний доступ до всіх ресурсів, тоді як звичайний користувач може мати лише доступ до певних даних. RBAC дозволяє централізовано управляти доступом до ресурсів на основі ролей, що полегшує адміністрування та знижує ймовірність помилок.

3. Атрибутивний контроль доступу (Attribute-Based Access Control, ABAC) — підхід, який визначає доступ до ресурсів на основі атрибутів користувача або системи. Цей метод є більш гнучким порівняно з RBAC, оскільки доступ можна обмежити не тільки на основі ролей, а й на основі конкретних атрибутів, таких як місце перебування користувача, час доби, тип пристрою або інші параметри.
4. Контроль доступу на основі політик (Policy-Based Access Control, PBAC) — цей підхід дозволяє більш детально налаштовувати доступ до ресурсів на основі політик, які визначаються адміністраторами. Вони можуть бути як централізованими, так і дистрибутивними, і дозволяють організаціям створювати дуже детальні правила доступу до різних типів даних.

Найпоширеніший спосіб гарантувати безпечний доступ до API - це використання токенів автентифікації, таких як OAuth 2.0 або JWT. Ці токени дозволяють чітко визначити права доступу для кожного запиту і обмежити доступ лише до тих ресурсів, до яких користувач має право доступу. Токени можуть мати різні рівні доступу залежно від ролі користувача і типу запиту, що дозволяє гнучко керувати доступом. Крім того, багато сучасних систем API використовують механізми реєстрації та аудиту сеансів для запису всіх дій користувачів і автоматичного сповіщення адміністраторів про підозрілі або несанкціоновані запити. Це дозволяє швидко виявляти порушення безпеки та спроби несанкціонованого доступу до ресурсів.

Управління сесіями та контроль доступу в системах, що використовують відкриті API, є ключовими елементами для забезпечення безпеки даних і надійності роботи системи. Використання ефективних методів аутентифікації, таких як токени

доступу, а також впровадження політик контролю доступу, дозволяє точно визначати, хто має доступ до яких ресурсів. Системи, що забезпечують високий рівень безпеки сесій, можуть ефективно захистити дані від несанкціонованого доступу і знизити ризики порушення цілісності даних у процесі роботи з відкритими API.

1.9 Вимоги до масштабованості та продуктивності при інтеграції з API

Масштабованість і продуктивність є критично важливими факторами при проектуванні систем, що використовують відкриті API. Оскільки кількість даних і запитів, що обробляються системою, продовжує зростати, важливо забезпечити ефективну і стабільну роботу навіть при високих навантаженнях.масштабованість API означає здатність системи адаптуватися до збільшення обсягів користувачів, даних і запитів без шкоди для продуктивності або доступності сервісів означає здатність системи адаптуватися до збільшення кількості користувачів, даних і запитів без шкоди для продуктивності або доступності сервісів. У зв'язку з цим архітектура API та серверна інфраструктура повинні бути належним чином сплановані, щоб забезпечити високий рівень обслуговування в умовах збільшення трафіку.

Для забезпечення масштабованості відкритих API важливо враховувати кілька ключових аспектів. По-перше, API повинен бути розроблений таким чином, щоб підтримувати горизонтальне масштабування. Це дозволяє додавати нові сервери для обробки запитів у міру зростання навантаження. Це включає в себе використання балансувальників навантаження, які можуть ефективно розподіляти запити між декількома серверами. По-друге, важливо застосовувати кешування на різних рівнях, як на стороні сервера, так і на стороні клієнта, щоб зменшити кількість запитів до бази даних і скоротити час відгуку API. використання технологій кешування, таких як Redis і Memcached, може значно зменшити навантаження на сервер і пришвидшити доступ до даних, що часто запитуються. Іншим важливим

аспектом є продуктивність API, яка визначає, наскільки швидко система обробляє запити.

Для досягнення високої продуктивності необхідно оптимізувати обробку запитів, щоб зменшити затримки і гарантувати швидке виконання навіть при високому навантаженні. Цього можна досягти за допомогою різних методів, таких як асинхронні запити, паралельна обробка запитів і механізми пакетної обробки, які можуть обробляти запити до декількох ресурсів одночасно. Крім того, важливо мінімізувати складність запиту і використовувати легкі формати передачі даних, такі як JSON і буфери протоколу, щоб скоротити час серіалізації та десеріалізації. При інтеграції з зовнішніми сервісами або базами даних особливо важливо враховувати продуктивність API. Запити до зовнішніх систем часто мають тривалий час затримки, що впливає на загальну продуктивність. Для вирішення цієї проблеми можна використовувати паралельне виконання запитів або черги повідомлень. Використовуючи черги повідомлень, запити можна зберігати для обробки і виконувати їх по черзі, не блокуючи основну операційну систему. Використання індексів, правильна денормалізація таблиць та ефективні запити можуть значно підвищити швидкість обробки даних. Іншим важливим фактором є надійність API. Сюди входить здатність системи продовжувати роботу в разі збою або виходу з ладу окремого компонента. З цією метою часто використовуються методи аварійного відновлення та резервне копіювання даних для швидкого відновлення доступності API.

Крім того, необхідно впровадити механізми моніторингу та оповіщення для своєчасного виявлення можливих проблем з продуктивністю або доступністю API, щоб скоротити час аварійного відновлення. також важливо враховувати сумісність API та вимоги до його взаємодії з іншими сервісами. Для цього API повинні бути сумісними з поширеними протоколами і стандартами (наприклад, HTTP, REST, GraphQL, WebSocket) і мати чітко визначену документацію для полегшення інтеграції з іншими системами. Крім того, відкриті API повинні мати можливість працювати з різними типами клієнтів, такими як мобільні додатки, веб-додатки та інші мікросервіси. Ці елементи разом забезпечують високий рівень продуктивності та масштабованості відкритого API.

Це має вирішальне значення для систем, які повинні обробляти велику кількість запитів і взаємодіяти з великою кількістю користувачів і сервісів. Системи, які можуть ефективно масштабуватися і швидко обробляти запити, забезпечують високий рівень користувацького досвіду.

1.10 Тестування та забезпечення якості при роботі з відкритими API

Одним з перших кроків є функціональне тестування API, щоб перевірити, чи відповідає API вимогам бізнес-логіки. Це тестування може визначити, чи правильно API обробляє запити, чи повертає правильні дані і чи є помилки в обробці запитів. Для такого тестування можна використовувати автоматизовані інструменти, такі як Postman, Swagger і SoapUI, які створюють і запускають тести для різних кінцевих точок API, перевіряють відповіді на різні вхідні дані і генерують звіти про помилки. Також важливо виконувати тести продуктивності, щоб переконатися, що API може обробляти велику кількість запитів за короткий час. Для цього використовуються інструменти стрес- і навантажувального тестування, такі як JMeter, LoadRunner і Gatling. Це дозволяє з'ясувати, чи може API витримувати пікові навантаження, чи може система коректно реагувати на високий трафік і чи виникають якісь проблеми, такі як затримки або помилки при обробці запитів під високим навантаженням.

Тестування продуктивності дозволяє виявити потенційні вузькі місця і оптимізувати систему до того, як вона буде запущена в реальне середовище. Тестування безпеки API також є важливим кроком, оскільки відкриті API часто стикаються з різними загрозами безпеки, включаючи несанкціонований доступ, SQL-ін'єкції, атаки на відмову в обслуговуванні (DoS) і міжсайтовий скриптинг (XSS). Тестування безпеки допомагає виявити вразливості, які можуть бути використані зловмисниками для отримання несанкціонованого доступу або атаки на систему. Для цього використовуються спеціальні інструменти, такі як OWASP ZAP, Burp Suite

тощо, які допомагають виявити потенційні проблеми в аутентифікації, шифруванні, обробці помилок та інших критичних компонентах API. Такі тести допомагають виявити і виправити вразливості до того, як система буде використана у виробничому середовищі. Інтеграційне тестування є наступним важливим етапом і перевіряє взаємодію API з іншими компонентами системи та зовнішніми сервісами. Оскільки багато відкритих API взаємодіють зі сторонніми системами, важливо перевірити, чи правильно API обробляє запити при взаємодії з іншими сервісами та базами даних. Інтеграційне тестування може гарантувати, що всі компоненти системи, які взаємодіють через API, працюють коректно, і що система в цілому може безпомилково обробляти складні операції; тестування сумісності API також може перевірити, як система працює з різними версіями клієнтів і різними типами пристроїв.

Оскільки API можуть використовуватися різними додатками і веб-сервісами, необхідно забезпечити сумісність API з різними технологіями, операційними системами, пристроями і різними типами браузерів для веб-додатків. Тестування сумісності може гарантувати, що API працює в будь-якому середовищі, в якому він використовується; автоматизація тестування API значно полегшує процес перевірки якості та стабільності веб-системи. Використовуючи фреймворки автоматизованого тестування, такі як JUnit, Mocha і RSpec, можна проводити регулярні перевірки для перевірки функціональності кожного компонента системи після внесення змін. Це можна зробити кількома способами. Крім того, важливо виконувати тестування на зворотну сумісність після оновлення API. Оскільки відкриті API часто оновлюються або змінюються з часом, необхідно перевірити, що старі версії API залишаються сумісними з новими запитами і що це не перешкоджає функціональності клієнтів, які вже інтегрувалися з цією системою. Це необхідно для того, щоб переконатися, що нова версія API сумісна зі старою версією. Це допоможе запобігти проблемам з користувачами, які покладаються на старі версії API. Завершальним етапом тестування є юзабіліті-тестування, яке має на меті оцінити зручність використання API розробниками. Відкриті API повинні мати чітку і зрозумілу до-

кументацію, бути логічно структурованими і підтримувати зручні методи взаємодії; тестування юзабіліті API може гарантувати, що розробники зможуть швидко інтегрувати API в свої продукти, а час, витрачений на вивчення і використання цього інтерфейсу, буде мінімізувати час, витрачений на вивчення та використання цього інтерфейсу. Тестування відкритих API є необхідною частиною процесу розробки будь-якої веб-системи, оскільки воно гарантує, що API працює стабільно, безпечно і ефективно в умовах високих навантажень.

Комплексне тестування допомагає виявити можливі вразливості та помилки на етапі розробки і значно підвищує надійність і якість продукту ще до його впровадження в реальне середовище.

2 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ВЕБ-СИСТЕМИ

2.1 Загальний огляд архітектури розроблюваної системи

Розробка веб-системи для автоматизації процесу пропозиції пільгових продуктів передбачає інтеграцію декількох ключових компонентів, які забезпечують ефективну взаємодію між користувачами, адміністраторами та зовнішніми сервісами. Архітектура системи базується на мікросервісному підході, що дозволяє легко розширювати окремі компоненти, не впливаючи на інші частини системи. Основними компонентами архітектури є:

1. Клієнтська частина (веб-сайт) призначена для взаємодії між кінцевими користувачами та адміністраторами. Вона складається з двох інтерфейсів: для користувачів, які отримують пільгові продукти, і для адміністраторів, які керують правами доступу та користувачами.
2. Сервіс API забезпечує зв'язок між веб-інтерфейсом та серверною частиною. Це відкритий інтерфейс для перевірки даних користувачів, аутентифікації та взаємодії з базами даних і зовнішніми реєстрами.
3. Серверна частина, яка реалізує бізнес-логіку системи, обробляє запити, виконує перевірки та взаємодіє з іншими компонентами.
4. База даних, яка зберігає інформацію про користувачів, їхні права доступу та історію транзакцій. База даних забезпечує швидкий доступ до необхідних даних та їх ефективне оновлення.

Архітектура системи розроблена таким чином, щоб забезпечити високий рівень надійності, масштабованості та безпеки. Основна увага приділяється безпечному зберіганню даних, а також швидкому доступу до них в режимі реального часу.

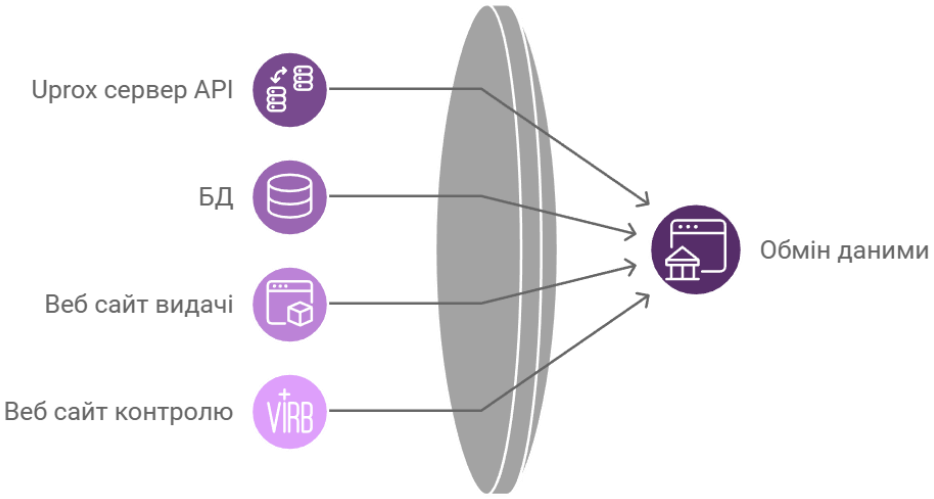


Рисунок 2.1 — Схематична архітектура системи

На рисунку 2.1 зображено загальну схему взаємодії між компонентами системи, де кожен елемент відповідає за свою частину процесу обробки даних та управління доступом.

2.1.1 Опис архітектурних компонентів

Основними компонентами веб-системи є:

1. Клієнтська частина. Це веб-інтерфейс, через який користувачі можуть отримувати доступ до пільг, перевіряти свою інформацію та взаємодіяти з системою. Для адміністраторів передбачений окремий інтерфейс, який дозволяє управляти користувачами, правами доступу та здійснювати моніторинг діяльності.
2. API. Інтерфейс, який забезпечує доступ до зовнішніх систем та реєстрів для перевірки даних користувачів. API відповідає за обробку запитів від клієнтської частини, перевірку прав доступу, обробку транзакцій і взаємодію з базою даних.

3. Серверна частина. Цей компонент відповідає за виконання бізнес-логіки системи. Він обробляє запити від клієнтів, здійснює перевірки прав доступу, обробляє помилки та взаємодіє з базою даних для збереження чи отримання необхідної інформації.
4. База даних. Центральне сховище даних, яке містить інформацію про користувачів, продукти, права доступу, а також історію транзакцій. База даних дозволяє оперативно зберігати і виводити необхідні дані, а також підтримує цілісність та безпеку даних.

2.1.2 Вибір архітектурного стилю

У процесі розробки веб-системи для автоматизації надання пільгових продуктів було обрано мікросервісний стиль архітектури. Таке рішення було зумовлене кількома важливими факторами, серед яких гнучкість, масштабованість та можливість самостійного оновлення окремих компонентів системи.

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, при якому система поділяється на окремі мікросервіси. Ці сервіси виконують свої власні специфічні завдання, мають чітко визначені обов'язки та взаємодіють з іншими мікросервісами через стандартизовані інтерфейси, такі як API. Мікросервіси можуть бути написані різними мовами програмування і мати різні технології, що дозволяє їм вирішувати конкретні завдання більш ефективно, приклад зображено на рис. 2.2.

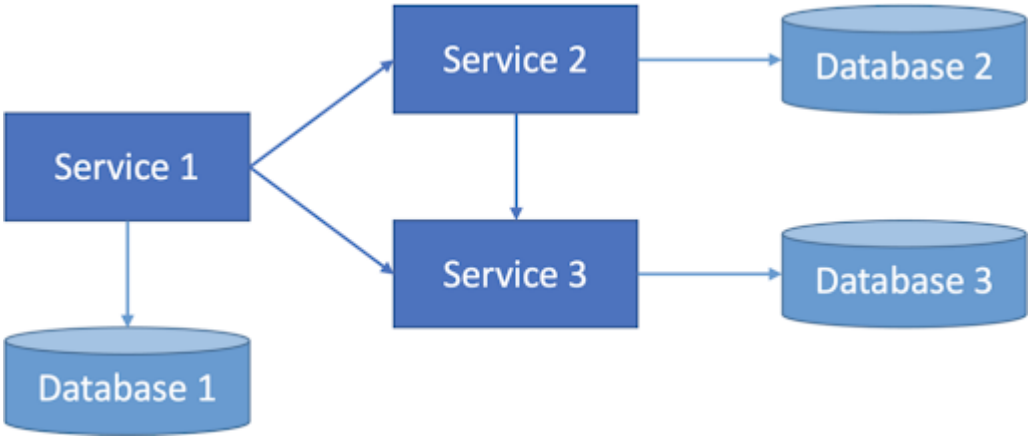


Рисунок 2.2 — Схематичне зображення мікросервісного стилю

Основні переваги обраної архітектури:

1. Масштабованість: Мікросервісна архітектура дозволяє масштабувати окремі компоненти системи незалежно один від одного. Наприклад, якщо навантаження на API-сервіс зросте, можна масштабувати тільки цей сервіс без необхідності масштабування всієї системи. Це значно знижує витрати на ресурси і дозволяє оптимізувати використання інфраструктури.
2. Гнучкість та незалежність розробки: Кожен мікросервіс є автономною одиницею, яка може розвиватися та оновлюватися окремо від інших компонентів системи. Це дозволяє паралельно розробляти та оновлювати різні частини системи без необхідності синхронізації змін між усіма компонентами. Крім того, команди розробників можуть працювати над різними мікросервісами одночасно, що підвищує швидкість розробки та знижує ймовірність виникнення помилок.
3. Незалежність технологій: Мікросервісна архітектура дає змогу використовувати різні технології для кожного сервісу. Наприклад, один мікросервіс може бути реалізований на Python, а інший — на Java, в залежності від того, яка технологія найкраще підходить для виконання конкретного завдання. Це дозволяє більш ефективно вирішувати спеціалізовані задачі, а також надає можливість впроваджувати нові технології без необхідності переписувати всю систему.
4. Забезпечення високої доступності та відмовостійкості: Оскільки кожен мікросервіс є окремим компонентом, у разі його несправності система може продовжувати працювати, за умови, що інші мікросервіси не залежать від нього. Це дозволяє забезпечити високий рівень доступності та відмовостійкості системи. Наприклад, навіть якщо один з мікросервісів не працює, інші можуть продовжувати виконувати свої функції без порушення роботи всієї системи.

5. Модульність та простота обслуговування: Мікросервісна архітектура дозволяє організувати систему в логічно відокремлені, незалежні частини, що спрощує її обслуговування та тестування. Оскільки кожен мікросервіс має свою окрему відповідальність, команда може зосередитись на роботі з певним компонентом без необхідності розбиратися в усій системі.
6. Простота оновлення та інтеграції: Оновлення окремих компонентів мікросервісної системи можуть виконуватися незалежно від інших частин. Якщо необхідно оновити API для інтеграції з новим зовнішнім сервісом або реалізувати нову функцію в одному з мікросервісів, це не потребує глобальних змін в інших частинах системи. Це значно знижує ризик збоїв при оновленнях та інтеграціях.

З іншого боку, монолітна архітектура об'єднує всі компоненти системи в єдиний додаток і працює як один великий блок. У цій архітектурі всі функції та процеси обробляються в межах одного додатка. Однак монолітні системи мають деякі суттєві недоліки, такі як складність масштабування, складність модифікації та оновлення, а також відсутність гнучкості при роботі з різними технологіями.

Використання монолітної архітектури може викликати значні труднощі при розширенні або зміні функціональності, оскільки кожне нове оновлення вимагає значних загальносистемних змін. З огляду на потребу в масштабованості, збільшення навантаження та необхідність інтеграції з великою кількістю зовнішніх сервісів, монолітна архітектура буде не дуже ефективною для такого типу веб-систем.

Обрана мікросервісна архітектура забезпечує максимальну гнучкість та можливості для подальшого розвитку системи. Можна ефективно обробляти запити користувачів, масштабувати окремі компоненти відповідно до навантаження та зменшити ризики, пов'язані з оновленнями та змінами системи. Це забезпечує високий рівень доступності та відмовостійкості, що важливо для систем, які обробляють великі обсяги даних і взаємодіють з іншими сервісами в режимі реального часу.

2.1.3 Зв'язок між компонентами системи

При розробці веб-системи для автоматизації надання пільгових продуктів ключовим елементом є ефективна взаємодія між компонентами системи, що забезпечує безперебійну роботу та високу продуктивність. Враховуючи вибір архітектури мікросервісів, система розділена на окремі мікросервіси, кожен з яких відповідає за певну функцію. Взаємодія між цими компонентами відбувається через стандартизовані інтерфейси, такі як API, що дозволяє кожному мікросервісу обмінюватися даними з іншими, незалежно від технології, на якій він був побудований.

Ключовим компонентом системи є API-шлюз, який виконує роль шлюзу для всіх зовнішніх запитів, що надходять до системи. Всі запити користувачів та інтеграція з іншими сервісами проходять через цей шлюз, що дозволяє централізовано контролювати та маршрутизувати запити до відповідних мікросервісів API-шлюз також відповідає за балансування навантаження між різними мікросервісами, що особливо важливо при великих обсягах трафіку Так і є.

Основними мікросервісами є, наприклад, модуль обробки запитів — цей сервіс отримує запити від API-шлюзу та вирішує, який мікросервіс повинен обробити запит. Цей модуль перевіряє ідентифікацію користувача, права доступу та інші умови, перш ніж передати запит відповідному компоненту системи для обробки; Модуль перевірки даних API — цей компонент відповідає за взаємодію із зовнішнім API та перевірку права користувача на отримання пільг. За допомогою цього мікросервісу статус користувача перевіряється в режимі реального часу шляхом запитів до зовнішніх реєстрів та баз даних, що значно прискорює процес обробки запитів та зменшує ймовірність помилок. Модуль бази даних — цей компонент відповідає за зберігання всіх даних про користувача, історію запитів та статус доступу до пільг. База даних є основним місцем зберігання інформації про права доступу користувача до різних видів товарів, а також забезпечує можливість виконання транзакцій при видачі товарів. Цей модуль відповідає за взаємодію з реляційними або нереляційними базами даних, залежно від вимог до продуктивності та

масштабованості. Модуль обробки транзакцій — цей сервіс обробляє всі фінансові операції, які відбуваються при видачі товарів зі знижкою. Цей модуль відповідає за правильний розрахунок вартості пільги, ведення обліку виданих інструментів та оновлення записів у базі даних. Для забезпечення інтеграції з платіжними системами цей мікросервіс взаємодіє з платіжними шлюзами та системами обробки транзакцій за допомогою відкритих API. Модуль адміністрування та контролю доступу — цей компонент призначений для системних адміністраторів і керує правами доступу, додає та видаляє користувачів, відстежує їхню активність у системі. Він також контролює конфігурацію системи та підтримує багатомовність інтерфейсу.

Зв'язок між компонентами забезпечується за допомогою RESTful API або GraphQL для комунікації між мікросервісами. Кожен мікросервіс має чітко визначений API, який дозволяє обмінюватися даними, забезпечує високий рівень абстракції та спрощує роботу з різними сервісами. Важливою частиною цього процесу є моніторинг та логування, що дозволяє відстежувати всі операції, які виконуються в системі, що важливо для відстеження помилок та забезпечення безпеки. Для досягнення високої відмовостійкості та масштабованості система використовує технології контейнеризації (наприклад, Docker) та оркестрування контейнерів (наприклад, Kubernetes).

Кожен мікросервіс розгортається як окремий контейнер, що дозволяє швидко масштабувати окремі частини системи відповідно до навантаження та потреб. Зв'язок між компонентами також включає систему черг для обробки асинхронних завдань, таких як обробка запитів від зовнішніх API або обчислення складних транзакцій. Це зменшує затримки для користувачів і підвищує продуктивність системи. Важливим аспектом є інтеграція із зовнішніми платіжними системами та національними реєстрами. Це забезпечує точність та актуальність інформації, а також автоматичну перевірку права користувачів на пільги.

Для цього система використовує стандартизовані API для обміну даними із зовнішніми джерелами, що підвищує ефективність обробки запитів і мінімізує ризик помилок.

Рисунок 2.3 — Компоненти системи

На рисунку 2.3 зображено як компоненти системи грають відіграють свою обрану роль. Це більш розширений опис зв'язку між компонентами системи, який детально пояснює, як кожен мікросервіс взаємодіє з іншими частинами системи через API, а також включає інформацію про використання контейнеризації, оркестрації та інтеграції з зовнішніми системами.

2.2 Модуль перевірки даних через API

Модуль валідації даних через API є важливим елементом системи, оскільки відповідає за обробку запитів, пов'язаних з перевіркою прав доступу користувачів до привілейованих продуктів та послуг. У зв'язку з необхідністю обробки великої кількості запитів у режимі реального часу, мікросервіс перевірки даних інтегрується з різними зовнішніми ресурсами, такими як державні реєстри, реєстраційні бази даних та іншими третіми сторонами, за допомогою стандартизованих API.

Одним з основних завдань цього модуля є перевірка ідентифікації користувача, яка здійснюється за допомогою API-запитів до зовнішніх джерел даних. Враховуючи, що система може використовувати різні типи ідентифікаційних даних, такі як ідентифікаційні номери, серії, номери документів тощо, система повинна перевіряти відповідність між цими даними та офіційною базою даних. Наприклад, для перевірки права на пільговий продукт може знадобитися запит до державного реєстру, який містить актуальну інформацію про соціальні та медичні пільги користувача API-запити здійснюються за допомогою стандартних протоколів, таких як REST і GraphQL, які дозволяють легко інтегруватися з різними зовнішніми сервісами.

Після отримання даних від зовнішніх систем модуль валідації виконує перевірку на основі набору правил, які визначають, чи має користувач право на привілейовані продукти та послуги. Сюди входять такі перевірки, як вік, фінансові можливості, медичні та соціальні пільги тощо. Важливо, що всі перевірки здійснюються в режимі реального часу, що дозволяє швидко реагувати на запити користувачів. Це досягається завдяки використанню високопродуктивних API-запитів, оптимізованих для обробки великих обсягів даних і забезпечення швидкості перевірки. Модуль також виконує необхідні перевірки, щоб переконатися, що отримані дані є коректними. Наприклад, якщо дані, надані користувачем, не збігаються з даними в офіційному реєстрі, система надсилає відповідне повідомлення про помилку або запит на оновлення даних.

Іншим важливим аспектом цього модуля є обробка помилок і кодів стану, отриманих від зовнішніх систем. Якщо в процесі запиту виникає проблема, наприклад, відсутнє з'єднання або невірні дані, модуль валідації обробляє такі ситуації і гарантує, що користувач отримає чітке повідомлення про проблему. При цьому система має можливість надсилати різні HTTP-статуси і коректно інформувати клієнта про результати виконання запиту. Вони включають в себе, чи була перевірка успішною, чи потрібна повторна спроба або модифікація даних. Для забезпечення безпеки та конфіденційності даних модуль верифікації забезпечує аутентифікацію та авторизацію через API. Всі запити, зроблені від імені користувачів, повинні бути попередньо автентифіковані з використанням сучасних стандартів, таких як OAuth або JWT. Це гарантує, що лише авторизовані користувачі мають доступ до конфіденційної інформації та підвищує загальну безпеку системи.

Таким чином, при кожному запиті перевіряється не лише особа користувача, але й його права доступу до відповідної інформації. Модуль також передбачає використання кешування для підвищення ефективності роботи системи. Кешування зменшує навантаження на зовнішні API, оскільки часто використовувані дані зберігаються на сервері і можуть бути використані повторно без необхідності робити новий запит до зовнішнього джерела. Це значно підвищує швидкість обробки запитів і зменшує затримки для користувачів.

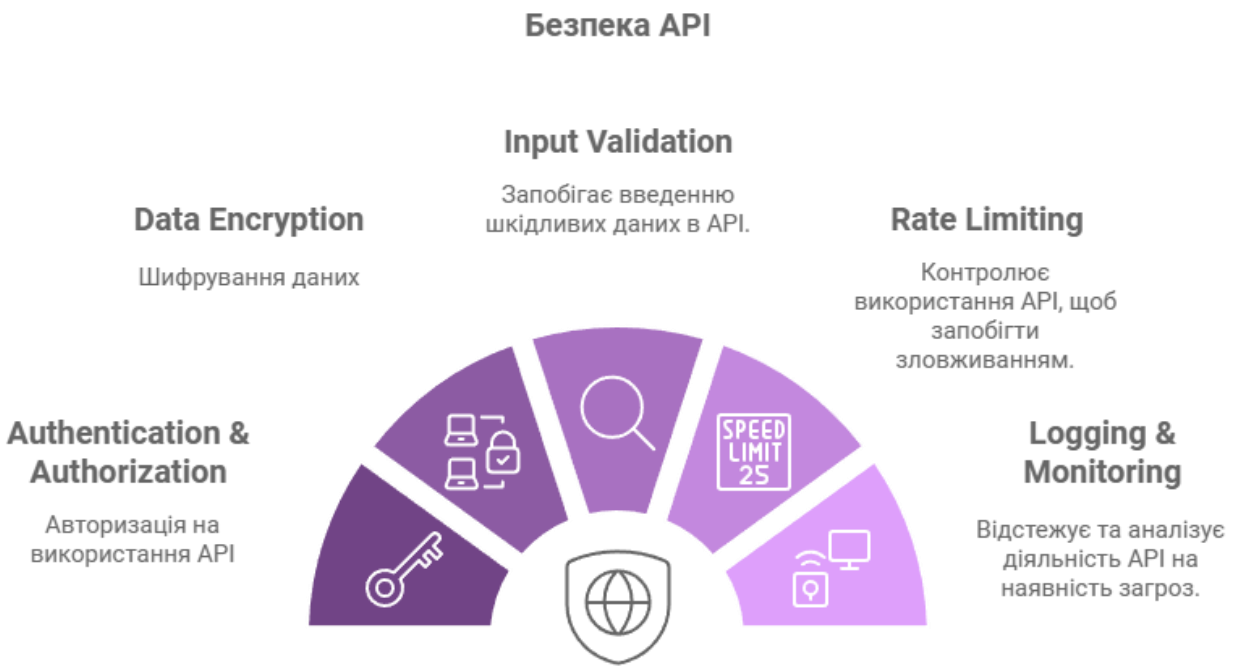


Рисунок 2.4 — Зображення засобів безпеки API

У результаті, як зображено на рисунку 2.4, модуль перевірки даних через API забезпечує не лише високий рівень автоматизації та ефективності обробки запитів, але й гарантує безпеку та конфіденційність даних, що є критичними в контексті надання пільгових продуктів. Він дозволяє інтегрувати систему з різними зовнішніми сервісами, забезпечуючи зручність, гнучкість і масштабованість веб-системи.

2.3 Модуль роботи з базою даних

Модуль бази даних є основним елементом системи, оскільки він відповідає за зберігання та обробку інформації про користувачів, їхні пільги та їхню взаємодію з системою. Веб-система, яка автоматизує надання пільгових продуктів, вимагає високоефективного та надійного управління даними, включаючи дані про користувачів, категорії пільг та транзакції. Модуль забезпечує підтримку всіх операцій з базою даних, включаючи зберігання даних, пошук, оновлення та видалення

інформації, і виконується за допомогою стандартних CRUD-операцій. Основною структурною одиницею бази даних є таблиці, які взаємодіють між собою за допомогою встановлених зв'язків. Створення таких зв'язків є важливим аспектом проектування бази даних, що забезпечує цілісність і точність даних та дозволяє виконувати ефективні запити на основі різних критеріїв.

Структура бази даних системи реалізована за допомогою декількох ключових таблиць, включаючи таблиці користувачів, таблиці пільг та таблиці транзакцій. Структура бази даних (ER-модель) ER-модель (Entity-Relationship Model) візуалізує основні сутності в базі даних та зв'язки між ними. У нашій системі основними сутностями є Користувач, Вигода, Транзакція, Продукт, Адміністратор та ін. Користувач: містить інформацію про користувачів системи, їхні персональні дані, права доступу та зв'язок з вигодами. Вигода: містить записи про різні вигоди, надані користувачеві і пов'язані з ним через таблицю «вигоди користувача». Транзакція: містить записи інформації, пов'язаної з видачею пільгових продуктів, включаючи час транзакції, ідентифікаційний номер користувача, тип і кількість продукту. Продукт: містить дані про пільгові продукти, які можуть бути видані, включаючи тип, кількість, опис та інші атрибути. Адміністратор: містить дані про адміністратора, який має доступ до управління правами користувачів, додавання та видалення записів. Між цими таблицями існують зв'язки, які визначають спосіб взаємодії даних. Наприклад, користувач може мати одну або кілька пільг, а транзакції пов'язані з користувачами через унікальні ідентифікаційні номери, щоб відстежувати видачу товару кожному користувачеві.

Зв'язки між таблицями реалізуються за допомогою зовнішніх ключів, які гарантують цілісність та узгодженість даних. Нижче на рисунку 2.5 наведено схему ER-моделі, яка показує основні таблиці бази даних та їхні зв'язки: CRUD-операції.

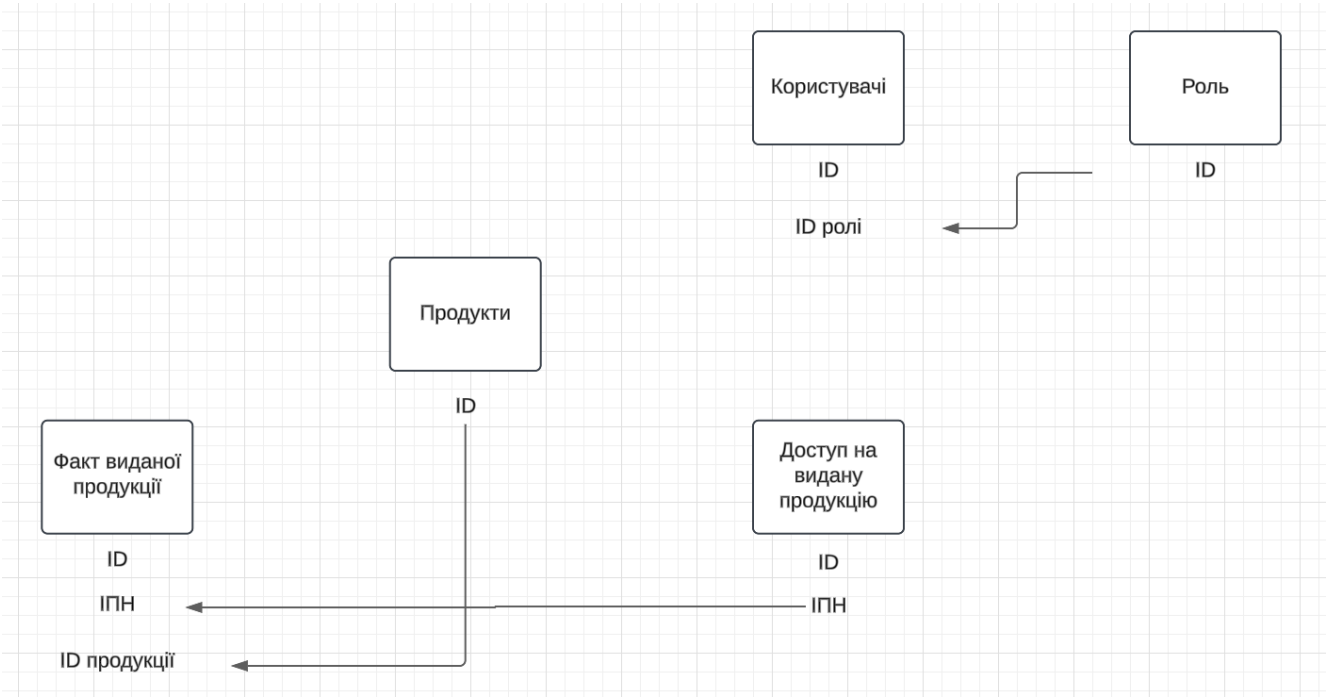


Рисунок 2.5— ER-модель зв’язків даних БД

Важливим аспектом використання бази даних є те, що вона підтримує стандартні CRUD-операції (створення, читання, оновлення та видалення). Кожна з них виконується за допомогою запиту до бази даних, що відповідає певній операції. Створення: ця операція відповідає за додавання нових записів до таблиці бази даних. Наприклад, коли в системі реєструється новий користувач, створюється новий запис у таблиці Користувач. Вона також включає в себе додавання нових пільг і продуктів, пропонованих користувачеві. Читання: Операції читання використовуються для отримання даних з таблиці і відображення їх в інтерфейсі системи або для подальшої обробки. Наприклад, можна прочитати дані про пільги, надані конкретному користувачеві, або інформацію про виконані транзакції. Це можуть бути прості запити на вибірку даних або складніші запити, що передбачають фільтрацію, сортування та агрегування даних. Оновлення: Оновлення даних необхідне, коли інформація змінилася, наприклад, оновлення інформації про користувача або пільги. Якщо користувач змінює свої контактні дані або додається нова пільга, система повинна оновити відповідний запис у базі даних. Видалення: операція видалення використовується для видалення непотрібних або застарілих записів у базі даних. Наприклад, якщо користувача видалено з системи, всі пільги та транзакції для

цього користувача мають бути видалені або позначені як неактивні. Для візуального представлення CRUD-операцій використовується діаграма, яка показує, як ці операції виконуються на різних етапах обробки даних у системі можна використати схему, як на рисунку 2.6, яка показує, як ці операції виконуються на різних етапах обробки даних у системі. Індексування та оптимізація запитів.

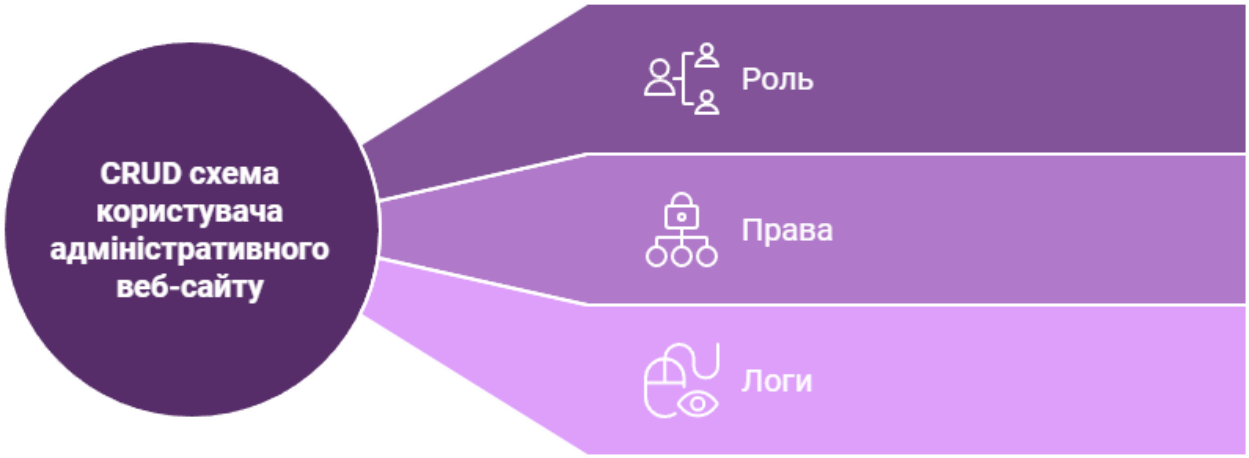


Рисунок 2.6 — CRUD схема на прикладі таблиці користувача

Одним з найважливіших аспектів продуктивності бази даних є індексування. Індексування може прискорити виконання запитів, особливо коли вам потрібно шукати за певними атрибутами (наприклад, за ідентифікаційним номером користувача або датою транзакції). Створення індексів на основі часто використовуваних полів може значно скоротити час виконання складних запитів. Крім індексування, важлива також оптимізація запитів. Це включає використання ефективних SQL-запитів, правильне використання операцій JOIN для об'єднання таблиць і оптимізацію ресурсів для обробки великих обсягів даних. Системи повинні бути спроектовані так, щоб обробляти запити якомога швидше, не створюючи зайвого навантаження на сервер. Модуль бази даних є критично важливим елементом у забезпеченні надійності та ефективності веб-системи для автоматизації соціальних виплат.

За допомогою правильно розробленої ER-моделі, CRUD-операцій та індексування система може швидко і точно обробляти запити, зберігати та оновлювати дані про користувачів, пільги та транзакції. Це не тільки забезпечує високу ефективність і зменшує ймовірність помилок, але й дозволяє розширювати систему в майбутньому.

2.4 Інтерфейс адміністрування для керування користувачами

Інтерфейс адміністрування є важливою частиною веб-системи. Це пов'язано з тим, що інтерфейс адміністрування надає адміністраторам зручний та ефективний спосіб управління користувачами, їхніми правами доступу та профілями в системі. Цей інтерфейс повинен бути інтуїтивно зрозумілим, простим у використанні та зручним для виконання таких операцій, як зміна прав доступу та управління профілями користувачів. Налаштовуваний інтерфейс дозволяє адміністратору виконувати всі операції, необхідні для підтримання актуальності та безпеки даних у системі. Інтерфейс адміністратора зазвичай складається з декількох основних компонентів, кожен з яких відповідає за окремий аспект управління користувачами та їхніми правами. Візуально він повинен бути зручним та інтуїтивно зрозумілим, щоб адміністратори могли швидко орієнтуватися та ефективно виконувати необхідні завдання. Ключові елементи інтерфейсу адміністратора Панель навігації: панель навігації: розташована вгорі або збоку екрана, вона містить основні розділи, за допомогою яких адміністратори можуть переходити до відповідних розділів, таких як користувачі, пільги, продукти та транзакції.

Панель навігації також забезпечує швидкий доступ до налаштувань системи, журналів активності та відновлення доступу до системи. Список користувачів: у цьому розділі адміністратори можуть переглянути всі профілі користувачів, які мають доступ до пільгових продуктів. Для кожного користувача відображається основна інформація, така як ім'я, ідентифікаційний номер (ІПН), статус пільги, історія транзакцій та рівень доступу.

Пошук і фільтрація користувачів: для додаткової зручності адміністратори можуть використовувати вбудовані інструменти для пошуку користувачів за різними параметрами, такими як ім'я, ІПН, пільговий статус і рівень доступу. Це дозволяє швидко знайти потрібного користувача в системі. Управління профілями користувачів: інтерфейс адміністратора надає можливість переглядати та редагувати профілі користувачів. Це включає зміну персональних даних, оновлення інформації про пільги, додавання або видалення прав доступу, а також перегляд і зміну статусу користувача (наприклад, активний, неактивний).

Управління правами доступу: адміністратори можуть змінювати рівень доступу кожного користувача відповідно до його ролі та статусу в системі. Це може бути доступ до всіх функцій або обмежений доступ до певних частин системи, наприклад, доступ до пільг або фінансових даних. Журнал активності: важливою частиною інтерфейсу є журнал активності, який дозволяє відстежувати всі дії, що виконуються адміністраторами та користувачами. Сюди входить інформація про створення, редагування, видалення записів, зміни прав доступу та всі транзакції, пов'язані з пільговими інструментами. Ведення журналу контролює безпеку системи та прозорість діяльності. Налаштування системи: інтерфейс також має розділ налаштувань, де адміністратори можуть змінювати основні параметри системи, такі як налаштування прав доступу, налаштування безпеки, налаштування сповіщень та автоматичного оновлення.

Модуль управління правами доступу є одним з найважливіших компонентів інтерфейсу адміністратора. Кожен користувач в системі може мати різні рівні доступу в залежності від його ролі та обов'язків. Інтерфейс управління правами доступу повинен бути гнучким, дозволяючи адміністраторам надавати або обмежувати доступ до різних функцій і ресурсів системи. Система повинна підтримувати різні моделі контролю доступу, такі як RBAC (контроль доступу на основі ролей) та ABAC (контроль доступу на основі атрибутів). Адміністратори можуть вибирати ролі (наприклад, «Адміністратор», «Менеджер», «Користувач») для кожного користувача і встановлювати відповідні права доступу для кожної ролі. Це дозволяє точно визначити, які операції може виконувати кожен користувач, наприклад, права

на редагування, перегляд звітів або зміну інформації про транзакції. Функції керування профілями

Окрім керування правами доступу, адміністратори можуть виконувати такі функції над профілями користувачів: додавання нових користувачів: адміністратори можуть додавати нових користувачів до системи шляхом ручного введення даних про користувача або імпортування їх із зовнішнього джерела. . Редагування профілю користувача: якщо користувач змінює свої дані, адміністратор може оновити інформацію в системі. Видалити користувача: якщо користувач більше не має доступу до привілеїв або якщо обліковий запис потрібно видалити, адміністратор може виконати цю операцію з інтерфейсу. Блокувати або активувати профілі, при необхідності адміністратор може тимчасово заблокувати доступ до профілю користувача або активувати його після завершення перевірок і внесення змін. Дана концепція представлена на рисунку 2.7

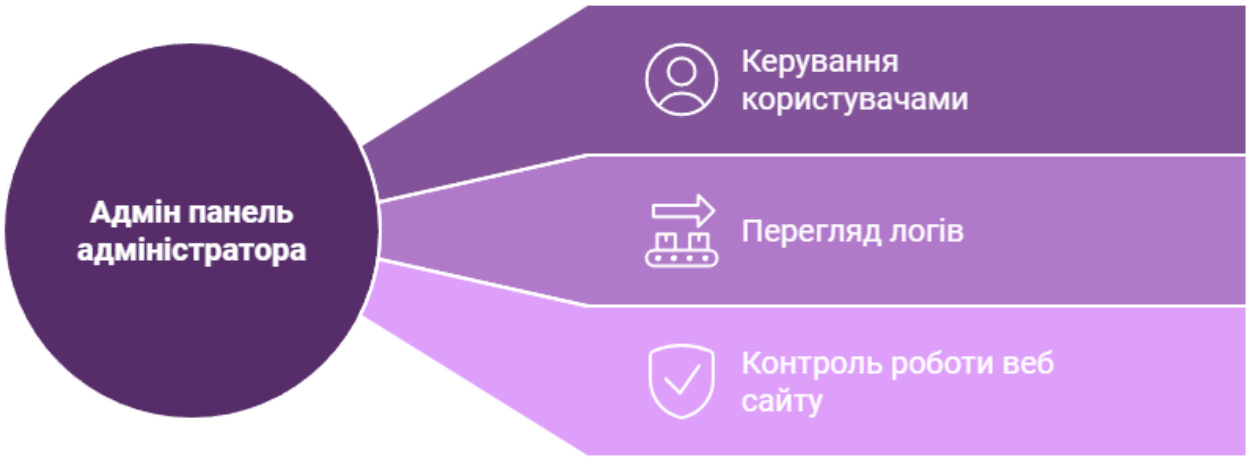


Рисунок 2.7 — Схема панелі адміністратора

Інтерфейс адміністрування є необхідним компонентом для ефективного керування веб-системою та забезпечення безпеки її роботи. Завдяки зручному й інтуїтивно зрозумілому інтерфейсу адміністратор може швидко і точно виконувати всі

необхідні операції для управління користувачами та їх правами доступу, що є ключовим аспектом для забезпечення прозорості та ефективності роботи системи.

2.5 Захист даних у системі

Захист даних є одним з ключових аспектів при проектуванні та розробці будь-якої веб-системи, особливо коли мова йде про конфіденційні дані користувачів, такі як особиста інформація, інформація про пільги, фінансові операції та медичні записи. У сучасному світі забезпечення безпеки даних є не лише технічним, але й юридичним завданням. Це пов'язано з тим, що порушення конфіденційності та несанкціонований доступ до конфіденційної інформації можуть мати серйозні наслідки як для користувачів, так і для організацій. Веб-системи, які автоматизують процес надання привілеїв або доступу до привілейованих продуктів, повинні не тільки включати технічні методи захисту даних, але й відповідати найвищим стандартам безпеки та конфіденційності, визначеним на національному та міжнародному рівнях. У багатьох галузях, таких як фінанси, охорона здоров'я та соціальне забезпечення, дані користувачів є критично важливими, і їх несанкціоноване розголошення може мати серйозні наслідки. Наприклад, якщо система пільг неправильно обробляє або передає індивідуальні дані користувачів, це може призвести до втрати довіри до системи, порушення зобов'язань щодо конфіденційності і навіть фінансових втрат. Тому забезпечення безпеки цих даних є не лише технічним завданням, але й вимогою для підтримки нормальної роботи системи та дотримання правових стандартів, таких як GDPR (Загальний регламент про захист даних) Європейського Союзу та Закон України «Про захист персональних даних». Основними цілями захисту даних є забезпечення конфіденційності, цілісності та доступності даних. Ці принципи гарантують, що дані не можуть бути передані або змінені без дозволу користувача, а доступ до них можуть мати лише уповноважені особи; у цьому розділі пояснюється, як ці аспекти можуть бути досягнуті в контексті роз-

робки веб-систем. Для забезпечення конфіденційності всі конфіденційні дані повинні бути захищені від несанкціонованого доступу. Одним з найефективніших способів досягти цього є шифрування даних як під час передачі (за допомогою протоколу TLS), так і під час зберігання в базі даних (за допомогою таких алгоритмів, як AES або RSA). Механізми перевірки цілісності використовуються для того, щоб гарантувати, що дані не будуть змінені або пошкоджені під час обробки або передачі. Це може бути реалізовано шляхом використання хешів і контрольних сум або протоколів, які гарантують цілісність даних під час передачі. Щоб гарантувати, що дані завжди доступні авторизованим користувачам, система повинна мати надійні механізми контролю доступу, такі як багаторівнева автентифікація, використання токенів доступу (JWT), а також резервне копіювання та відновлення даних.

2.5.1 Аутентифікація користувачів (OAuth, JWT)

Автентифікація користувачів є одним з ключових компонентів захисту даних у будь-якій веб-системі, і особливо важлива для систем, які обробляють конфіденційну інформацію, наприклад, у випадку з пільгами та соціальними послугами. Надійна автентифікація гарантує, що лише авторизовані користувачі можуть отримати доступ до системи, а всі дані, що передаються між клієнтом і сервером, захищені від несанкціонованого доступу. OAuth - це загальний протокол автентифікації, в якому сторонні додатки не обмінюються автентифікаційною інформацією, Він дозволяє обмежений доступ до ресурсів користувача без обміну обліковими даними.

Для веб-систем виплат OAuth можна використовувати для інтеграції із зовнішніми сервісами, такими як банківські або державні реєстри, які вимагають автентифікації користувачів через сторонні платформи (наприклад, Google або Facebook) Використовуючи OAuth, усуває необхідність зберігати конфіденційні дані користувачів, такі як паролі, на стороні веб-системи. Це забезпечує вищий рівень безпеки

та зменшує ризик витоку даних.JWT (JSON Web Token) - ще один важливий інструмент для автентифікації та авторизації.JWT - це компактний та безпечний спосіб передачі інформації між сторонами у вигляді токену. Коли користувач успішно входить в систему, сервер генерує JWT, що містить всі дані, необхідні для ідентифікації користувача та його прав доступу. Цей токен передається користувачеві, який використовує його для доступу до захищених ресурсів.

Перевага JWT полягає в тому, що його можна використовувати для безпечної ідентифікації в масштабованих системах Важливим аспектом використання JWT є те, що він працює без збереження будь-якого стану на сервері (stateless аутентифікація). Це означає, що кожен запит містить токен і сервер не зберігає інформацію про поточну сесію, що робить цю архітектуру особливо корисною для розподілених систем. Це також зменшує навантаження на сервер, оскільки немає необхідності зберігати дані про сесії. У контексті веб-системи пільг належне використання OAuth та JWT може значно підвищити безпеку та покращити користувацький досвід. За допомогою OAuth можна встановити безпечне з'єднання зі стороннім реєстром, а конфіденційні дані користувачів можуть бути перевірені на предмет наявності пільг. Використовуючи JWT, можна ефективно і безпечно передавати аутентифікаційні дані між клієнтом і сервером, зберігаючи при цьому гнучкість і масштабованість системи. Ці механізми не тільки підвищують безпеку, але й спрощують контроль доступу до конфіденційних даних і забезпечують високий ступінь захисту від несанкціонованого доступу та зловмисників. Нижче на рисунку 2.8 зображена типова логіка роботи OAuth.

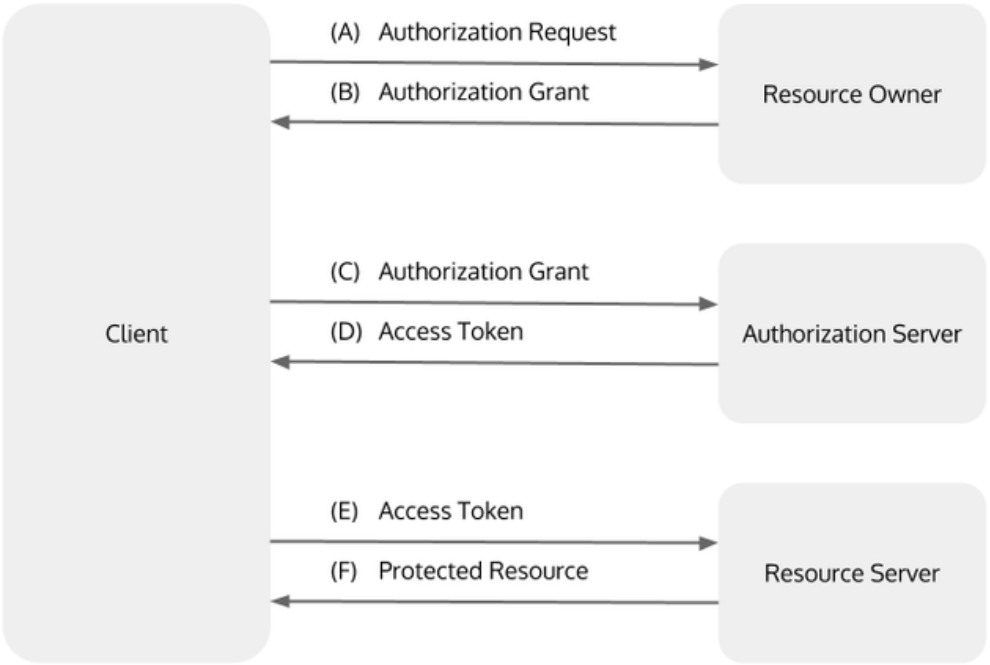


Рисунок 2.8 — Базове представлення OAuth

2.5.2 Шифрування даних (TLS, AES, RSA)

Шифрування даних є ключовим елементом забезпечення конфіденційності та безпеки у веб-системах, які обробляють конфіденційні або персональні дані. Воно запобігає несанкціонованому доступу до інформації, навіть якщо дані перехоплені в процесі передачі або зберігання. Для цього використовуються різні методи шифрування, найпоширенішими з яких є TLS, AES і RSA. Кожен з цих методів має свої особливості і використовується відповідно до конкретних вимог системи.

TLS (Transport Layer Security) - це протокол, який використовується для забезпечення безпеки передачі даних через мережу. Він шифрує дані між клієнтом і сервером і гарантує конфіденційність і цілісність переданих даних; TLS захищає дані від перехоплення і підробки під час передачі, що особливо важливо при використанні відкритих або ненадійних мереж, таких як Інтернет. Коли користувачі взаємодіють з веб-системами, наприклад, при вході в систему або здійсненні пла-

тежів, TLS шифрує весь трафік між браузером користувача і сервером. Це унеможливлює перехоплення або зміну цих даних третьою стороною. Основною перевагою TLS є його здатність створювати безпечне з'єднання, яке запобігає атакам типу «людина посередині», коли зловмисник може перехоплювати і змінювати інформацію, що передається між клієнтом і сервером. Протокол використовує симетричні шифри для швидкості передачі даних і асиметричні шифри для безпечного обміну криптографічними ключами. Прицнип його роботи представлений на рисунку 2.9.

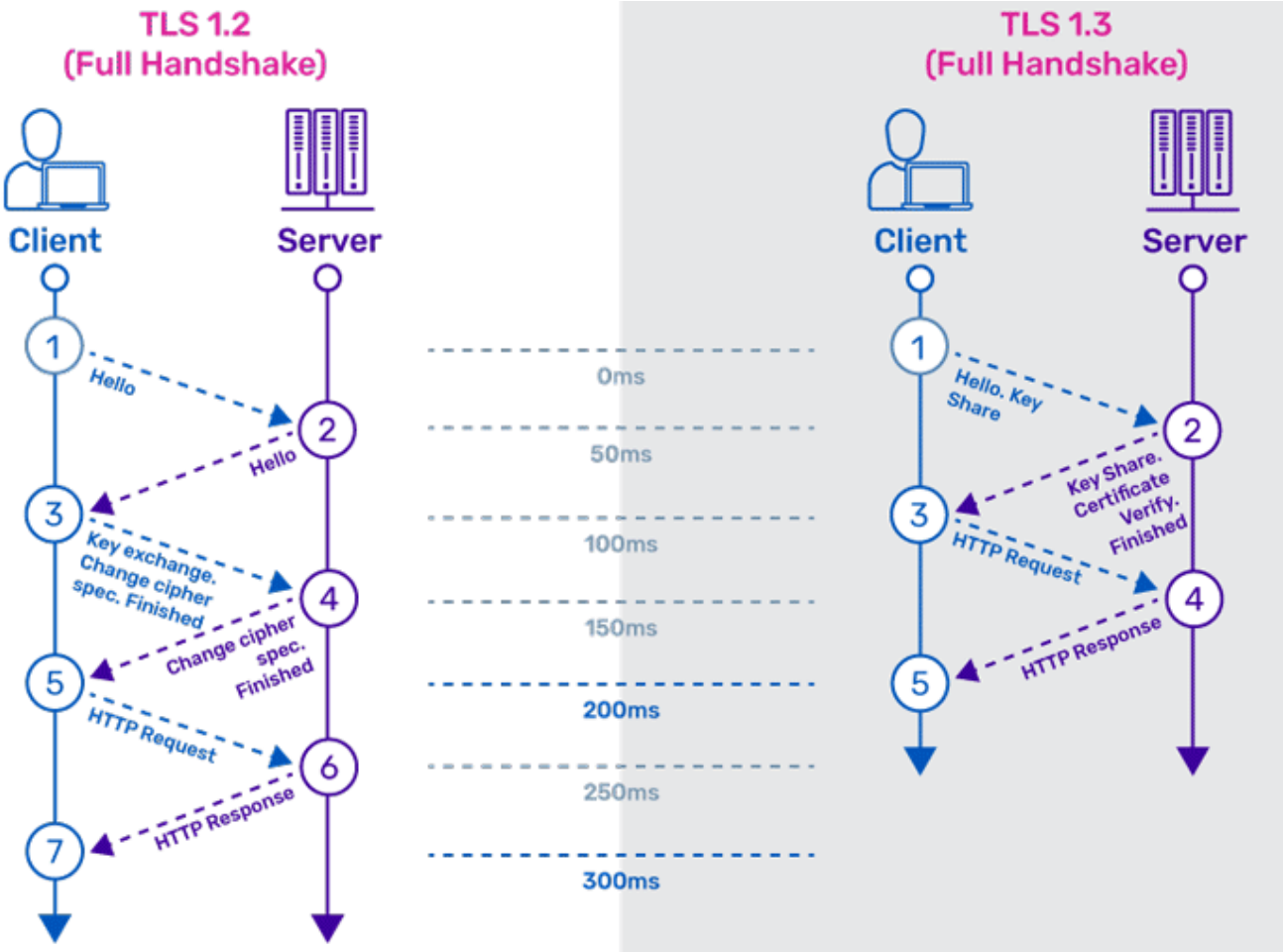


Рисунок 2.9 — Схема роботи протоколоу TLS 1.2 та 1.3

AES (Advanced Encryption Standard) - один з найпоширеніших симетричних алгоритмів шифрування. Він використовується для шифрування даних як при зберіганні, так і при передачі.AES використовує один і той же ключ для шифрування і розшифровки інформації, тому обидві сторони, що обмінюються зашифрованими даними, повинні мати доступ до цього ключа.AES підтримує різні довжини ключів:

128-біт, 192-біт і 256-біт AES підтримує різні довжини ключів: 128-біт, 192-біт і 256-біт, забезпечуючи різні рівні безпеки в залежності від конкретних вимог системи. Веб-системи з конфіденційними даними часто використовують AES для шифрування даних, що зберігаються в базах даних, або для зберігання важливої автентифікаційної інформації, такої як паролі користувачів, номери кредитних карток, медичні записи і т.д. Висока ефективність AES робить його популярним вибором для шифрування великих обсягів даних. зробила його популярним вибором для шифрування великих обсягів даних. Однак проблема симетричного шифрування полягає в тому, що, незважаючи на його високу ефективність, ключ шифрування повинен бути захищений по обидва боки зв'язку, щоб запобігти крадіжці. Принцип його роботи представлений на рисунку 2.10.

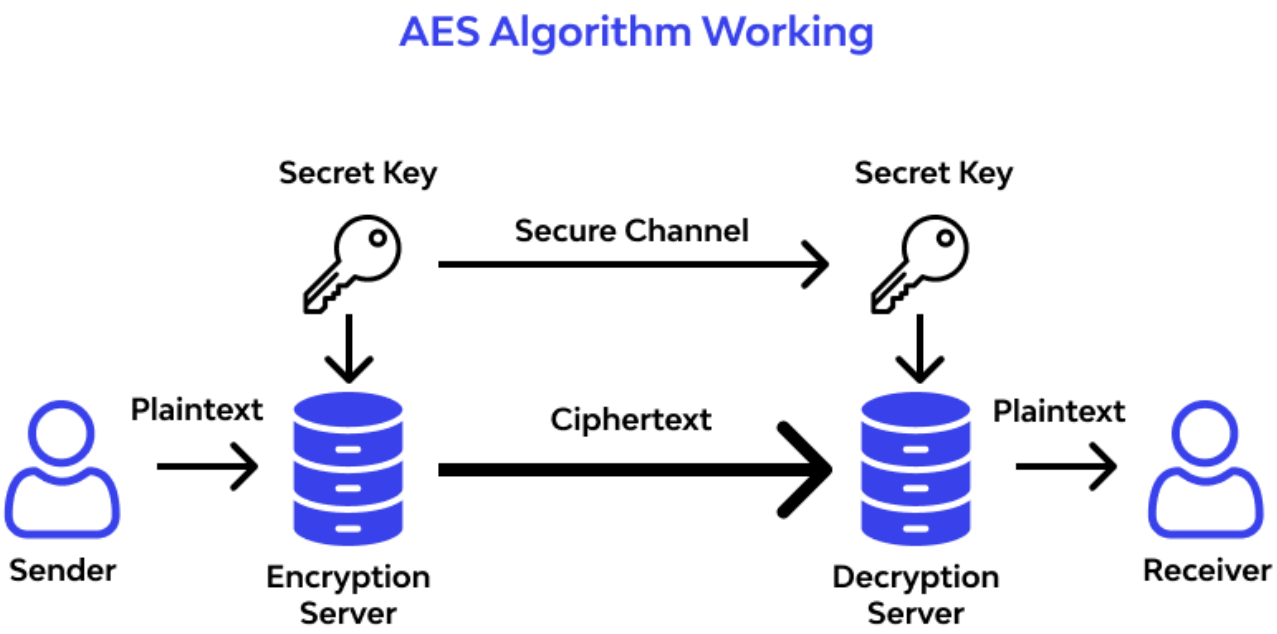


Рисунок 2.10 — Принцип роботи AES

RSA (Rivest-Shamir-Adleman) - це асиметричний алгоритм шифрування, в якому використовується пара відкритого і закритого ключів. Відкритий ключ використовується для шифрування даних, а закритий - для розшифрування даних RSA є основою для багатьох сучасних систем шифрування, включаючи такі протоколи,

як TLS У RSA відкритий ключ може бути наданий будь-якому користувачеві або системі для шифрування повідомлень, але розшифрувати дані може тільки власник закритого ключа може розшифрувати дані RSA використовується, зокрема, при створенні захищених з'єднань (наприклад, при ініціалізації з'єднання TLS) і для захисту симетричних ключів шифрування, таких як AES Ключі RSA набагато довші, ніж ключі AES, що значно сповільнює швидкість обробки. Однак, використання цього ключа є необхідним для початкового безпечного обміну між сторонами, перед використанням більш швидкого симетричного шифрування для обміну великими обсягами даних. Прицнип його роботи представлений на рис. 2.11.

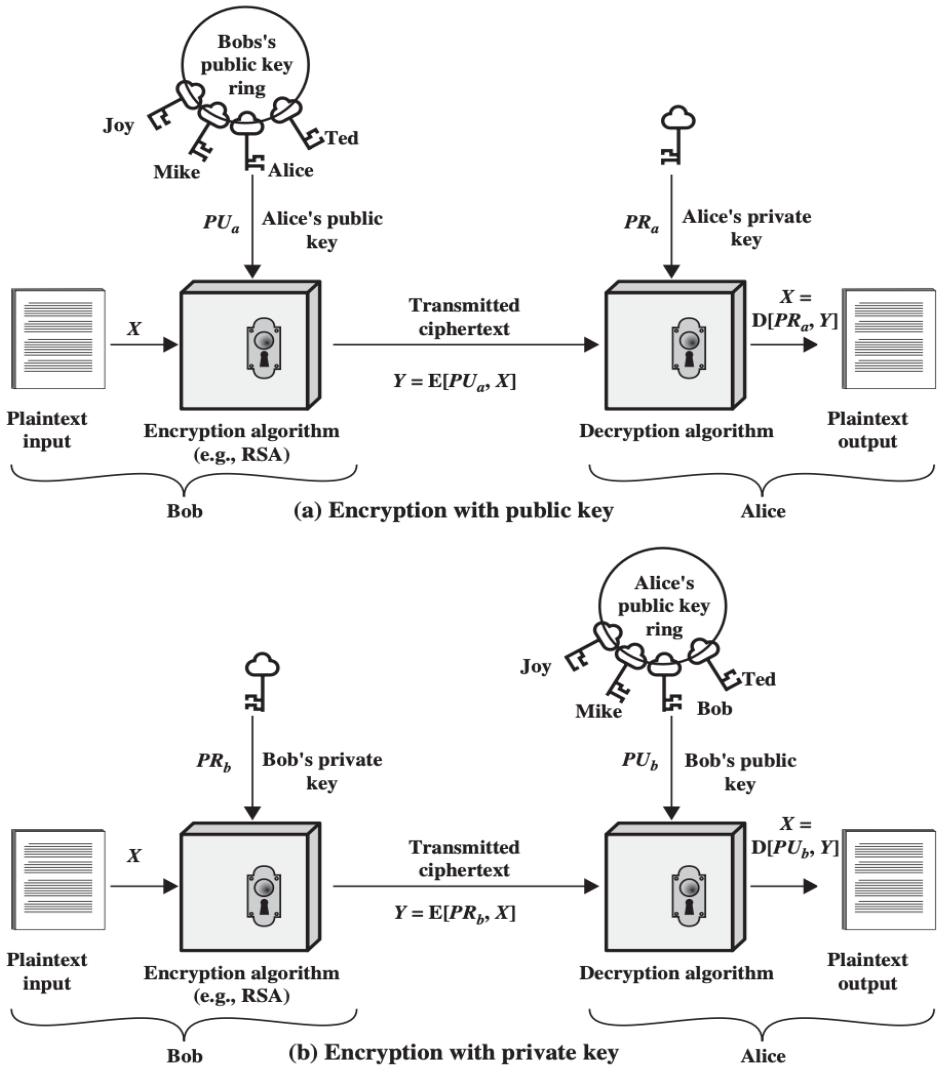


Figure 9.1 Public-Key Cryptography

Рисунок 2.11 — Принцип роботи алгоритму RSA

Шифрування даних за допомогою TLS, AES і RSA є ключем до безпеки в веб-системах, де важлива конфіденційність і цілісність інформації. основний метод забезпечення безпеки у веб-системах, де важлива конфіденційність і цілісність інформації. Використання цих технологій гарантує захист переданих і збережених даних від несанкціонованого доступу та перехоплення, і ці методи є ключовими елементами в побудові безпечних веб-додатків.

2.5.3 Захист від SQL-ін'єкцій та інших загроз

Оскільки SQL-ін'єкції є однією з найпоширеніших і найнебезпечніших вразливостей у веб-додатках, захист від них є важливим аспектом безпеки веб-систем. Цей тип атаки дозволяє зловмисникам виконувати небажані SQL-запити до бази даних, що призводить до крадіжки, модифікації або видалення даних, а також до доступу до конфіденційної інформації, такої як паролі користувачів і конфіденційні внутрішні дані.SQL-ін'єкція може бути використана для входу, пошуку, реєстрації Вона виконується шляхом введення шкідливих SQL-команд у поля введення веб-сторінки, наприклад, форми.

Якщо веб-система не перевіряє належним чином або не очищає вхідні дані, ці команди передаються безпосередньо в базу даних, де вони можуть бути виконані, потенційно змінюючи або пошкоджуючи дані. Використання підготовлених операторів і параметричних запитів: підготовлені оператори є найефективнішим способом запобігання SQL-ін'єкціям, оскільки вони відокремлюють SQL-код від вхідних даних, так що змінні (дані, введені користувачем) не можуть бути інтерпретовані як частина SQL-команди не можуть бути інтерпретовані як частина команди SQL. Замість того, щоб вставляти значення безпосередньо в запит, змінні передаються як параметри, які обробляються окремо від самого запиту. Це запобігає виконанню шкідливих SQL-команд, оскільки введені дані не можуть змінити структуру SQL-запиту. Приклад коду представлений на рис. 2.12.

```php
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password = ?");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
```

Рисунок 2.12 — Код на PHP SQL-ін’єкції

Перевірка та санітарна обробка вхідних даних: ще одним важливим способом захисту від SQL-ін'єкцій є перевірка та санітарна обробка даних, що вводяться користувачами. Це означає, що тип, формат і значення даних повинні бути перевірені перед тим, як їх можна буде використовувати в запиті до бази даних. Наприклад, поле, яке приймає лише числа, може обмежувати введення лише числовими значеннями. У текстових полях можна обмежити кількість символів або заборонити використання, спеціальних символів (таких як ', ;, -- тощо, які є частиною синтаксису SQL), дивитись рис. 2.13.

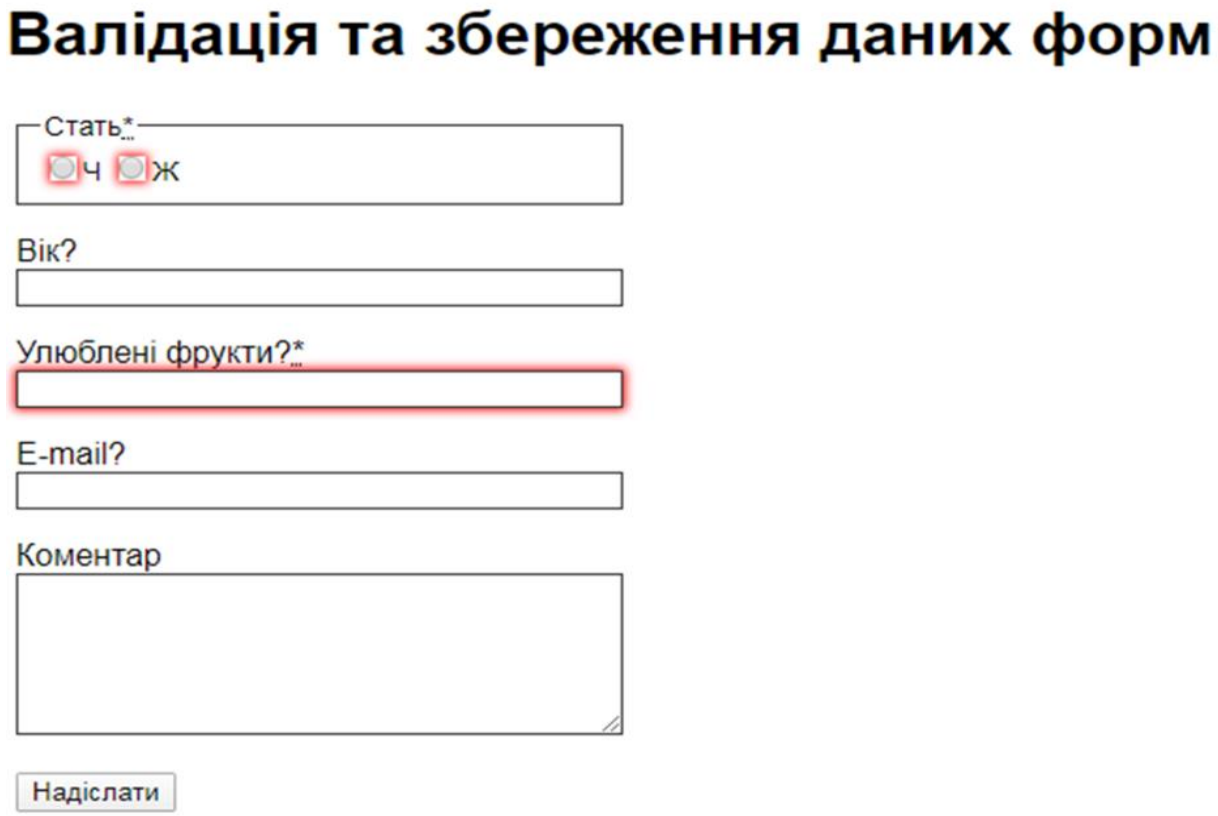


Рисунок 2.13 — Базовий приклад валідації вхідних даних на клієнті

Екранування спеціальних символів: при використанні традиційної генерації SQL-запитів без підготовленого запиту важливо екранувати спеціальні символи, щоб вони не інтерпретувалися як частина команди SQL. Наприклад, MySQL використовує функцію mysql_real_escape_string() для екранування таких символів, як одинарні лапки ('), подвійні лапки (") і зворотний слеш (‟~‟). Важливо надати користувачам бази даних мінімально необхідні права доступу. Це означає, що навіть у разі успішної SQL-ін'єкції зловмисник не зможе виконати шкідливі операції, такі як видалення таблиць або зміна критично важливих даних. Наприклад, обліковий запис користувача, який використовується для підключення до бази даних з веб-додатку, повинен мати лише права на читання та запис і не повинен мати можливості видаляти або змінювати структуру бази даних.

Окрім SQL-ін'єкцій, веб-система повинна бути захищена від низки інших потенційних загроз, таких як:

1. Cross-Site Scripting (XSS): Це атака, при якій зловмисник вставляє шкідливий JavaScript-код у веб-сторінку, який потім виконується на стороні користувача. Для захисту від XSS важливо правильно екранувати або фільтрувати ввідні дані, особливо ті, що вставляються у HTML-контент.
2. Cross-Site Request Forgery (CSRF): Цей тип атаки змушує користувача виконувати несанкціоновані дії на веб-сайті, на якому він вже аутентифікований. Для запобігання CSRF використовуються токени, які підтверджують, що запит був зроблений з довіреного джерела.
3. Brute Force атаки: Веб-система повинна використовувати механізми захисту від брутфорс-атак, наприклад, лімітування кількості спроб входу, CAPTCHA, двофакторну аутентифікацію.
4. Використання безпечних сесій: Важливо захищати сесії користувачів від крадіжки за допомогою таких методів, як використання безпечних cookie, встановлення коректних параметрів для сесій та забезпечення шифрування всіх переданих даних.

Захист від SQL-ін'єкцій та інших загроз є невід'ємною частиною загальної стратегії безпеки веб-системи, і його реалізація повинна бути системною та комплексною. Це дозволяє забезпечити належний рівень захисту від численних атак, що можуть загрожувати як самим даним, так і функціональності веб-системи.

2.5.4 Регулювання доступу до чутливих даних

Контроль доступу до конфіденційних даних є критично важливим елементом безпеки веб-системи, особливо коли мова йде про особисту інформацію користувачів, фінансові дані, медичні записи та іншу конфіденційну інформацію. Система повинна забезпечувати адекватний контроль доступу, щоб уникнути несанкціонованого перегляду, зміни або видалення таких даних. Це може мати серйозні наслідки, включаючи порушення вимог законодавства та втрату довіри користувачів. Для контролю доступу до конфіденційних даних використовуються різні підходи, найефективнішими з яких є контроль доступу на основі ролей (RBAC) та контроль доступу на основі атрибутів (ABAC). У цих підходах доступ до даних визначається на основі ролі користувача в організації та певних атрибутів, які визначають, які ресурси і в якому обсязі користувач може використовувати RBAC, наприклад, адміністратор, користувач, менеджер або інші категорії, які визначають повноваження кожної особи.

Розподіл доступу на основі ролей. Наприклад, менеджери мають повний доступ до всієї системи, включаючи можливість змінювати або видаляти конфіденційні дані, тоді як звичайні користувачі можуть лише переглядати обмежену інформацію. Такий підхід зменшує кількість людей, які мають доступ до конфіденційних даних, і гарантує, що поведінка користувачів контролюється відповідно до їхніх посадових обов'язків. abac є більш гнучким підходом і дозволяє використовувати додаткові критерії для регулювання доступу, такі як час доби, місцезнаходження користувача або контекст запиту на доступ до даних. Можна застосовувати додаткові критерії, такі як конкретні умови, що відображають контекст запиту на

доступ до даних. Це дозволяє забезпечити більш детальний рівень контролю за доступом до конфіденційних даних, що важливо в ситуаціях, коли необхідно враховувати інші фактори, а також роль користувача.

Для забезпечення належного контролю доступу також важливо використовувати методи автентифікації та авторизації, які підтверджують особу користувача та його право на доступ до певних ресурсів. Багатофакторна автентифікація (MFA) є важливим інструментом для підвищення безпеки. Це може бути комбінація паролів, одноразових кодів, отриманих через SMS або мобільні додатки, біометрія або інші методи.

Слід також приділити увагу моніторингу та реєстрації доступу до конфіденційних даних. Всі дії користувача, пов'язані з доступом до таких даних, повинні ретельно реєструватися для подальшого аналізу та виявлення підозрілих або несанкціонованих спроб доступу. Журнали корисні не тільки для виявлення потенційних атак і помилок, але й для аудиту дотримання політик безпеки та регуляторних вимог. Іншим важливим аспектом є регулярний перегляд та оновлення політик доступу, оскільки організаційні структури та вимоги до безпеки можуть змінюватися з часом. Це включає перегляд ролей і дозволів користувачів, видалення або зміну прав доступу для користувачів, які більше не працюють в організації, а також оновлення засобів контролю в міру зміни стандартів безпеки і законодавчих вимог. Таким чином, управління доступом до конфіденційних даних має бути багаторівневим процесом, що включає використання відповідних методів автентифікації, встановлення політик доступу та моніторинг для підтримання належного рівня безпеки та відповідності вимогам. Це зменшить ризик витоку та несанкціонованого використання конфіденційних даних, а також забезпечить довіру користувачів та дотримання законодавства.

2.6 Масштабування та ефективність роботи системи

Масштабованість і продуктивність веб-систем є одним з ключових аспектів, що впливають на їхню здатність працювати в умовах високих навантажень, а також на їхню здатність швидко реагувати на мінливі умови і запити користувачів. Для того, щоб системи могли справлятися з великими обсягами даних і численними запитами, необхідно впроваджувати сучасні методи масштабування та оптимізації, серед яких ключову роль відіграють хмарні сервіси та архітектури високої доступності.

2.6.1 Хмарні сервіси та їх роль у масштабуванні

Хмарні сервіси стали важливим інструментом для забезпечення масштабованості веб-систем, оскільки ресурси можна швидко адаптувати до поточних потреб. Використовуючи хмару, компанії можуть уникнути витрат, пов'язаних з фізичною інфраструктурою, і мати гнучкість у зміні обсягів обробки даних та обчислювальної потужності. Найбільші хмарні провайдери, такі як Amazon Web Services (AWS), Microsoft Azure і Google Cloud, пропонують ряд інструментів автоматичного масштабування, які знижують витрати на ресурси, зберігаючи при цьому високу доступність системи. Однією з головних переваг використання хмарних сервісів є можливість горизонтального масштабування. Це дозволяє додавати нові ресурси, такі як сервери, бази даних і системи зберігання, без значних капітальних інвестицій.

Горизонтальне масштабування означає додавання нових серверів і віртуальних машин до існуючої інфраструктури, що дозволяє їй обробляти більші обсяги запитів і даних, а також підвищує надійність і ефективність системи. Хмарні сервіси можуть забезпечувати автоматичне масштабування (здатність системи самостійно збільшувати або зменшувати свою обчислювальну потужність залежно від

поточного навантаження). Наприклад, якщо сервер перевантажений, хмарний провайдер може автоматично додавати екземпляри для обробки запитів, щоб підтримувати належну швидкість роботи. Це може зменшити витрати на інфраструктуру, оскільки ресурси можуть бути тимчасово зменшені в періоди низького навантаження. З іншого боку, хмара також забезпечує вертикальне масштабування (збільшення потужності окремих серверів) (наприклад, шляхом додавання ресурсів оперативної пам'яті або процесора), завдяки чому сервери можуть швидко адаптуватися до нових ситуацій без необхідності додавати нові екземпляри. Ще однією важливою особливістю хмарних обчислень є можливість автоматичного балансування навантаження.

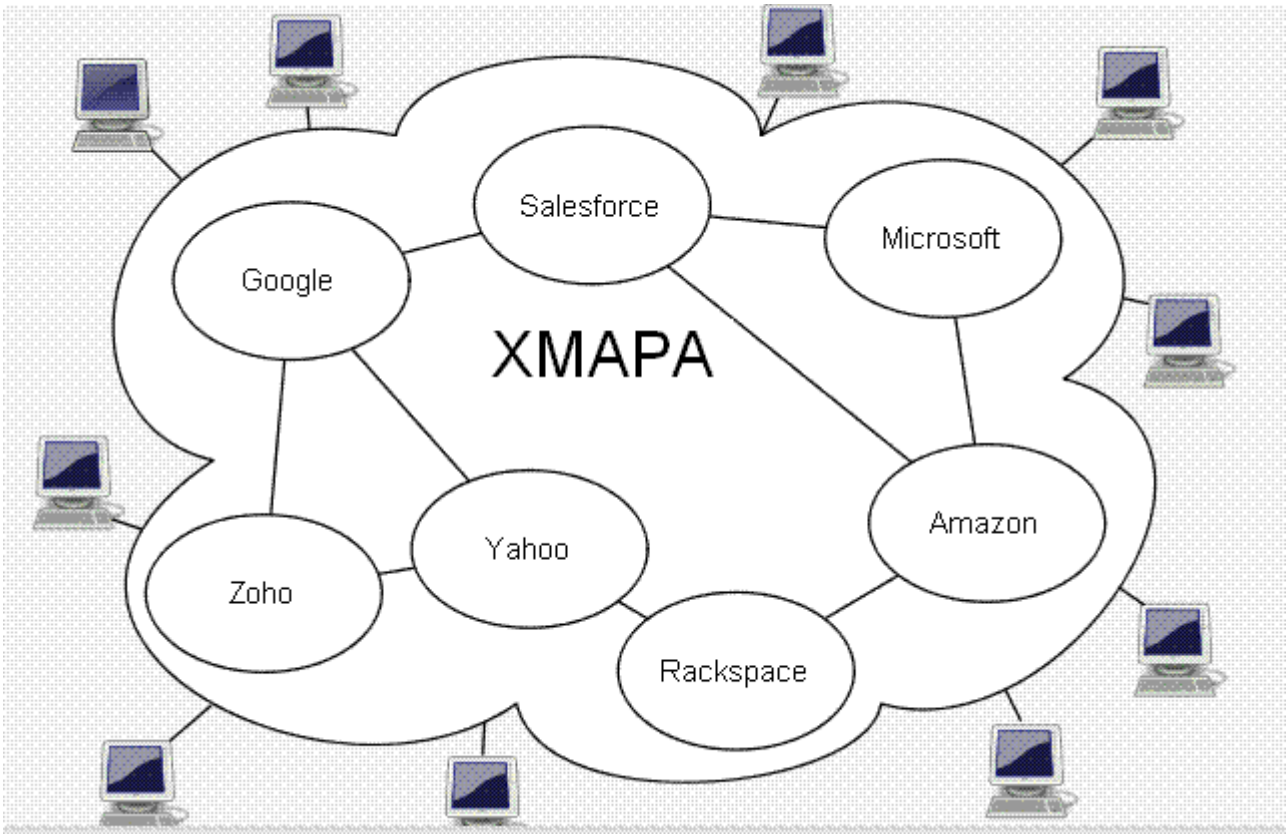


Рисунок 2.14 — Схематичне зображення взаємодії хмарних середовищ

Як зображено на рисунку 2.14 балансувальники навантаження розподіляють вхідні запити між кількома серверами, щоб вирівняти навантаження на систему і максимізувати швидкість обробки запитів. Цей процес важливий для підтримки високої продуктивності, навіть коли кількість запитів стрімко зростає. Крім того, хмарні сервіси забезпечують надійне резервне копіювання та відновлення даних, що

знижує ризик їх втрати та забезпечує безперервність роботи системи у випадку непередбачуваних збоїв. Завдяки географічно розподіленим центрам обробки даних хмарні провайдери можуть запропонувати резервне копіювання та відновлення даних у разі локальної катастрофи або технічного збою, що значно підвищує надійність системи.

2.6.2 Відмовостійкість та високодоступні архітектури

Відмовостійка та високодоступна архітектура є ключовим елементом у забезпеченні безперервної роботи веб-системи. Відмовостійкість - це здатність системи продовжувати роботу в разі виникнення помилок або збоїв в окремих компонентах. Високодоступна архітектура має на меті забезпечити максимально можливий час безвідмовної роботи в разі збою інфраструктури. Однією з основних стратегій досягнення відмовостійкості є побудова архітектури на основі принципу «відмовостійкості». Відмовостійкість може бути досягнута за рахунок використання декількох серверів, баз даних і серверів балансування навантаження, розподілених між різними фізичними або віртуальними середовищами. Це гарантує, що, наприклад, якщо один із серверів або сервісів виходить з ладу, трафік автоматично перенаправляється на інші активні компоненти системи. Резервування на рівні бази даних також є важливим фактором забезпечення високої доступності. Бази даних з можливістю реплікації можуть створювати копії даних на різних серверах, так що в разі виходу з ладу одного сервера система може швидко переключитися на резервний сервер без втрати даних або порушення роботи системи. Крім того, деякі хмарні провайдери пропонують можливість налаштувати автоматичну реплікацію даних на рівні сховища, що забезпечує безпеку та доступність даних. Для підвищення відмовостійкості також використовуються механізми моніторингу та автоматичного відновлення. Моніторинг дозволяє постійно перевіряти стан усіх компонентів системи та виявляти потенційні проблеми до того, як вони призведуть до серйозних

збоїв. Системи автоматичного відновлення, особливо ті, що базуються на контейнерних технологіях (Docker, Kubernetes), можуть виявляти несправності і запускати нові екземпляри сервісів або відновлювати попередні стани системи без втручання оператора. У контексті архітектури високої доступності також важливо впроваджувати механізми балансування навантаження між серверами. Балансувальники навантаження можуть використовуватися для розподілу запитів між декількома серверами або екземплярами, що не тільки підвищує продуктивність системи, але і забезпечує безперервну роботу навіть при високих навантаженнях.

Балансувальники навантаження також можуть автоматично виявляти неактивні або несправні сервери і перенаправляти трафік на працюючі компоненти системи. Крім того, архітектура високої доступності передбачає наявність резервних каналів зв'язку, що дозволяє системі продовжувати функціонувати, навіть якщо основний канал передачі даних виходить з ладу. Такий підхід особливо важливий для фінансових, медичних та урядових організацій, де системи повинні підтримувати безперервний доступ до ресурсів у будь-який час, і де переривання доступу може мати серйозні наслідки. Крім того, для підвищення відмовостійкості важливими є методи горизонтального масштабування та автоматичного відновлення. Горизонтальне масштабування дозволяє системам додавати нові сервери та екземпляри для обробки запитів, зменшуючи навантаження на окремі компоненти та забезпечуючи швидке відновлення у разі збою Контейнеризація та оркестрування з використанням таких технологій, як Kubernetes автоматизує процеси розгортання, масштабування та відновлення і зменшує час простою системи.

Важливою частиною побудови високодоступної архітектури є надійний моніторинг продуктивності та стану системи в режимі реального часу, а системи моніторингу, такі як Prometheus та Grafana, можуть використовуватися для отримання даних про стан та навантаження системи в режимі реального часу та попередити вас про потенційні проблеми до того, як вони стануть критичними. Моніторинг дає можливість швидко реагувати на проблеми та мінімізувати час відновлення після збою. Масштабованість та відмовостійкість неможливі без ефективної організації процесів резервного копіювання та відновлення даних.

Регулярне автоматизоване резервне копіювання мінімізує ризик втрати даних і гарантує, що систему можна буде відновити до попереднього стану в разі критичного збою. Це ключовий елемент архітектури системи високої доступності, де навіть короткочасна втрата даних може мати катастрофічні наслідки. Таким чином, хмарні сервіси та архітектури високої доступності є ключовими інструментами для масштабування сучасних веб-систем, підтримки працездатності в умовах високих навантажень і забезпечення безперервної роботи в разі різних збоїв і проблем. Вони дозволяють системі бути гнучкою, адаптивною та готовою до змін, що в кінцевому підсумку сприяє підвищенню надійності системи та довіри користувачів.

2.7 Інтеграція з зовнішніми системами та сервісами

Інтеграція із зовнішніми системами є ключовим елементом для сучасних веб-систем. Вона дозволяє обмінюватися даними між різними платформами та системами, створюючи більш потужні, гнучкі та ефективні рішення. При розробці веб-системи для автоматизації процесів виплат пільг інтеграція із зовнішніми сервісами та системами, такими як сервери uplox та 1С, забезпечує актуальність даних, точність верифікації користувачів та виплат пільг, а також знижує ризик помилок при обробці інформації. Всі ці процеси здійснюються через API та бази даних, які є основними інструментами для синхронізації та обміну даними між платформами.

2.7.1 Інтеграція з базами даних

Оскільки всі основні дані про користувачів, пільги, транзакції та інші елементи системи зберігаються в локальних і зовнішніх базах даних, інтеграція баз даних є важливим елементом для належного функціонування системи. Веб-системи, які автоматизують процес оформлення пільг, повинні інтегруватися з різними базами

даних, щоб забезпечити актуальність і точність даних, а також ефективно управляти правами доступу, верифікацією користувачів і оновленням даних у режимі реального часу.

Важливим елементом такої інтеграції є використання зовнішніх баз даних 1С, з яких вона отримує інформацію про користувачів та їхні пільги. 1С - одна з найпопулярніших автоматизованих систем для обліку, управління бізнесом та бухгалтерії в Україні. Вона використовується для обробки широкого спектру облікових даних, включаючи фінансову звітність, нарахування пільг та соціальних виплат.

Інтеграція з 1С є важливою частиною нашої веб-системи, яка дозволяє користувачам отримувати актуальну інформацію про надані їм пільги. Використовуються різні підходи до інтеграції з 1С, залежно від того, яка версія цієї системи використовується. Одним з найпоширеніших підходів є обмін даними через стандартні формати файлів (XML, CSV тощо) або через API та веб-сервіси, що підтримуються самою системою 1С Інтеграція через API дозволяє досягти високого ступеня автоматизації обміну даними. Це дає можливість, наприклад, забезпечити оновлення даних у режимі реального часу: через API можна регулярно отримувати інформацію про користувачів та їхні права, що зменшує ймовірність помилок при обміні даними.

Зменшення потреби у ручному введенні даних: автоматичне отримання даних з 1С може значно зменшити ймовірність помилок, пов'язаних з ручним, неточним або неповним введенням даних. Автоматична перевірка та оновлення даних: API дозволяє користувачам миттєво перевірити, чи мають вони право на пільги в поточному періоді, чи змінилися їхні дані

Для ефективної інтеграції з 1С важливо чітко визначити структуру обміну даними та взаємодії між системами Важливо чітко визначити структуру обміну даними та взаємодії між системами. У випадку нашої системи основними даними, які отримуються з 1С, є: список користувачів, які мають право на пільги: інформація про осіб, зареєстрованих у системі пільг, їхні дані (ідентифікаційний номер, адреса, кількість пільг, інші параметри тощо). Дані про видачу пільг: інформація про кількість виданих пільг за звітний період та статус кожної транзакції (статус видачі,

дата видачі тощо). Інформація про умови надання пільг: дані про термін, на який надаються пільги, та характеристики пільгових продуктів або послуг, доступних користувачеві; підключення до API та бази даних 1С, підключення до сервера 1С та налаштування доступу через API. Збір даних: здійснення запитів до API 1С та читання файлів для отримання необхідної інформації про користувачів та пільги. Обробка даних: отримання даних з 1С, їх обробка в системі, перетворення у формат, зручний для подальшої обробки та зберігання в локальній базі даних.

Синхронізація даних, отриманих з 1С, є автоматичною з локальною базою даних веб-системи; після отримання даних з 1С система повинна виконати деякі важливі операції для їх ефективної обробки. Форматування даних: дані, отримані з 1С, можуть бути в різних форматах (наприклад, XML, CSV тощо) і повинні бути приведені до єдиного формату, зручного для подальшого використання, перед тим, як зберігатися в базі даних веб-системи. Зберігання даних у локальній базі даних: після обробки та перевірки даних вони зберігаються в локальній базі даних веб-системи. Для цього створюються відповідні таблиці для кожного типу інформації (користувачі, пільги, транзакції). Індексування та оптимізація запитів: щоб забезпечити швидкий доступ до даних, слід створити індекси в таблицях бази даних. Це прискорює виконання запитів при доступі до бази даних, що особливо важливо при роботі з великими обсягами інформації. Для кожного типу даних в базі даних необхідно створити окрему структуру таблиць, що дозволить забезпечити зручний доступ та ефективне оновлення і видалення даних.При інтеграції з 1С безпека даних має вирішальне значення, особливо коли мова йде про конфіденційну інформацію користувачів. Захист даних під час передачі по мережі за допомогою різних методів шифрування, шифрування даних, що передаються через API: для захисту інформації, що передається між веб-системою і 1С, використовуються протоколи, які підтримують шифрування (наприклад, HTTPS, SSL/TLS). Це запобігає перехопленню даних під час передачі по мережі. Шифрування даних у базі даних, щоб запобігти несанкціонованому доступу до даних, всі конфіденційні дані повинні зберігатися в зашифрованому вигляді. Сюди входять персональні дані користувачів, номери ІПН та інформація про пільги.

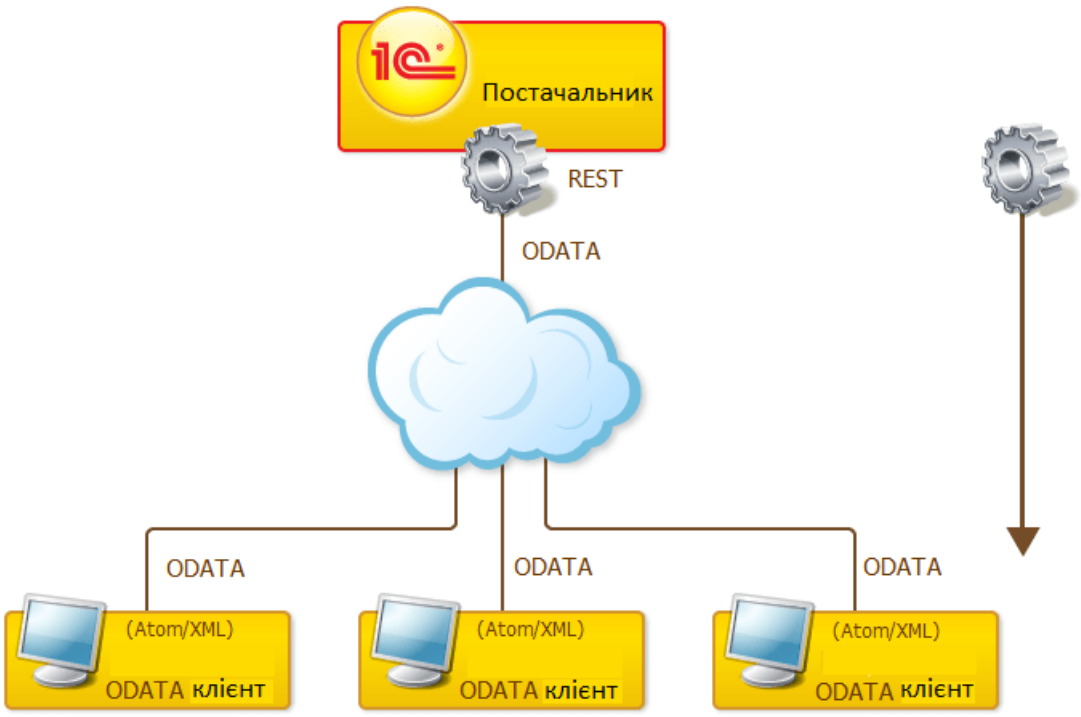


Рисунок 2.15 — Схема роботи RESTAPI 1C

Для того, щоб підтримувати дані в системі в актуальному стані, необхідно регулярно оновлювати їх через інтеграцію з 1С. Це можна здійснювати за допомогою таких методів оновлення: регулярні запити до 1С Система налаштовує автоматичні запити до API або імпортує дані з 1С щомісяця або через певний проміжок часу, ця схема представлена на рис. 2.15. Синхронізація даних у режимі реального часу Для того, щоб дані завжди були актуальними, система може перевіряти та оновлювати їх у режимі реального часу або з фіксованою частотою (наприклад, кожні 5 або 10 хвилин). Такий підхід гарантує, що всі зміни, які відбуваються в 1С, своєчасно відображаються в системі, що дозволяє уникнути неточностей і непорозумінь під час оформлення виплат.

2.7.2 Використання API для зовнішньої верифікації користувачів

Верифікація користувачів є важливим кроком у процесі надання пільг у вебсистемі, і важливо забезпечити точність та достовірність інформації про кожного

користувача. З цією метою наша система інтегрована з зовнішніми API серверів Applox, які дозволяють перевіряти дані користувачів в автоматизованому режимі. Такі API дають змогу перевірити особу користувача, право на пільги та інші важливі параметри, гарантуючи високий рівень безпеки та знижуючи ризик шахрайства.

Використання API для зовнішньої автентифікації користувачів гарантує, що дані в системі є актуальними та точними. Зовнішні системи Applox надають API, за допомогою яких веб-системи Applox можуть взаємодіяти з даними користувачів, перевіряти права та здійснювати перехресну перевірку інформації на різних платформах. API для верифікації зазвичай реалізується за допомогою стандартизованого веб-протоколу, такого як Representational State Transfer (REST), який використовує HTTP-запити для отримання або надсилання даних.

У цьому випадку взаємодія між нашою системою та Applox відбувається за допомогою GET, POST, PUT або DELETE запитів. Весь обмін інформацією відбувається у таких форматах, як JSON або XML, що полегшує обробку та передачу даних.

Процес верифікації користувачів через API включає кілька основних етапів:

1. Ідентифікація користувача: Першим кроком є ідентифікація користувача, яка може здійснюватися на основі введених користувачем даних, таких як ІПН (Ідентифікаційний код), серія та номер паспорта або інші індивідуальні дані. Ця інформація передається в запитах до API сервера Uprox для подальшої перевірки.
2. Перевірка даних користувача: API сервера Uprox надає інформацію про користувача, наприклад, його статус (активний чи неактивний), права на пільгу, наявність заборгованості або інших обмежень. Після того як інформація буде отримана від Uprox, система порівнює її з внутрішніми даними (наприклад, з даними, що зберігаються в базі даних 1С) для верифікації достовірності.
3. Аналіз результатів перевірки: Отримані дані від API аналізуються. Якщо користувач відповідає всім умовам для надання пільг, система

автоматично відновлює або надає пільгу. Якщо користувач не відповідає вимогам, система відмовляє у наданні пільги. Це дозволяє знизити ймовірність помилок та шахрайства.

4. Обробка винятків і помилок: Верифікація через API не завжди проходить без помилок або затримок. Це може бути викликано проблемами на стороні зовнішнього сервісу (Uprox), помилками в даних або мережею. У таких випадках система повинна мати механізми обробки помилок та відновлення, щоб гарантувати надійність роботи, навіть якщо дані не можуть бути отримані або перевірені вчасно.
5. Актуалізація внутрішніх даних: Після отримання результатів верифікації, система автоматично оновлює свої дані. Це включає оновлення статусу користувача, коригування його прав на пільгу або навіть видалення користувача з системи, якщо він більше не має права на отримання пільг.

Для забезпечення ефективного і надійного обміну даними між системами важливо використовувати стандартизовані протоколи і формати даних, що сприяють їх сумісності та простоті інтеграції. У нашому випадку для інтеграції з сервером Uprox ми використовуємо:

1. Протокол HTTP — це стандартний протокол для передачі даних через Інтернет, який підтримується більшістю API. HTTP-запити, такі як GET, POST і PUT, використовуються для отримання, оновлення та видалення даних.
2. RESTful API — REST (Representational State Transfer) може бути використаний для того, щоб зробити взаємодію між веб-системами простою і REST дозволяє створювати прості запити, придатні для маніпулювання даними користувача, приклад запиту зображений на рис. 2.16.
3. JSON або XML — дані, що передаються через API, можуть бути представлені у форматі JSON або XML. Обидва ці формати легко піддаються аналізу і можуть бути легко інтегровані з більшістю мов

програмування; JSON є найпоширенішим форматом для веб-сервісів завдяки своїй легкості і простоті обробки.

Рисунок 2.16 — Приклад запиту та відповіді API

2.7.3 Синхронізація даних між платформами

Синхронізація даних між різними платформами є важливим кроком для забезпечення актуальності та точності інформації в системі. У нашому випадку основною метою синхронізації є узгодження даних між локальною веб-системою та зовнішньою платформою, наприклад, сервером Applox або базою даних 1С. Синхронізація даних між платформами передбачає автоматичне оновлення інформації в режимі реального часу або через певний проміжок часу. Це дозволяє уникнути ситуацій, коли система містить застарілі або неточні дані, які можуть призвести до помилкових призначень або відмов у призначенні допомоги.

Кожен етап синхронізації передбачає отримання найновіших даних із зовнішнього джерела, їх обробку та внесення змін до локальної бази даних. Важливо також забезпечити двосторонній обмін даними, щоб усі зміни, які відбуваються на одній платформі, автоматично відображалися на іншій платформі. Основний процес синхронізації передбачає інтеграцію із зовнішнім uplox-сервером. Сервер uplox отримує інформацію про користувача, права на пільги, статус та інші важливі параметри. Ці дані синхронізуються з нашою локальною базою даних, яка зберігає інформацію про пільги, транзакції та інші облікові записи користувачів.

Регулярне оновлення даних гарантує, що користувачі отримують пільги лише на основі найактуальнішої інформації, а зміни у пільгах чи статусі користувача виявляються вчасно. Для досягнення ефективної синхронізації використовуються різні методи. Один з них - регулярні запити до API, який отримує регулярно оновлювані дані з серверів Applocs та 1С. Ці запити можуть виконуватися автоматично щомісяця або з фіксованою частотою, залежно від того, як часто змінюється інформація про користувача. Таким чином, забезпечується постійна актуальність і точність необхідних даних. Синхронізація також може відбуватися в режимі реального часу. Сингулярність зібраних платформ видно на рис. 2.17.

Рисунок 2.17 — Синхронізація даних між платформами

У цьому випадку щоразу, коли відбувається запит користувача або дія, пов'язана з пільгою, інформація в базі даних миттєво оновлюється, що зводить до мінімуму час, протягом якого користувачі отримують застарілі дані та пільги. Такий підхід особливо важливий там, де термін дії пільги або її умови можуть часто змінюватися, а точність даних є критично важливою. Щоб забезпечити синхронізацію між платформами, системи повинні включати механізми обробки помилок. Якщо в процесі обміну даними виникають проблеми з доступом до зовнішніх сервісів або

отриманням даних, система повинна мати можливість автоматично повторити запит або відкласти операцію до моменту, коли обмін даними стане можливим.

Така система зменшить ймовірність втрати критично важливої інформації або виходу системи з ладу. Важливою частиною синхронізації є моніторинг змін, що відбуваються на зовнішніх платформах. Якщо дані змінюються на серверній стороні UPROCS або в базі даних 1С, наша система повинна скоригувати записи відповідно до нової інформації. Це дозволяє уникнути ситуацій, коли користувачі отримують пільги шахрайським шляхом або, навпаки, безпідставно отримують відмову в пільгах через застарілі або неточні дані. Для забезпечення високого рівня безпеки під час синхронізації даних використовуються найсучасніші методи шифрування та автентифікації.

Це включає захист під час передачі даних між платформами за допомогою протоколу HTTPS, який запобігає перехопленню інформації, і використання механізмів аутентифікації для доступу до зовнішніх API. Ці заходи захищають персональні дані користувачів і забезпечують конфіденційність при обміні чутливою інформацією. Правильна синхронізація даних між платформами гарантує, що система завжди працює на основі актуальної та точної інформації. Це не тільки автоматизує процес обробки пільг і перевірки прав користувачів, а й забезпечує високий рівень безпеки та надійності, що є важливим для систем, які обробляють персональні дані та фінансові операції.

2.8 Тестування та забезпечення якості

Щоб забезпечити надійність і стабільність системи, всі компоненти повинні бути ретельно протестовані перед розгортанням у виробничому середовищі. Це включає в себе автоматизоване тестування, моніторинг продуктивності та управління безпекою системи. У цьому випадку система складається з двох основних

частин: одна, яка працює в закритій мережі для обробки виплат і публікації продуктів, а інша - для редагування контенту через відкритий мережевий інтерфейс адміністратора.

2.8.1 Використання автоматизованих тестів для перевірки API

Для забезпечення стабільності та безпеки взаємодії між різними частинами системи активно використовується автоматизоване тестування, зокрема для тестування API. Автоматизоване тестування дозволяє значно прискорити процес перевірки якості та коректності роботи API за рахунок перевірки функціональності API та коректності відповідей на різні запити. Для тестування API в цій системі використовуються наступні підходи:

1. Тести для перевірки основних функцій API: Тести перевіряють правильність відповідей на стандартні запити, наприклад, на отримання даних про користувачів, валідацію доступу до пільг, а також створення або оновлення записів. Кожен запит перевіряється на коректність відповіді, наявність помилок та дотримання стандартів (JSON, XML тощо).
2. Тести для перевірки безпеки API: Це тести на наявність уразливостей, таких як SQL-ін’єкції, CSRF-атаки, а також правильність роботи з аутентифікацією (JWT, OAuth) і авторизацією користувачів. Для таких тестів застосовуються спеціалізовані інструменти, наприклад, Postman або SoapUI, а також фреймворки для автоматизованого тестування API, як-от JUnit з MockMvc або RestAssured.
3. Тести на навантаження та стрес-тестування API: Важливо перевірити, як API буде працювати під великим навантаженням, коли кількість запитів до сервера зростає. Для цього використовуються інструменти

для тестування навантаження, такі як JMeter або LoadRunner, що дозволяють симулювати велику кількість одночасних запитів і перевіряти час відгуку, пропускну здатність та стабільність роботи API.

2.8.2 Моніторинг продуктивності в реальному часі

Моніторинг продуктивності системи в режимі реального часу є важливим елементом обслуговування системи. Оскільки веб-система складається з двох частин, одна з яких працює в закритій мережі, а інша - у відкритій мережі для редагування контенту, важливо відстежувати і контролювати продуктивність кожної частини. Методи моніторингу продуктивності в режимі реального часу включають моніторинг сервера та моніторинг інфраструктури: віртуальні машини, на яких розміщений веб-сайт, відстежуються за допомогою таких інструментів, як Zabbix і Nagios.

Ці інструменти дозволяють відстежувати використання процесора, пам'яті та диска, мережеве підключення та інші важливі показники. Моніторинг веб-додатків Такі інструменти, як New Relic та AppDynamics, відстежують продуктивність окремих компонентів веб-систем. Це дозволяє контролювати час відгуку на запити, швидкість виконання критично важливих операцій і виявляти потенційні вузькі місця в продуктивності. Логування та аналіз: такі інструменти, як ELK Stack (Elasticsearch, Logstash та Kibana) та Splunk використовуються для збору та аналізу логів. Журнали дозволяють відстежувати помилки, аномалії та інші події, які можуть вплинути на систему.

2.9 Оновлення та підтримка системи

Оновлення та підтримка системи є важливою частиною її життєвого циклу. Це включає регулярні оновлення, підтримку нових версій, резервне копіювання та відновлення даних, а також моніторинг після розгортання.

2.9.1 Механізми оновлення та релізу нових версій

Веб-система використовує систему контролю версій GIT для управління змінами коду та забезпечення швидких оновлень. Кожна нова версія системи містить виправлення помилок, нові функції або оновлення безпеки.

1. Розробка та тестування нових версій: Кожен реліз проходить стадії розробки, тестування та перевірки на всіх етапах життєвого циклу. Перед публікацією нової версії проводяться автоматизовані та інтеграційні тести для перевірки сумісності всіх компонентів.
2. Реліз через GIT: Використання GIT для контролю версій дозволяє розробникам ефективно управляти змінами в коді і легко інтегрувати нові функції без ризику порушення роботи системи. Для оновлення застосовуються методи CI/CD (Continuous Integration / Continuous Deployment), що забезпечує автоматичне впровадження змін після проходження всіх етапів тестування.
3. Деплоймент на IIS Express: Для деплойменту веб-сайтів, розміщених на віртуальних машинах, використовується IIS Express, що дозволяє безперешкодно розгорнути нову версію веб-додатку на сервері. Оновлення відбувається з мінімальним часом простою, завдяки використанню процесу автоматизованого деплойменту.

2.9.2 Стратегії резервного копіювання та відновлення даних

Резервне копіювання та відновлення даних є важливими компонентами стратегії підтримки веб-системи, щоб забезпечити безпеку і доступність даних у разі їх втрати чи пошкодження.

1. Резервне копіювання бази даних: Всі дані, що зберігаються в базі даних, регулярно копіюються на зовнішні сервери. Це дозволяє забезпечити їх безпеку і можливість швидкого відновлення у разі несанкціонованого доступу або відмови системи.
2. Резервне копіювання конфігураційних файлів і кодових репозиторіїв: Використання GIT для зберігання всього коду дозволяє мати резервні копії конфігураційних файлів, а також сценаріїв та налаштувань для кожної версії системи.
3. Відновлення даних: У разі пошкодження або втрати даних застосовуються стратегії відновлення. Це може включати автоматичне відновлення з резервних копій бази даних або конфігураційних файлів, а також відновлення коду через GIT.

2.9.3 Моніторинг і підтримка в процесі експлуатації

Після впровадження веб-системи важливо забезпечити моніторинг її працездатності та своєчасне виявлення потенційних проблем. Це включає моніторинг серверів, бази даних, а також самих веб-додатків для виявлення помилок або перевантажень.

1. Моніторинг роботи серверів та додатків: За допомогою Zabbix або New Relic здійснюється постійний моніторинг роботи серверів і додатків, що дозволяє оперативно реагувати на будь-які збої чи зниження продуктивності.

2. Аналіз продуктивності та помилок: Використання систем логування (ELK Stack або Splunk) дозволяє аналізувати продуктивність додатків і виявляти помилки або проблеми, що можуть виникнути під час експлуатації системи.

3 РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ НАДАННЯ ПІЛЬГОВОЇ ПРОДУКЦІЇ

3.1 Загальний огляд реалізації системи

Розробка веб-системи для автоматизації процесу видачі пільгових продуктів базується на відкритих API та інтеграції з базами даних для забезпечення адміністративної ефективності та прозорості. Система складається з двох основних веб-сайтів, закритого для адміністраторів та відкритого для користувачів, які автоматизують процес видачі пільгових продуктів, перевіряють права користувачів та відстежують події в режимі реального часу. Важливою складовою системи є інтеграція з базою даних, яка зберігає інформацію про користувачів, права доступу та історію видачі продуктів. Ця база даних також взаємодіє із зовнішніми API, такими як Applox та 1C, для перевірки достовірності даних та статусу користувача. Після перевірки інформації подія зчитується за допомогою спеціального пристрою, такого як картка або перепустка, і пільгові продукти автоматично видаються через інтерфейс.

Користувачів, які мають право на пільги, можна додавати і видаляти через інтерфейс адміністрування, підтримуючи список пільговиків в актуальному стані. Модулі для завантаження та управління списком користувачів дозволяють змінювати дані пільговиків і контролювати доступ до продуктів, мінімізуючи можливість помилок і зловживань.

Рішення забезпечує автоматизований контроль доступу, спрощуючи роботу адміністраторів і підвищуючи ефективність процесів. Для розробки цієї системи були використані найсучасніші технології та інструменти, включаючи серверний фреймворк, який підтримує інтеграцію з API та базами даних. Такий вибір технологій забезпечує гнучкість і масштабованість системи, дозволяючи розширювати її функціонал за потреби без значних витрат часу і ресурсів. Візуально система розділена на два основних інтерфейси. Один - це інтерфейс, доступний для

адміністраторів, де вони можуть керувати даними користувачів та переглядати інформацію про надані пільги. Інший - інтерфейс для кінцевих користувачів, який дозволяє їм перевіряти наявність пільг та отримувати продукти відповідно до встановлених критеріїв.

Наступним етапом розробки буде інтеграція з API Applocs та API 1C, що дозволить системі взаємодіяти з державними реєстрами та іншими зовнішніми ресурсами для перевірки прав користувачів. Ця інтеграція уможливить автоматичне отримання актуальних даних без необхідності ручного введення, що значно скоротить час обробки заявок та підвищить точність системи. На рисунку 3.1 видно головну точку входу в адміністративну частину програми.

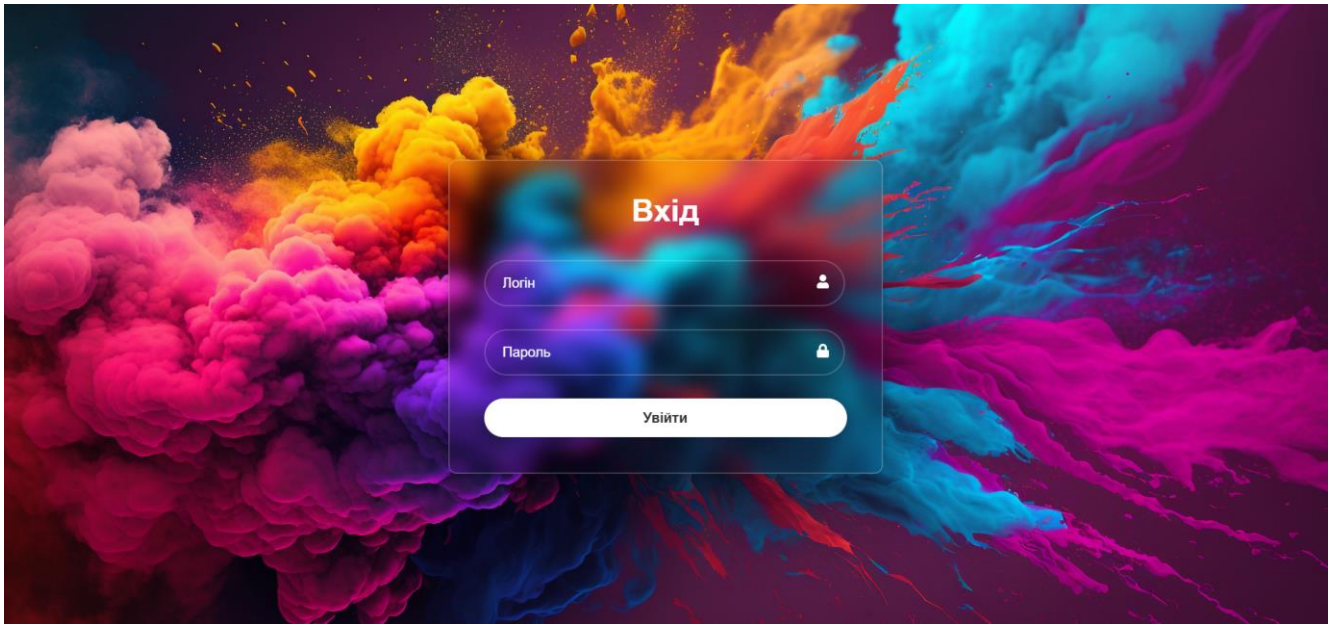


Рисунок 3.1 — Зображення головної строінки веб-системи

3.1.1 Опис основних компонентів

Розроблена веб-система для автоматизації процесу надання пільгових продуктів включає кілька ключових компонентів, які забезпечують ефективне функціонування процесів видачі продуктів, зберігання даних та перевірки прав доступу.

Основні компоненти системи та їх інтеграція перераховані нижче: веб-сайт адміністратора: інтерфейс адміністратора дозволяє керувати базою даних користувачів, контролювати пільги та перевіряти статус доступу кожного користувача в режимі реального часу. Адміністратори можуть додавати нових користувачів до списку пільговиків, перевіряти їхні права доступу та контролювати процес видачі пільг. Інтерфейс адміністрування реалізований з використанням новітніх веб-технологій, що дозволяє швидко реагувати на зміни та зручно керувати даними. Веб-сайт користувача дозволяє кінцевим користувачам перевіряти статус своєї пільги та наявність виданих товарів. Через інтерфейс користувачі можуть перевірити свої дані, отримати інформацію про наявні пільги та замовити товари; після успішної перевірки через API та базу даних система автоматично підтверджує або відхиляє пільгову товарну пропозицію. Інтерфейс користувача розроблений таким чином, щоб забезпечити максимальну зручність та швидке обслуговування, реалізація зображена на рис. 3.2.

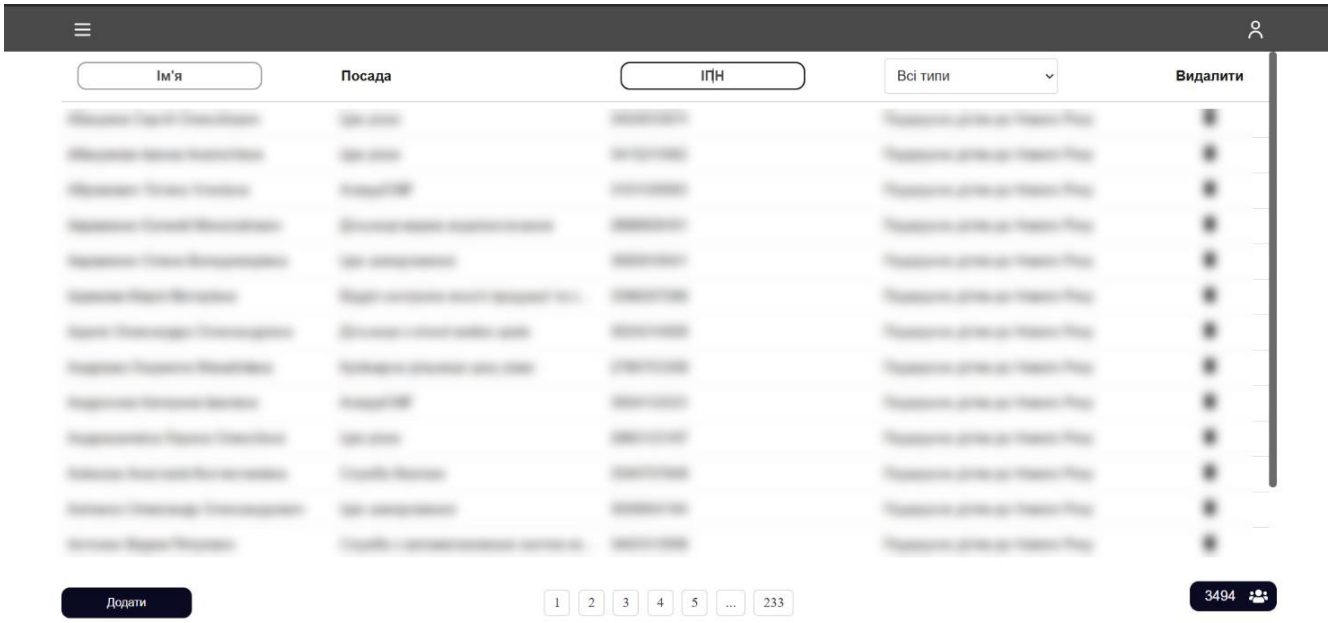


Рисунок 3.2 — Зображення головноъ сторінки веб-системи

База даних системи містить всі ключові дані про пільговиків, історію доступу до пільгових продуктів та інформацію про транзакції. Вона інтегрована з API Uprox та 1С і дозволяє в режимі реального часу перевіряти статус користувача та його права доступу. База даних має високу продуктивність і надійність, що дозволяє

ефективно обробляти великі обсяги даних і швидко отримувати до них доступ, список таблиць зображений на рис. 3.3.

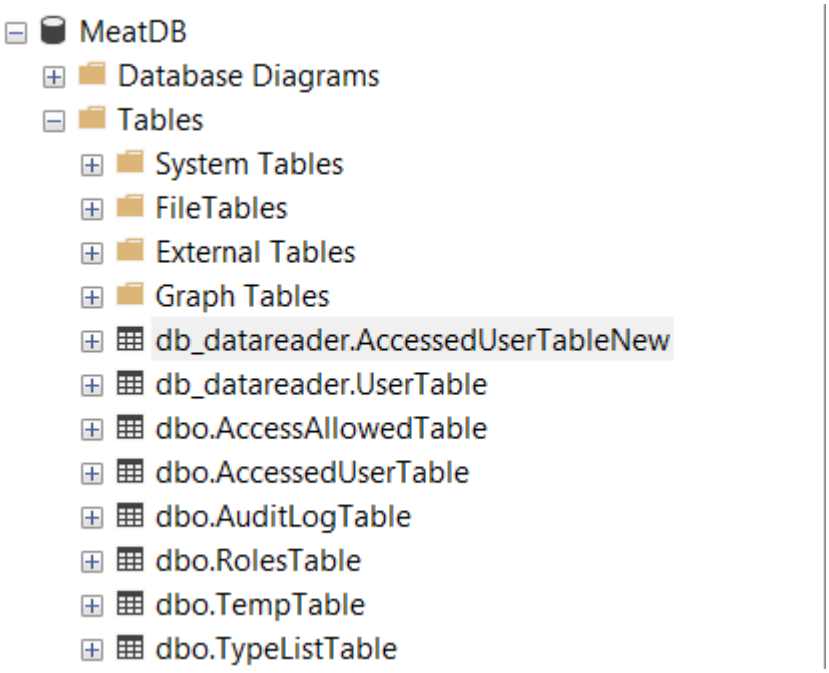


Рисунок 3.3— Зображення таблиць бази даних

Інтеграція із зовнішніми API (Uprox та 1C): є важливою частиною процесу автоматизації. Веб-системи підключені до API Uprox і можуть перевіряти дані користувачів, щоб побачити статус бенефіціара та право на товари в режимі реального часу. API 1C інтегрований для обміну даними між внутрішніми системами бухгалтерського обліку, що дозволяє здійснювати контроль і облік руху товарів. контроль та облік руху товарів, як, наприклад, на рис. 3.4.

```
public async Task<bool> Insert1CUsers(int IdTypeList)
{
    Console.OutputEncoding = Encoding.UTF8;

    HttpClientHandler handler = new HttpClientHandler();
    string tokenOfReport = "ee5af690-0628-11ef-98c1-00505691a3b3";

    DateTime now = DateTime.Now;
    DateTime firstDayOfLastMonth = now.AddMonths(-1).Date;
    firstDayOfLastMonth = new DateTime(firstDayOfLastMonth.Year, firstDayOfLastMonth.Month, 1);
    DateTime lastDayOfLastMonth = now.Date;
    lastDayOfLastMonth = new DateTime(lastDayOfLastMonth.Year, lastDayOfLastMonth.Month, 1).AddDays(-1);
    string beginPeriod = firstDayOfLastMonth.ToString("yyyy-MM-dd");
    string endPeriod = lastDayOfLastMonth.ToString("yyyy-MM-dd");

    using (HttpClient client = new HttpClient(handler))
    {
```

Рисунок 3.4 — Демонстрація частини коду запиту до 1C

Модуль аутентифікації та контролю доступу, система забезпечує безпеку даних завдяки інтеграції з такими механізмами аутентифікації, як OAuth та JWT, приклад використання JWT зображено на рис. 3.5. Ці технології забезпечують безпечний доступ до системи як для адміністраторів, так і для користувачів. Механізми контролю доступу реалізовані через систему рольового контролю доступу (RBAC), яка дозволяє користувачам чітко визначити, хто має право виконувати певні операції в системі.

```
public string GenerateJwtToken(UserModel user, RoleModel roles)
{
    try
    {
        Claim[] claims =
        {
            new("userEmail", user.Email.ToString()),
            new(ClaimTypes.Role, roles.Caption.ToString())
        };

        var signingCredentials = new SigningCredentials(
            new SymmetricSecurityKey(Encoding.UTF8.GetBytes(
                "tessstkeytessstkeytessstkeytessstkeytessstkeytessstkey")),
            SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            claims: claims,
            signingCredentials: signingCredentials,
            expires: DateTime.UtcNow.AddHours(1));

        var tokenValue = new JwtSecurityTokenHandler().WriteToken(token);

        return tokenValue;
    } catch
    {
        return null;
    }
```

Рисунок 3.5 — Зображення коду генерації JWT токену

Веб-система відстежує всі події, пов'язані з випуском пільгових продуктів. Інтерфейс адміністратора дозволяє в режимі реального часу контролювати поточні процеси, такі як перевірка прав доступу, видача продуктів та інші важливі транзакції. Всі події реєструються для забезпечення можливості аудиту та виявлення можливих проблем на ранній стадії. Ці компоненти забезпечують повну автоматизацію процесу надання пільгових товарів, від перевірки прав доступу до фактичної

видачі товару. Веб-система розроблена таким чином, щоб бути гнучкою і масштабованою, легко адаптуватися до потреб різних організацій і зростаючого обсягу оброблюваних даних, збереження логів зображено на рисунку 3.6.

	Id	UserId	ActionTime	Action	IpAddress	AdditionalInfo
1	9	1	31.07.2024 20:12:35	RestorePassword	[redacted]	[redacted]
2	10	1	31.07.2024 20:12:37	RestorePassword	[redacted]	[redacted]
3	11	4	31.07.2024 22:29:03	ChangePassword	[redacted]	[redacted]
4	12	1	31.07.2024 22:29:29	RestorePassword	[redacted]	[redacted]
5	13	1	31.07.2024 22:30:02	RestorePassword	[redacted]	[redacted]
6	14	1	31.07.2024 22:30:23	DeleteUser	[redacted]	[redacted]
7	15	1	01.08.2024 07:35:02	DeleteUser	[redacted]	[redacted]
8	16	1	01.08.2024 07:35:51	PushFrom1C	[redacted]	[redacted]
9	17	1	01.08.2024 08:44:13	DeleteUser	[redacted]	[redacted]
10	18	5	01.08.2024 08:55:50	ChangePassword	[redacted]	[redacted]
11	19	5	01.08.2024 09:01:05	PushFrom1C	[redacted]	[redacted]
12	20	5	01.08.2024 09:03:38	DeleteAccess	[redacted]	[redacted]
13	21	5	01.08.2024 09:04:34	AddAccess	[redacted]	[redacted]
14	22	5	01.08.2024 09:14:47	PushFrom1C	[redacted]	[redacted]
15	23	5	01.08.2024 11:59:54	PushFrom1C	[redacted]	[redacted]
16	24	6	01.08.2024 13:39:21	AddAccess	[redacted]	[redacted]
17	25	6	01.08.2024 13:42:15	PushFrom1C	[redacted]	[redacted]
18	26	7	01.08.2024 13:42:39	PushFrom1C	[redacted]	[redacted]
19	27	6	01.08.2024 13:47:24	DeleteAccess	[redacted]	[redacted]
20	28	6	01.08.2024 13:47:31	DeleteAccess	[redacted]	[redacted]
21	29	1	02.08.2024 15:25:56	PushFrom1C	[redacted]	[redacted]
22	30	1	02.08.2024 15:28:07	AddAccess	[redacted]	[redacted]
23	31	1	02.08.2024 15:28:18	DeleteAccess	[redacted]	[redacted]

Рисунок 3.6 — Зображення аудит логів в таблиці

3.1.2 Використані технології та інструменти

Для розробки веб-системи для автоматизації процесу надання пільгових продуктів було обрано низку потужних сучасних технологій, які забезпечать високу продуктивність, безпеку та масштабованість системи. Однією з основних технологій є Node.js - JavaScript-рушій, який дозволяє коду працювати на сервері; Node.js обробляє запити асинхронно, тому навіть великі обсяги даних можуть оброблятися без значних затримок, роблячи високопродуктивні, масштабовані веб-додатки Він ідеально підходить для створення високопродуктивних, масштабованих веб-додатків. Для спрощення розробки на стороні сервера використовується Express.js - мінімальний веб-фреймворк для Node.js, який дозволяє швидко налаштовувати маршрутизацію запитів, обробку HTTP-запитів та інтеграцію з іншими системами через API. React.js використовується для створення динамічних та інтерактивних користувацьких інтерфейсів. Це потужний JavaScript-фреймворк для створення одно-

сторінкових веб-додатків, що дозволяє користувачеві створювати швидкі, інтерактивні інтерфейси, де дані оновлюються без перезавантаження сторінки. React можна використовувати для ефективного управління станом додатку та створювати інтерфейси, які є зручними для користувача та швидко реагують на зміни. Дані зберігаються та обробляються за допомогою MySQL, реляційної бази даних, яка забезпечує надійність та високу продуктивність при роботі з великими обсягами структурованих даних. MySQL підтримує складні запити та транзакції і гарантує цілісність даних. Це дуже важливо для систем, які обробляють конфіденційну інформацію, таку як користувацькі та транзакційні дані.C# з фреймворком ASP.NET використовується для побудови серверної частини, включаючи розробку API та бізнес-логіки. Технологія надає широкі можливості для створення надійних і масштабованих серверних додатків. ASP.NET пропонує високу продуктивність, зручне управління запитами і відповідями, а також підтримує різні механізми безпеки та інтеграцію з іншими системами, такими як бази даних і сторонні API. Він підтримує інтеграцію з іншими системами, такими як бази даних і сторонні API. Ці технології забезпечують гнучкість у розробці, дозволяють інтегрувати різні системні компоненти та гарантують високий рівень безпеки та продуктивності, що є необхідним для ефективної автоматизації процесу надання пільг.

3.2 Реалізація серверної частини

Реалізація серверної частини веб-системи, що автоматизує процес пропозиції пільгових продуктів, є важливим етапом, оскільки вона забезпечує зв'язок між усіма компонентами системи, обробляє запити та взаємодіє із зовнішніми сервісами через бази даних та API. Серверна частина системи реалізована з використанням Node.js та C# ASP.NET для забезпечення високої продуктивності та масштабованості. Для управління обробкою та маршрутизацією запитів використано Express.js, що спрощує налаштування серверної логіки та взаємодію з клієнтською частиною. Кожен запит від клієнта проходить через сервер, де обробляється бізнес-

логіка, перевіряються дані та ініціюється відповідна дія, наприклад, перевірка доступу до пільгових продуктів або передача даних до бази даних. Оскільки в системі передбачена інтеграція із зовнішніми API, такими як API Uprox для перевірки користувачів та їх прав, на серверній стороні міститься логіка запитів до цих API Використовуючи HTTP запити до API, система отримує необхідну інформацію в режимі реального часу через Для забезпечення безпеки та коректного виконання запитів використовуються різні методи аутентифікації та авторизації, такі як OAuth та JWT, що гарантує безпечний доступ до зовнішніх сервісів та запобігає несанкціонованому доступу до критично важливих даних.Взаємодія з базою даних MySQL побудована на основі ефективних реалізована на стороні сервера за допомогою спеціальних ORM-бібліотек, які дозволяють виконувати запити до бази даних, надають швидкий доступ до інформації про користувача, доступ до товарів зі знижками та історії транзакцій.

Операції з базою даних включають додавання, видалення та оновлення записів, а також перевірку прав доступу та управління сесіями користувачів. Крім того, на стороні сервера реалізована логіка обробки та перевірки даних, що гарантує актуальність та коректність поданої інформації. У разі виникнення помилок або некоректних запитів система генерує відповідні повідомлення про помилки, щоб допомогти користувачам і адміністраторам швидко відреагувати на проблему. Реалізація на стороні сервера також включає налаштування механізмів масштабування для забезпечення високої доступності та продуктивності системи. Серверну інфраструктуру можна масштабувати за допомогою хмарних сервісів, щоб обробляти великі обсяги запитів і даних навіть в умовах високого навантаження. Система налаштована таким чином, щоб забезпечити відмовостійкість та автоматичне відновлення в разі збою або відмови, тим самим забезпечуючи безперервну роботу всіх функцій. Таким чином, серверна частина забезпечує належну інтеграцію між компонентами системи та гарантує безпеку, ефективну обробку запитів і надійність веб-системи, центральне місце входу зображено на рисунку 3.7.

```
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseDefaultFiles();
app.UseStaticFiles();
app.UseRouting();

app.UseCors("AllowSpecificOrigin");


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

//app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();

app.MapFallbackToFile("/index.html");

app.Run();
```

Рисунок 3.7 — Місце входу в серверну частину

3.2.1 Розподіл між двома сайтами

Веб-система, розроблена для автоматизації надання пільгових товарів, використовує два веб-сайти з різними функціональними ролями: перший сайт використовується для управління доступом користувачів до пільгових товарів; другий сайт використовується власниками магазинів для видачі товарів безпосередньо кінцевим споживачам Використання. Перший сайт (інтерфейс адміністрування) дозволяє адміністраторам та уповноваженим працівникам налаштовувати доступ до товарів зі знижками. Він використовується для реєстрації користувачів, перевірки права на пільги та ведення списку тих, хто має право на отримання товарів. Сайт інтегрується з базами даних і зовнішніми API для перевірки користувачів, зміни їхнього статусу доступу, додавання нових користувачів і видалення їх зі списку

одержувачів. Адміністратори можуть редагувати права доступу на рівні окремих користувачів і груп користувачів, що дозволяє їм гнучко керувати доступом до товарів зі знижками. Другий сайт (для власників магазинів) призначений для управління процесом безпосередньої дистрибуції товарів. Цей інтерфейс дозволяє власникам магазинів перевіряти список бенефіціарів, перевіряти право на отримання товарів і видавати товари відповідно до визначених правил. Інтеграція з першою частиною системи дозволяє комірникам мати доступ до актуальної інформації про статус бенефіціара, а також реєструвати фактичну видачу товарів шляхом сканування штрих-кодів або NFC-перепусток. Оскільки ці два сайти відіграють різні ролі в системі, кожен з них має свою окрему зону доступу, що допомагає забезпечити безпеку та контроль на кожному етапі процесу обробки запиту. Сайт адміністрації зосереджений на адміністративних функціях, тоді як клерк бере участь лише в процесі видачі товарів, що зменшує ймовірність помилок і підвищує ефективність обробки запитів Розподіл функцій між двома сайтами гарантує, що ролі користувачів чітко розмежовані і що до різних частин системи можна отримати доступ належним чином. Цього можна досягти. Це важливо для ефективного та безпечного управління пільгами. Це дозволяє організувати процес видачі документів таким чином, щоб кожен етап автоматизації функціонував з найвищим ступенем точності та безпеки.

3.2.2 Інтеграція серверної частини через API

Одним із ключових аспектів розробки серверної частини веб-системи є інтеграція із зовнішніми та внутрішніми сервісами за допомогою API. Це забезпечує безперебійний зв'язок між різними компонентами системи та автоматизує процес обміну даними; інтеграція через API дозволяє ефективно синхронізувати дані між різними частинами системи, щоб інформація була актуальною та доступною для користувачів у режимі реального часу. Серверна частина системи перевіряє авто-

ризацію користувача та забезпечує автоматизацію процесу обміну даними. Серверна частина системи використовує API для виконання базових функцій, таких як перевірка прав користувача, доступ до баз даних та взаємодія з іншими зовнішніми сервісами. Наприклад, для перевірки статусу бенефіціара система може звертатися до зовнішніх API, таких як Uprox та 1C, щоб автоматично підтверджувати або відхиляти запити користувачів на отримання товарів. Це дозволяє швидко і точно обробляти заявки без необхідності ручної перевірки. API також відіграє важливу роль у зберіганні та оновленні даних у базі даних. через API сервер може отримувати і відправляти дані. Це дозволяє зберігати інформацію про бенефіціарів, їхнє право на отримання товарів та інші дані, важливі для функціонування системи. взаємодія через API забезпечує обмін даними в режимі реального часу, що має важливе значення для автоматизації процесу видачі товарів. Для забезпечення надійної інтеграції між системами використовуються такі стандарти та протоколи, як RESTful API та JSON. Це дозволяє робити запити та отримувати відповіді у зручному для системи форматі; використання RESTful API знижує складність інтеграції та підвищує гнучкість системи. Це пов'язано з тим, що ці API є стандартом для багатьох сучасних веб-сервісів. Важливим аспектом інтеграції є обробка помилок та забезпечення надійного зв'язку між системами. Якщо запит не вдалося виконати, система повинна мати механізм для повторної спроби, реєстрації помилки та інформування адміністратора про те, що є проблема з API. Це забезпечить стабільність роботи системи в разі виникнення непередбачуваних ситуацій, дивитись рис. 3.8.

```
[Authorize]
[HttpGet("alluser")]
0 references
public IEnumerable<AccessedUserModel> GetAllUser() ...

[Authorize]
[HttpGet("alltypes")]
0 references
public IEnumerable<TypeModel> GetAllTypes()...

[Authorize]
[HttpGet("allUproxUsers")]
0 references
public async Task<List<UproxUsersModel>> GetAllUproxUsers()...

[Authorize]
[HttpPost("deleteAccess/{id}")]
0 references
public async Task<IActionResult> DeleteAccess(string id)...
[Authorize(Roles = "Admin")]
[HttpPost("registration")]
0 references
public async Task<IResult> Registration(RegisterModel model)...
[Authorize]
[HttpPost("AddAccessUser")]
0 references
public async Task<IActionResult> AddAccessUser(AccessedUserModel model)...
```

Рисунок 3.8 — Демонстарція API

3.2.3 Безпека серверної частини

Внутрішня безпека веб-систем має вирішальне значення, особливо в частині обробки персональних даних та доступу до привілейованих продуктів. Для забезпечення належного рівня безпеки було впроваджено кілька важливих заходів для захисту даних, запобігання несанкціонованому доступу та гарантування цілісності системи.

Першим кроком у забезпеченні безпеки стало впровадження аутентифікації користувачів за допомогою протоколів OAuth 2.0 та JWT (JSON Web Token). Ці технології забезпечують надійне управління сесіями користувачів і дозволяють системі коректно ідентифікувати користувачів без зберігання аутентифікаційної інформації на сервері.

Використання OAuth і JWT забезпечує безпечний доступ до системи і запобігає компрометації паролів і зловмисному доступу до конфіденційних даних і запобігає зловмисному доступу до конфіденційних даних. Для захисту від SQL-ін'єкцій та інших видів атак на базу даних на стороні сервера використовуються підготовлені оператори та параметризовані запити. Такий підхід ефективно захищає систему від проникнення шкідливого SQL-коду, який може призвести до витоку даних або доступу до несанкціонованих ресурсів. Ризик атаки значно знижується, оскільки всі запити до бази даних виконуються лише після перевірки та фільтрації даних, що вводяться користувачем. Для забезпечення шифрування даних між клієнтом і сервером налаштована передача даних за протоколом HTTPS з використанням SSL/TLS сертифікатів. Це гарантує, що вся інформація, яка передається між користувачем і сервером, захищена від перехоплення сторонніми особами, приклад підключення на рисунку 3.9.

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
```

Рисунок 3.9 — Підключення OAuth

Ще одним важливим аспектом є контроль доступу до різних частин системи. Для цього реалізована рольова модель доступу, за допомогою якої можна визначити, хто може використовувати ті чи інші функції в системі. Наприклад, адміністратори мають доступ до всіх функцій системи, включаючи управління користувачами та зміну конфігурації, тоді як продавці можуть лише видавати товари та переглядати обмежені набори даних, як зображено на рисунку 3.10. Це запобігає випадковим або зловмисним змінам у системі та забезпечує належний доступ до інформації відповідно до ролей користувачів.

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminPolicy", policy => policy.RequireRole("Admin"));
    options.AddPolicy("EmployeePolicy", policy => policy.RequireRole("Employee"));

});
```

Рисунок 3.10 — Розподіл ролей

Для запобігання атакам типу «відмова в обслуговуванні» (DoS) існує система обмеження швидкості, яка обмежує кількість запитів від певного користувача або IP-адреси протягом певного періоду часу. Це захищає сервер від перевантаження і забезпечує стабільну роботу системи навіть при великих навантаженнях. Загалом, безпека на серверній стороні веб-системи була забезпечена комплексним підходом, що включає сучасні методи аутентифікації, шифрування, захист від атак на бази даних та налаштування відповідних засобів контролю доступу, реалізація на рисунку 3.11.

```
lic void Delete()

lock (_lockObject) //Об'єкт який блокує роботу методу з іншого потоку до тих пір поки метод не буде виконано повністю в по
{
    List<ModelEvent> allPosts = new List<ModelEvent>();
    if (_dataEventContext.Posts.Count != 0)
    {
        try
        {
            allPosts = _dataEventContext.Posts.ToList();
        }
        catch { }
    }

    state.isTrue = true;
    state.globalTrue = true;
    _dataEventContext.Posts.RemoveRange(0, allPosts.Count());
}
```

Рисунок 3.11 — Приклад блокування потоку на запит в методі

Ці заходи допомагають забезпечити безпечну та ефективну роботу системи і надають високий ступінь надійності для користувачів та адміністраторів.

3.3 Інтерфейс адміністрування

Інтерфейс адміністрування є важливою частиною веб-системи для автоматизації процесу надання пільгових продуктів, що дозволяє адміністраторам ефективно керувати користувачами, встановлювати права доступу та контролювати всі процеси, що відбуваються в системі. Основна мета інтерфейсу - надати зручний і зрозумілий інструмент управління, який дозволяє виконувати важливі операції і контролювати систему з мінімальними зусиллями.

3.3.1 Функціональність інтерфейсу адміністрування

Інтерфейс управління розроблений таким чином, щоб бути зручним та інтуїтивно зрозумілим, дозволяючи адміністраторам виконувати різні операції всередині системи. Основними функціями інтерфейсу є:

1. Управління користувачами: Адміністратор має можливість додавати нових користувачів, редагувати їх профілі, а також видаляти користувачів, які більше не мають доступу до системи. Кожен користувач має свій унікальний профіль, який включає інформацію про права доступу, статус та історію операцій.
2. Перегляд історії операцій: Інтерфейс дозволяє адміністраторам переглядати всі дії, що були виконані в системі, з можливістю фільтрувати і знаходити конкретні записи за різними критеріями (наприклад, за датою або користувачем). Це дає можливість контролювати,

які дії виконувалися, коли і яким користувачем, забезпечуючи прозорість і контроль над усіма операціями.

3.3.2 Робота з правами доступу

У системі реалізована рольова модель доступу, що дозволяє адміністратору визначати рівні доступу до різних функцій системи для кожного користувача. Це забезпечує контроль над тим, хто і яку інформацію може переглядати чи змінювати.

Ролі користувачів: У системі передбачено кілька типів ролей, кожна з яких має певний набір прав доступу. Наприклад, роль "Адміністратор" має повний доступ до всіх функцій, включаючи редагування налаштувань і управління користувачами. Роль "Комірник" має доступ лише до модуля видачі продукції і перегляду списків пільговиків, але не має права змінювати системні налаштування.

Налаштування прав доступу: Адміністратор може налаштувати права для кожної ролі, визначаючи, які функції доступні для кожного користувача, і які дії можуть бути виконані в межах їх повноважень. Це дозволяє мінімізувати ризики помилок або несанкціонованих змін, обмежуючи доступ до чутливих даних лише для певних осіб.

Ієрархія доступу: Для кожної ролі створюється ієрархія прав, яка гарантує, що користувачі можуть доступати тільки ту інформацію або функціональність, яка відповідає їхній ролі. Наприклад, користувачі з роллю "Комірник" не можуть змінювати дані пільговиків або налаштування API, вони можуть лише отримувати доступ до процесу видачі продукції.

Аудит доступу: Для забезпечення контролю над правами доступу в системі реалізована можливість аудитування всіх змін у правах доступу. Це дозволяє адміністратору в будь-який час перевірити, хто і які зміни вносив до налаштувань прав доступу, що підвищує рівень безпеки і прозорості роботи системи.

Завдяки впровадженій моделі прав доступу та налаштувань ролей, система гарантує, що користувачі можуть працювати тільки з тим функціоналом, для якого

вони мають відповідні права, знижуючи ймовірність помилок або зловживань. Цей підхід забезпечує гнучкість у управлінні доступом і дозволяє зберігати безпеку і конфіденційність даних, дивитись рис. 3.12.

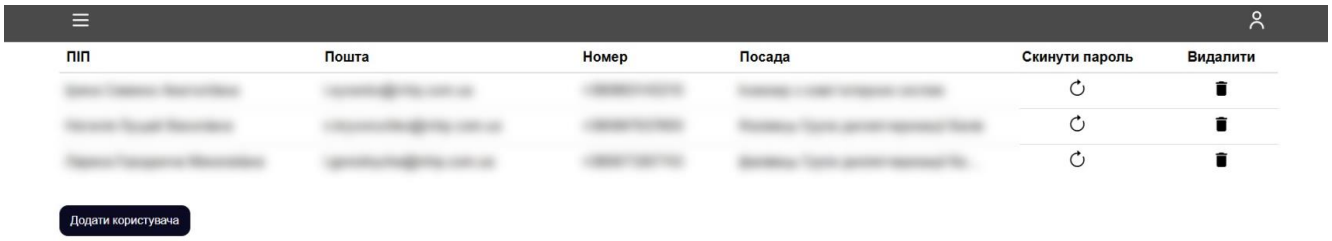


Рисунок 3.12 — Адміністративна панель

3.4 Реалізація модулю перевірки даних через API

Модуль валідації даних API є важливим елементом системи, що забезпечує інтеграцію із зовнішніми сервісами для перевірки та валідації інформації про користувачів та пільгові продукти. Використання відкритих API дозволяє системі швидко отримувати актуальні дані, перевіряти відповідність та забезпечувати достовірність наданої інформації.

3.4.1 Перевірка даних по API Uprox

Одним з основних API, з якими інтегрована система, є API Uprox. Цей сервіс використовується для перевірки даних користувачів у режимі реального часу, зокрема для перевірки персональних даних бенефіціарів та права на отримання пільгових продуктів. Процес перевірки Uprox полягає в тому, що система надсилає запит, що містить дані користувача (наприклад, ІПН або інші ідентифікаційні дані) до сервісу Uprox, який повертає підтвердження статусу користувача, тобто на-

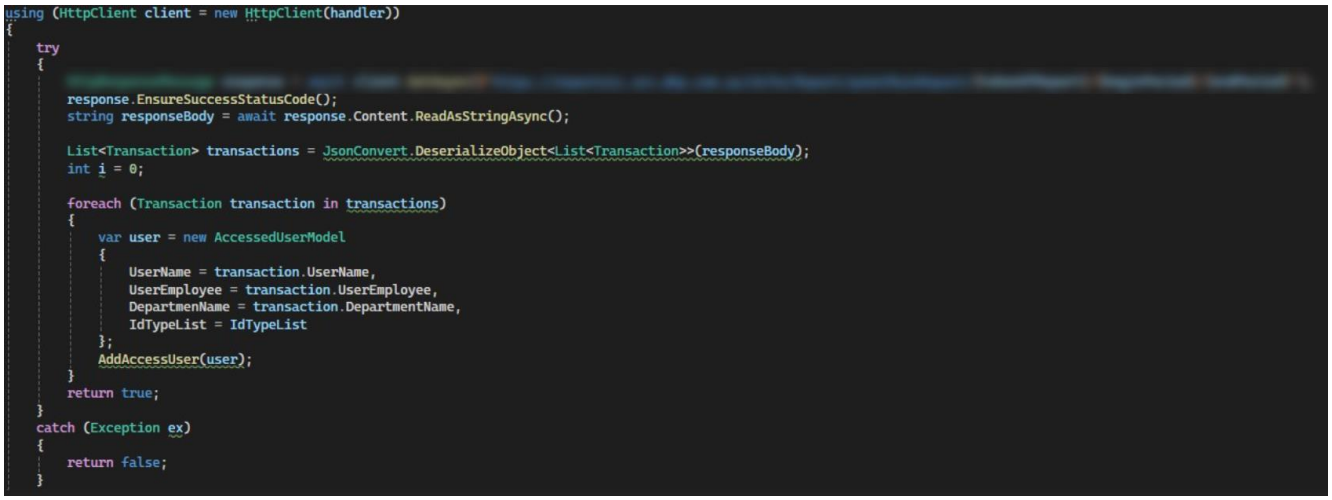
явність чи відсутність права на пільги. Запит до API Uprox складається з параметрів, які надсилаються сервісу через HTTPS. Це забезпечує шифрування даних і підвищену безпеку під час передачі інформації. Після отримання відповіді від Uprox система аналізує отриману інформацію та відображає результати перевірки. Після того, як користувач підтверджений як бенефіціар, процес видачі продукту може бути продовжений, приклад запиту зображено на 3.13.

Рисунок 3.13 — Процес вдправки запиту на API

3.4.2 Перевірка даних по API 1C

Ще одним важливим API для перевірки даних є інтеграція з 1C. Цей API використовується для валідації даних з обліку фінансових операцій та пільгових продуктів. Для цього система автоматично отримує дані з 1C. Ці дані включають інформацію про наявність, кількість та доступність пільгових продуктів. Перевірка даних здійснюється через API 1C шляхом створення запиту до відповідного облікового запису в 1C. Запит містить параметри, необхідні для перевірки конкретного товару або користувача. Система отримує відповідь від API, що містить актуальну інформацію про наявність товару зі знижкою та чи доступний він до видачі; відповідь від 1C повертається у форматі JSON, що дозволяє системі швидко обробити

дані та відобразити їх у відповідному інтерфейсі. Цей механізм дозволяє системі автоматично оновлювати статус товарів і користувачів без ручного втручання, реалізація зображена на рис. 3.14.



```
using (HttpClient client = new HttpClient(handler))
{
    try
    {
        response.EnsureSuccessStatusCode();
        string responseBody = await response.Content.ReadAsStringAsync();

        List<Transaction> transactions = JsonConvert.DeserializeObject<List<Transaction>>(responseBody);
        int i = 0;

        foreach (Transaction transaction in transactions)
        {
            var user = new AccessedUserModel
            {
                UserName = transaction.UserName,
                UserEmployee = transaction.UserEmployee,
                DepartmenName = transaction.DepartmentName,
                IdTypeList = IdTypeList
            };
            AddAccessUser(user);
        }
        return true;
    }
    catch (Exception ex)
    {
        return false;
    }
```

Рисунок 3.14 — Процес отримання та валідації даних з 1С

3.4.3 Обробка даних та валідація

Після того, як система отримує дані через API, вона повинна їх обробити та перевірити. Це важливий крок для забезпечення точності та коректності інформації із зовнішніх джерел. Валідація даних складається з декількох ключових етапів. Система перевіряє, чи відповідають отримані дані певному формату. Наприклад, якщо це номер ІПН, система перевіряє, чи є він дійсним відповідно до стандарту. Після перевірки формату даних система порівнює їх з даними, що вже зберігаються в базі, щоб перевірити на актуальність і точність. Для кожного користувача, залежно від результатів валідації через API uplocks або 1С, перевіряється право на отримання пільгових продуктів. Якщо валідація пройшла успішно, дані переходять на наступний етап обробки. Ця обробка включає внесення даних до бази даних, оновлення статусу користувача та автоматичне управління доступом до продуктів. Валідація є дуже важливим етапом, оскільки вона є основою для прийняття рішень щодо доступу користувачів до привілейованих продуктів, дивитись рис. 3.15.

```
public async Task<List<AccessedUserModel>> GetAllUpdated1CUsers(int IdTypeList)
{
    DateTime today = DateTime.Today;
    string dateString = $"{today.Month} {today.Year}";

    List<AccessedUserModel> accessedUsers = new List<AccessedUserModel>();

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string queryStr = @"SELECT * FROM MeatDB.db_datareader.AccessedUserTableNew
                    WHERE IsActive = @IsActive
                    AND IdTypeList = @IdTypeList
                    AND DateOfDiscount = @DateOfDiscount";

        SqlCommand command = new SqlCommand(queryStr, connection);
        command.Parameters.AddWithValue("@IsActive", "1");
        command.Parameters.AddWithValue("@IdTypeList", IdTypeList);
        command.Parameters.AddWithValue("@DateOfDiscount", dateString);

        await connection.OpenAsync();

        using (var sqlReader = await command.ExecuteReaderAsync())
        {
            while (sqlReader.Read())
            {
                try
                {
                    var user = new AccessedUserModel();
                    user.Id = (int)sqlReader["Id"];
                    user.UserName = sqlReader["UserName"].ToString();
                    user.UserEmployee = sqlReader["UserEmploee"].ToString();
                    user.DepartmenName = sqlReader["DepartmenName"].ToString();
                    user.IdTypeList = (int)sqlReader["IdTypeList"];
                    user.DateOfUpload = sqlReader["DateOfUpload"].ToString();
                    user.Is1C = (int)sqlReader["Is1C"];
                    user.IsActive = (int)sqlReader["IsActive"];
                    user.DateOfDiscount = sqlReader["DateOfDiscount"].ToString();

                    accessedUsers.Add(user);
                }
```

Рисунок 3.15 — Параметрезований запит в БД з валідацією даних

3.5 Модуль роботи з базою даних

3.5.1 Проектування та реалізація бази даних

Модуль бази даних є ключовим елементом для зберігання та обробки інформації в автоматизованій системі пільгового доступу до товарів. У ньому зберігаються дані користувачів, історія видачі товарів, інформація про пільгові товари та

статус доступу до них, виконання транзакцій та запити на швидке оновлення та обробку даних. Основні таблиці бази даних включають:

Таблиця користувачів: містить інформацію про бенефіціарів, їхні дані (ІПН, ПІБ, контактні дані) та статус доступу до пільгових продуктів. Таблиця продуктів містить інформацію про пільгові продукти та їх статус. Таблиця транзакцій фіксує всі операції з видачі продуктів користувачам, забезпечуючи прозорість і контроль процесу. Таблиця прав доступу визначає права кожного користувача на доступ до різних продуктів і послуг.

При розробці бази даних особливу увагу було приділено нормалізації, щоб уникнути дублювання даних та ефективно зберігати інформацію. Одним з найважливіших аспектів роботи з базами даних є оптимізація індексування та запитів. Оскільки система обробляє великі обсяги даних, індексація може значно підвищити швидкість виконання запитів до бази даних, особливо для таких операцій, як пошук користувачів, перевірка наявності пільгових товарів та перегляд історії транзакцій.

В базі даних були створені індекси на основні поля, які часто використовуються в запитах. Наприклад:

1. Індекс на полі ІПН в таблиці користувачів для швидкого пошуку за ідентифікаційним номером.
2. Індекс на полі статусу в таблиці продукції для фільтрації доступних пільгових продуктів.
3. Індекс на полі дати в таблиці транзакцій для швидкого сортування за часом видачі продукції.

3.5.2 Підтримка транзакцій та атомарності операцій

Для забезпечення точності та надійності обробки даних система підтримує атомарність транзакцій та операцій. Це важливо для запобігання ситуацій, коли одні операції виконуються успішно, а інші - ні. Використання транзакцій гарантує, що всі зміни в базі даних або виконуються одночасно, або не виконуються взагалі.

Система автоматично гарантує атомарність таких операцій, як запис транзакцій, оновлення статусу доступу до товарів і перевірка стану запасів товарів. Якщо на якомусь етапі транзакції виникає помилка, всі попередні зміни скасовуються, що гарантує цілісність бази даних. Це забезпечує високу надійність і точність системи, особливо в ситуаціях, коли є одночасний доступ до бази даних з декількох джерел і користувачів.

3.6 Захист даних у системі

У рамках цієї системи було реалізовано кілька рівнів захисту даних, які включають аутентифікацію користувачів, шифрування даних та захист від SQL-ін’єкцій. Для забезпечення безпеки доступу до системи і захисту даних було використано сучасні технології та підходи до аутентифікації, шифрування та захисту від зловмисних запитів.

3.6.1 Аутентифікація користувачів через OAuth і JWT

Для забезпечення безпечного доступу до системи та захисту даних були використані новітні технології та підходи для автентифікації, шифрування та захисту від зловмисних запитів Автентифікація користувачів за допомогою OAuth та JWT є важливим кроком для забезпечення захисту даних. Процес реєстрації та входу в систему вимагає авторизації користувача, після чого система генерує токен доступу, що дозволяє користувачеві виконувати операції в межах його повноважень. Для цього було обрано два основних механізми автентифікації: OAuth та JWT (JSON Web Token) OAuth пропонує можливість делегованого доступу, що дозволяє користувачам надавати доступ до своїх даних третім особам без необхідності ділитися своїми обліковими даними. Цей підхід використовується для доступу до піль-

гових продуктів та інтеграції із зовнішніми сервісами, які забезпечують автентифікацію користувачів JWT використовується для автентифікації всередині системи. Після успішної аутентифікації користувач отримує токен JWT, зображено на рисунку .3.17, який надсилається з кожним запитом на сервер, забезпечуючи безпеку та зручність, дивитись рисунок 3.16. Система перевіряє токен на кожному етапі взаємодії, щоб забезпечити захист від несанкціонованого доступу.

```
public async Task<string> Login(string email, string password)
{
    var user = await _authRepos.GetByEmail(email);
    var roles = await _roleRepository.GetRoleByNameAsync(user.RoleId);

    var result = _Hasher.Verify(password, user.PasswordHash);

    var token = _userHelper.GenerateJwtToken(user, roles);
    return token;
}
```

Рисунок 3.16 — Функція логіну з генерацією нового токену входу

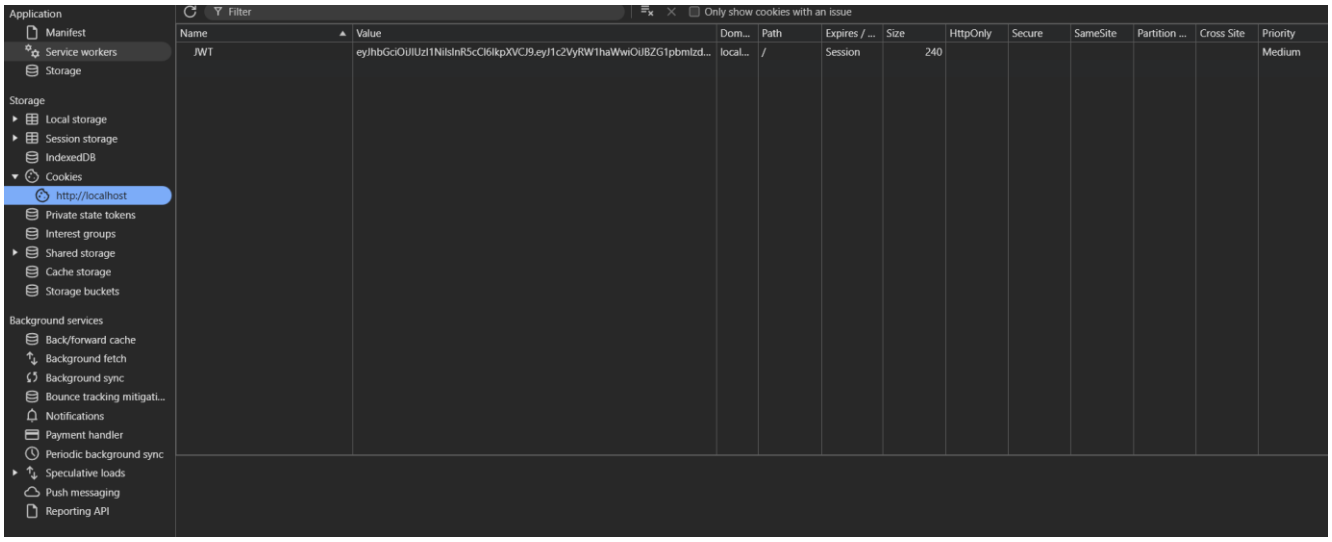


Рисунок 3.17 — Збереження Cookies в користувача

3.6.2 Шифрування даних

Шифрування даних - ще один важливий крок у забезпеченні безпеки. Для захисту передачі даних через мережу в системі використовується протокол TLS

(Transport Layer Security). Це гарантує, що всі дані, які передаються між клієнтською частиною (веб-сайтом або мобільним додатком) і сервером, зашифровані та захищені від перехоплення. Крім того, дані, що зберігаються в базі даних, були зашифровані за допомогою Advanced Encryption Standard (AES), одного з найбезпечніших симетричних алгоритмів шифрування. Всі конфіденційні дані, такі як паролі користувачів, інформація про привілейовані товари та транзакції, шифруються за допомогою AES перед тим, як зберігатися в базі даних. Для зберігання ключів шифрування використовується спеціальний модуль Secure Key Management, що забезпечує додатковий рівень безпеки. Ці заходи гарантують, що навіть якщо база даних буде скомпрометована, доступ до даних буде унеможливлений, а також гарантується висока конфіденційність інформації.

3.7 Масштабування та ефективність роботи системи

У рамках розробленої системи для автоматизації процесу надання пільгової продукції були реалізовані сучасні підходи до масштабування, відмовостійкості та оптимізації продуктивності, що дозволяють системі ефективно працювати при будь-яких умовах.

3.7.1 Хмарні сервіси для масштабування

Одним з основних інструментів забезпечення масштабованості є використання хмарних сервісів. Хмара дозволяє плавно збільшувати системні ресурси відповідно до поточного навантаження. Це важливо для систем, які обробляють великі обсяги даних і численні запити. Для цього було використано Amazon Web Services (AWS), оскільки він пропонує високий рівень масштабованості та доступності. Система побудована таким чином, що зі збільшенням навантаження на сервер автоматично збільшується кількість екземплярів сервісів і ресурсів, таких як сервери

баз даних, веб-сервери та сервери обробки запитів. Зокрема, для забезпечення обробки запитів та масштабованості було впроваджено використання AWS EC2 для віртуальних серверів, AWS RDS для керованих баз даних та AWS S3 для зберігання великих обсягів файлів та резервних копій. Ці сервіси дозволяють швидко масштабувати ресурси за потреби та ефективно управляти навантаженням без необхідності витрачати значні кошти на фізичну інфраструктуру. Найкращі з них представлені 3.18.

Рисунок 3.18 — Частка хварний сервісів в світі

Крім того, використовуються контейнери Docker для забезпечення легкості в деплоїменті та масштабуванні. Це дозволяє ефективно керувати мікросервісами і забезпечує високу мобільність, що важливо при масштабуванні системи. Інтеграція з Kubernetes дозволяє автоматично керувати контейнерами і забезпечувати їх розподіл по кількох серверах для кращого розподілу навантаження та ефективного використання ресурсів.

3.7.2 Відмовостійкість і високодоступні архітектури

Механізми відмовостійкості та високої доступності впроваджуються, щоб гарантувати безперервну роботу системи у випадку непередбачуваних подій або апаратних збоїв. У цьому контексті системи будуються за принципом кластеризації, щоб гарантувати безперервну роботу в разі виходу з ладу одного з компонентів або серверів. Одним з основних підходів є автоматичне балансування навантаження за допомогою AWS Elastic Load Balancer (ELB), який рівномірно розподіляє трафік між декількома серверами. Якщо один сервер стає недоступним, ELB автоматично перенаправляє запити на інші доступні сервери, забезпечуючи високу доступність системи. Ще одним важливим елементом є реплікація бази даних у режимі реального часу. Для цього ми використали механізм Multi-AZ для баз даних в AWS, який дозволяє створювати копії даних у різних дата-центрах. У разі виходу з ладу основного сервера система автоматично перемикається на резервну копію бази даних, що гарантує безперервність бізнесу.

Для забезпечення високої доступності також було застосовано використання кешування через Redis, що дозволяє значно знизити навантаження на сервери баз даних та підвищити швидкість обробки запитів до системи. Всі найбільш часто запитувані дані зберігаються в кеші, що дозволяє швидко отримувати необхідну інформацію без повторних запитів до бази даних.

3.7.3 Оптимізація продуктивності

Оптимізація продуктивності є важливим кроком у забезпеченні ефективної роботи системи. Для цього використовуються різні методи та інструменти, що дозволяють зменшити час відгуку та збільшити швидкість обробки запитів. Одним з основних методів є індексування бази даних. Всі основні таблиці, що містять великі обсяги даних, були проіндексовані, що значно скоротило час, необхідний для пошуку записів. Крім того, для зменшення навантаження на сервери бази даних були використані оптимізовані запити. Це значно підвищило швидкість обробки великих обсягів даних і зменшило час відгуку системи.

Для зменшення часу відгуку на стороні клієнта було оптимізовано фронтенд: за допомогою React було досягнуто швидкого рендерингу інтерфейсу з віртуальним DOM, що значно зменшило навантаження на браузер користувача, місце входу зображено на рисунку 3.19. Крім того, час завантаження сторінок було зменшено завдяки використанню лінивого завантаження для завантаження компонентів тільки тоді, коли це необхідно.

```
ReactDOM.createRoot(document.getElementById("root")!).render(
  <Context.Provider
    value={{
      userStore, accesedStore, typeListStore
    }}
  >
    <App />
  </Context.Provider>
);
```

Рисунок 3.19 — Місце входу в клієнтську частину додатку

Велику увагу також було приділено моніторингу продуктивності в режимі реального часу. Для цього були використані такі інструменти, як New Relic і Prometheus. Ці інструменти дозволяють в режимі реального часу відстежувати стан

сервісів, баз даних та веб-серверів для виявлення проблем з продуктивністю та швидкого реагування. В результаті використання цих технологій система забезпечує стабільну роботу під високими навантаженнями та гарантує високу доступність і оперативність в обробці запитів, що є ключовими аспектами для успішного функціонування автоматизованої системи доставки привілейованих продуктів.

3.8 Тестування та забезпечення якості

Тестування та забезпечення якості є важливими кроками на всіх етапах розробки системи для забезпечення надійності, стабільності та безпеки системи. В рамках системи, розробленої для автоматизації доставки привілейованих продуктів, були реалізовані комплексні заходи з тестування якості коду, перевірки працездатності та забезпечення безпеки. Це включає в себе як ручне, так і автоматизоване тестування, а також впровадження системи моніторингу, яка дозволяє відстежувати стан додатку в режимі реального часу і виявляти потенційні проблеми.

3.8.1 Моніторинг продуктивності в реальному часі

Один із основних аспектів забезпечення якості системи — це постійний моніторинг її продуктивності під час роботи. Всі критичні компоненти системи, зокрема сервери, бази даних і зовнішні API, постійно відстежуються для забезпечення своєчасного виявлення та усунення можливих проблем. Для цього було впроваджено спеціальні інструменти для моніторингу, такі як Prometheus і Grafana, які дозволяють в реальному часі збирати метрики, моніторити використання ресурсів, а також проводити аналіз ефективності системи.

Prometheus використовується для збору та зберігання метрик, таких як час відповіді серверів, навантаження на процесори, використання пам'яті, швидкість

з'єднання з базами даних і API. Ці дані передаються в Grafana, де створюються візуалізації та дашборди для виявлення аномалій та тенденцій, приклад зображено на рисунку 3.20. У разі перевищення допустимих порогових значень, система автоматично надсилає сповіщення, що дає можливість оперативно реагувати на будь-які зміни в роботі сервісів.

Цей підхід дозволяє не лише виявляти можливі проблеми в реальному часі, але й проводити детальну аналітику після їх усунення для запобігання аналогічних ситуацій у майбутньому. Такий моніторинг є важливою частиною стратегії забезпечення якості, оскільки він дає змогу оперативно виправляти будь-які дефекти продуктивності, до того як вони почнуть впливати на користувачів.

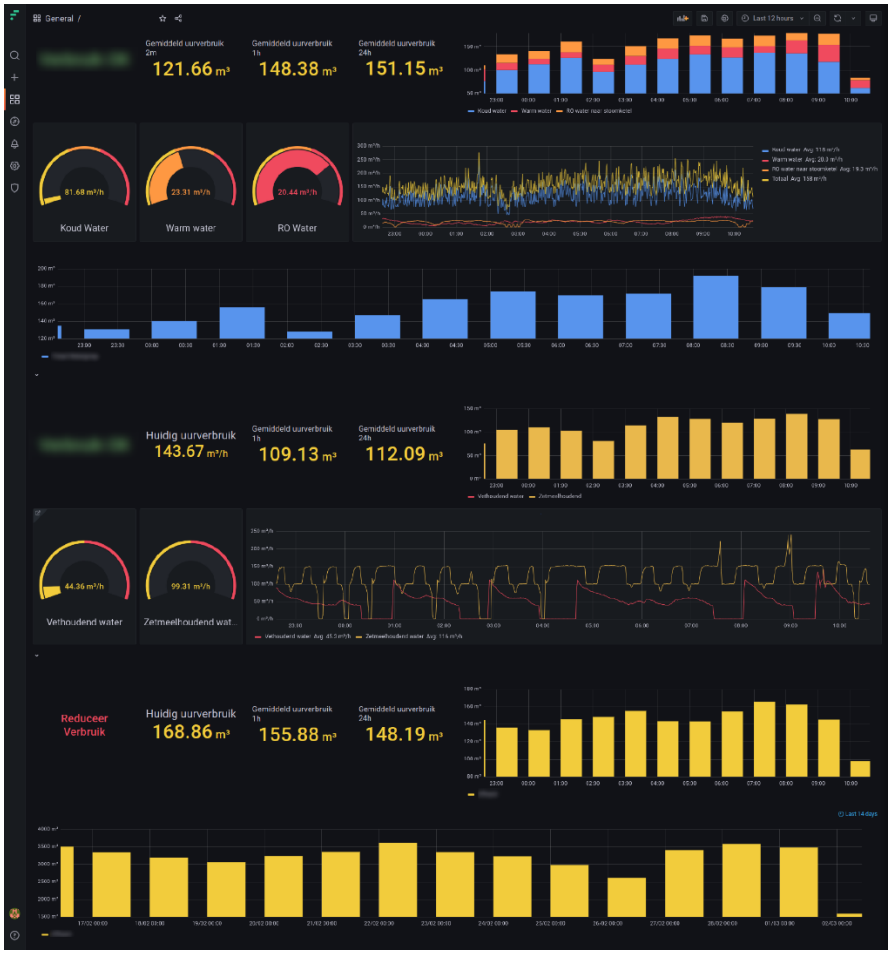


Рисунок 3.20 — Інтерфейс моніторингу Grafana

3.8.2 Тестування безпеки

Для забезпечення безпеки даних та захисту від зовнішніх атак було проведено комплексне тестування безпеки системи. Воно включало як статичне, так і динамічне тестування коду, а також використання різних інструментів для перевірки вразливостей. Тестування безпеки дозволяє виявити потенційні загрози на етапі розробки та забезпечити захист від можливих атак під час експлуатації. Одним з найважливіших етапів тестування є перевірка коду на наявність таких вразливостей, як SQL-ін'єкції, XSS-атаки (міжсайтовий скриптинг), CSRF (підробка міжсайтових запитів) та інших поширених вразливостей веб-додатків. Для цього були використані такі інструменти, як OWASP ZAP (Zed Attack Proxy). Цей інструмент автоматично сканує веб-додатки на наявність відомих вразливостей; для тестування безпеки API використовувався Postman в поєднанні з автоматизованими тестами для перевірки надійності механізмів автентифікації, таких як OAuth і JWT, а політики доступу та автентифікації політики доступу та автентифікації були перевірені на коректність реалізації.

Крім того, було протестовано шифрування даних на різних етапах обробки інформації. Це включало тестування таких алгоритмів шифрування, як AES для захисту даних у базі даних та захист під час передачі через TLS/SSL для забезпечення безпеки каналу зв'язку.

Такі тестування безпеки є основою для запобігання витоку даних та захисту системи від потенційних загроз. Вони допомагають виявити слабкі місця в архітектурі безпеки та забезпечити належний рівень захисту на всіх етапах роботи системи. Тут можна вставити скріншоти, що показують результати тестів безпеки, наприклад, виявлені вразливості або звіти про успішне тестування за допомогою інструменту OWASP ZAP. Всі ці заходи тестування забезпечують високу якість системи та захист даних користувачів і є важливими для забезпечення надійності та безпеки автоматизованих систем надання пільг.

3.9 Оновлення та підтримка системи

Оновлення та підтримка систем є важливим аспектом забезпечення їх стабільної роботи та адаптації до нових вимог і змін у навколишньому середовищі. Це включає в себе як регулярне оновлення функціоналу, так і виправлення виявлених помилок та вразливостей. Оскільки розробка та підтримка такого типу систем передбачає постійну взаємодію з користувачами та зміни в інфраструктурі, важливо, щоб існували чітко визначені механізми оновлення системи, управління резервними копіями та безперервного моніторингу її працездатності.

3.9.1 Механізми оновлення та релізу нових версій

Механізм оновлення є важливою частиною забезпечення стабільності та безперервності роботи системи. У нашій системі реалізовано підхід до автоматизації процесу розгортання нових версій через CI/CD (Continuous Integration/Continuous Deployment) pipeline. Це дозволяє зменшити час між розробкою нових функціональних можливостей і їх випуском в продуктивне середовище.

Використовуючи Jenkins або GitLab CI, процеси тестування та зборки коду автоматизовано таким чином, що нові версії коду проходять автоматичні тести і лише після їх успішного виконання потрапляють на продакшн. Це забезпечує безпеку і якість оновлень, оскільки система автоматично перевіряє, чи не вносяться помилки в нові версії.

Додатково для релізу використовуються Docker-контейнери для упаковки додатків, що дозволяє ізолювати середовище розгортання та забезпечити легкість перенесення між різними платформами. Після збору та тестування, нові версії публікуються на хмарному середовищі (наприклад, AWS, Azure), що дозволяє масштабувати систему без значних складнощів.

Таким чином, процес оновлення відбувається плавно і безперервно, що дозволяє оперативно впроваджувати нові функціональні можливості або виправлення помилок без значного впливу на працездатність системи. Реалізація представлена на рисунку 3.21.

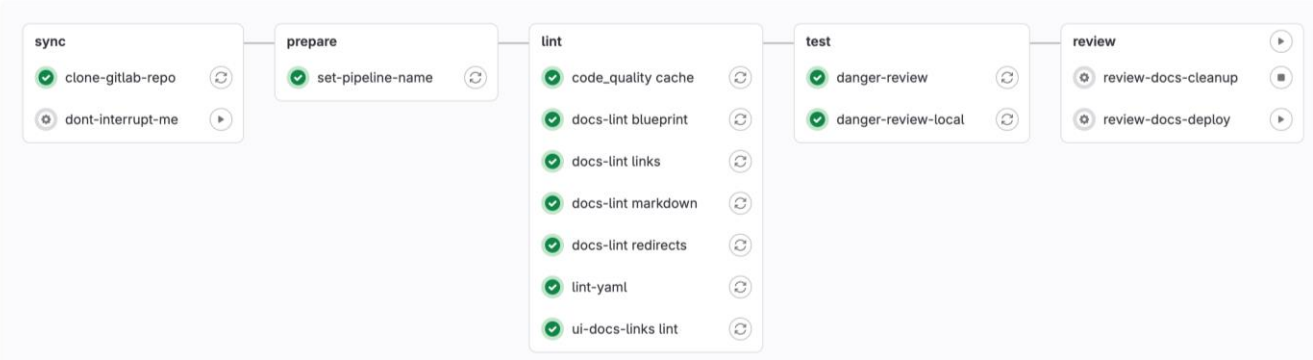


Рисунок 3.21 — Налаштування CI/CD pipeline

3.9.2 Стратегії резервного копіювання та відновлення даних

Важливою частиною підтримки будь-якої системи є забезпечення безпеки даних, що зберігаються, та можливість їх відновлення в разі аварійних ситуацій. Для цього були реалізовані стратегії резервного копіювання баз даних і важливих файлів конфігурацій, а також автоматизовані механізми для їх відновлення.

Система забезпечує щоденне резервне копіювання всіх критичних даних, що зберігаються в базах даних, за допомогою інструментів, таких як MySQL Dump для баз даних MySQL або pg_dump для PostgreSQL. Резервні копії зберігаються на окремому сервері або в хмарному сховищі, наприклад, в Amazon S3 або Google Cloud Storage, що гарантує їхню безпеку та доступність.

Процес відновлення даних був автоматизований за допомогою сценаріїв відновлення, що можуть бути запущені в разі потреби, з можливістю відновити дані до будь-якої точки часу, що була зафіксована в резервній копії. Це дозволяє мінімізувати час простою системи та забезпечити безперервність бізнес-процесів.

Для захисту від випадкових втрат даних реалізовано механізм двухфакторної аутентифікації при доступі до систем резервного копіювання та відновлення даних, що додає додатковий рівень безпеки при роботі з чутливою інформацією.

3.9.3 Моніторинг і підтримка в процесі експлуатації

Підтримка і моніторинг системи в процесі її експлуатації включає постійне спостереження за її працездатністю, виявлення та усунення проблем, а також реагування на зміни в навантаженні або вимогах користувачів. Це дозволяє забезпечити стабільну роботу системи протягом тривалого часу без значних перерв.

Для цього був налаштований постійний моніторинг усіх основних компонентів системи, таких як сервери, бази даних, сервери API, а також інтеграційні сервіси. Використання таких інструментів, як Prometheus для збору метрик та Grafana для візуалізації стану сервісів, дає можливість оперативно реагувати на будь-які відхилення від норми, наприклад, високе навантаження на сервери або затримки в роботі API.

До того ж, реалізовано систему сповіщень для оперативного реагування на критичні ситуації, що дозволяє вчасно вжити заходів для їх усунення. Вся команда підтримки має доступ до дашбордів, які дозволяють стежити за критичними параметрами системи, і одразу отримувати повідомлення в разі виникнення будь-якої неполадки.

Моніторинг також включає спостереження за ефективністю роботи резервних копій та забезпечення їх актуальності. Тестування резервного копіювання виконується автоматично через визначені проміжки часу, що дає впевненість у наявності працюючих копій даних у разі необхідності їх відновлення. Вигляд моніторингу представлений на рис. 3.22.

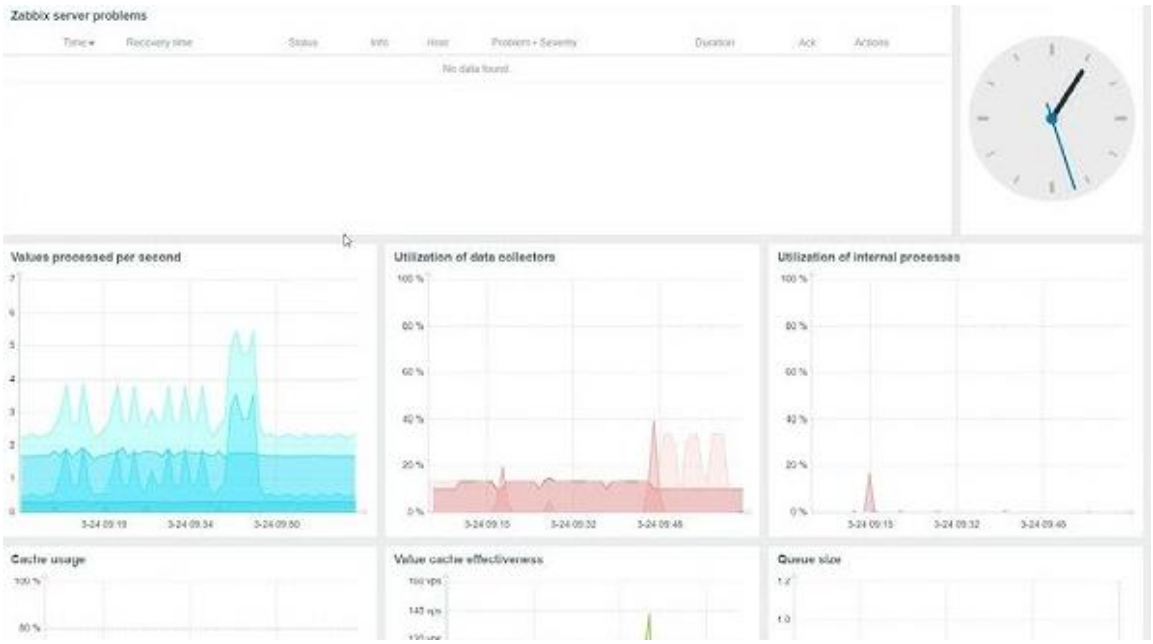


Рисунок 3.22 — Моніторинг за ресурсами

Ці стратегії і механізми забезпечують безперебійну роботу системи, мінімізацію часу на її відновлення та високу надійність, що критично важливо для успішної роботи автоматизованої системи надання пільгової продукції.

ВИСНОВКИ

В результаті дослідження було розроблено концепцію веб-системи, яка використовує відкриті API та інтеграцію з базами даних для автоматизації процесу надання пільгових товарів. Запропоноване рішення значно підвищує ефективність, точність та прозорість процесу управління доступом до товарів, мінімізуючи при цьому можливість помилок, пов'язаних з ручною обробкою даних. Використання відкритих API всередині системи забезпечує уніфікований та стандартизований доступ до зовнішніх сервісів та скорочує час, необхідний для обміну даними між різними компонентами.

Інтеграція з базами даних дозволяє автоматизувати процеси обробки та зберігання інформації, а також гарантує високий рівень безпеки та контролю доступу до конфіденційної інформації. Розроблена система складається з двох основних модулів: один модуль перевіряє дані користувача через API та видає привілейовані продукти, а інший модуль забезпечує функціональність для управління та управління правами доступу.

Така архітектура системи дозволяє ефективно керувати процесами та підтримувати високий рівень безпеки даних. Одним з найважливіших аспектів розробки є використання сучасних технологій, таких як Node.js, React, MySQL та C# ASP.NET, що забезпечує стабільну роботу системи, масштабованість та високу продуктивність; інтеграція з зовнішніми API, такими як Uprox та 1C, дозволяє перевіряти дані в режимі реального часу; а використання різноманітних зовнішніх API, в тому числі C# ASP.NET API, дозволяє використовувати систему в різних напрямках, інформація є більш точною та актуальною.

Розроблена система відповідає найсучаснішим вимогам до автоматизації процесів та інтеграції технологій і гарантує швидкість, ефективність та безпеку. Вона значно спростить управління доступом до привілейованих продуктів та зменшить ризики, пов'язані з людським фактором і помилками в процесі. Крім того, система має потенціал для подальшого розвитку та адаптації до різних сфер діяльності, що потребують подібних рішень.

В кінці-кінців, запропонована веб-система є ефективним інструментом для автоматизації та управління процесами надання пріоритетних послуг, що відповідає всім вимогам щодо швидкості, безпеки та точності даних. Її впровадження в організації значно покращить якість надання послуг, зменшить адміністративне навантаження та підвищить прозорість обробки даних і контролю доступу до них.

ПЕРЕЛІК ПОСИЛАНЬ

1. Платформа для розробки Node.js. Офіційна документація Node.js. [https://nodejs.org/en/docs/]
2. ReactJS. Офіційна документація React. [https://reactjs.org/docs/getting-started.html]
3. MySQL. Офіційна документація MySQL: Кращі практики для продуктивності. [https://dev.mysql.com/doc/refman/8.0/en/optimization.html]
4. C# ASP.NET. Офіційна документація для розробників ASP.NET. [https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0]
5. OAuth 2.0. Рекомендації щодо використання OAuth 2.0 для аутентифікації та авторизації. [https://tools.ietf.org/html/rfc6749]
6. JWT.io. Документація по JWT (JSON Web Token). [https://jwt.io/introduction/]
7. OWASP. OWASP API Security Top 10. [https://owasp.org/www-project-api-security/]
8. OWASP. SQL Injection Prevention Cheat Sheet. [https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html]
9. Cloud Computing. Моделі та стратегії масштабування в хмарних сервісах. [https://azure.microsoft.com/en-us/overview/scalable-applications/]
10. High Availability Architectures. Високодоступні архітектури для забезпечення безперервної роботи. [https://www.ibm.com/cloud/architecture/]
11. AWS. Високодоступні архітектури та стратегії відмовостійкості в хмарі. [https://aws.amazon.com/architecture/well-architected/]
12. Google Developers. Основи оптимізації продуктивності веб-застосунків. [https://developers.google.com/web/fundamentals/performance]
13. Percona. Стратегії індексації для оптимізації роботи з базами даних. [https://www.percona.com/blog/2021/08/11/database-indexing-strategies-for-performance-optimization/]

14. Microsoft Docs. Резервне копіювання та відновлення даних у базах даних SQL Server. [https://docs.microsoft.com/en-us/sql/ssms/backup-and-restore/backup-and-restore-of-sql-server-databases]
15. Redgate. Кращі практики для резервного копіювання MySQL. [https://www.red-gate.com/simple-talk/sql/backup-and-restore/mysql-backup-and-restore-best-practices/]
16. ElasticSearch. Офіційна документація по налаштуванню масштабування в Elasticsearch. [https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html]
17. MongoDB. Документація по масштабуванню та реплікації в MongoDB. [https://www.mongodb.com/docs/manual/sharding/]
18. Docker. Використання контейнерів для масштабування веб-застосунків. [https://www.docker.com/resources/what-container]
19. Kubernetes. Документація Kubernetes по автоматичному масштабуванню. [https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/]
20. Google Cloud. Інструменти для моніторингу та управління продуктивністю в реальному часі. [https://cloud.google.com/products/operations]
21. Prometheus. Огляд інструментів для моніторингу в реальному часі. [https://prometheus.io/docs/introduction/overview/]
22. Grafana. Використання Grafana для збору та візуалізації даних про продуктивність. [https://grafana.com/docs/grafana/latest/getting-started/]
23. Azure DevOps. Моніторинг та підтримка в процесі експлуатації в Azure. [https://azure.microsoft.com/en-us/services/devops/]
24. CI/CD. Підхід до автоматизації розгортання за допомогою CI/CD. [https://www.jenkins.io/doc/book/pipeline/]

ДОДАТОК А

Місце входу в програму

Main.tsx

```
import { createContext } from "react";
import ReactDOM from "react-dom/client";
import App from "./App.tsx";
import UserStore from "./store/UserStore.ts";
import AccesedStore from "./store/AccesedStore.ts";
import TypeListStore from "./store/TypeListStore.ts";

interface State {
  userStore: UserStore;
  accesedStore: AccesedStore;
  typeListStore: TypeListStore;
}

export const userStore = new UserStore();
export const accesedStore = new AccesedStore();
export const typeListStore = new TypeListStore();

export const Context = createContext<State>({
  userStore, accesedStore, typeListStore
});

ReactDOM.createRoot(document.getElementById("root")!).render(
  <Context.Provider
    value={{
      userStore, accesedStore, typeListStore
    }}
  >
    <App />
  </Context.Provider>
);
```

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Веб-система для автоматизації процесів надання пільгової продукції на основі відкритих API та баз даних

@BakalBogdan

Виконав студент групи КНДМ61
Бакал Богдан Олександрович

Вступ

Тема

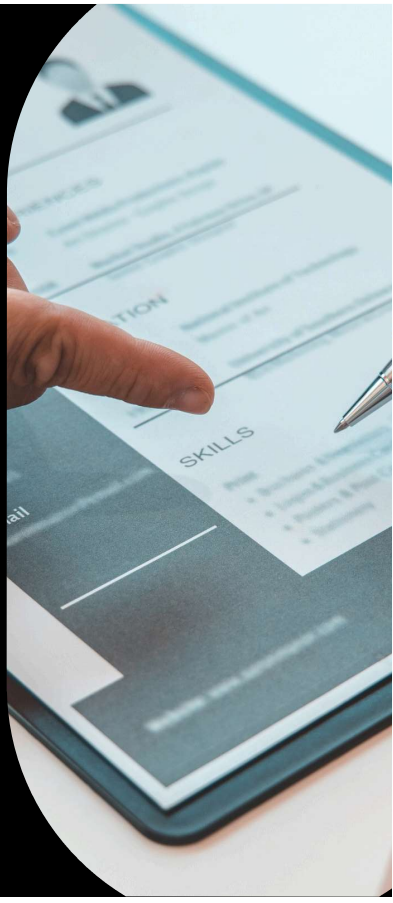
Сучасні підходи до автоматизації бізнес-процесів за допомогою відкритих API та інтеграції з базами даних.

Актуальність

Необхідність в автоматизації процесів у фінансовому, медичному та адміністративному секторах.

Ціль дослідження

Розробка веб-системи для автоматизації надання пільгової продукції



Проблематика

Олександр Коваль

У сучасних умовах швидкість та точність обробки даних стають критично важливими в різних секторах економіки (фінанси, медицина, адміністрація).

Марія Соколова

Необхідність у точному та своєчасному наданні пільгової продукції, що вимагає автоматизації процесу управління правами доступу та обліку видачі.

Дмитро Іванов

Проблема зберігання та обробки даних у різних системах, що може призвести до помилок або затримок у видачі продукції.

Олена Черненко

Враховуючи, що система працює з чутливою інформацією (особисті дані користувачів, інформація про пільги та доступ до них), важливо забезпечити високий рівень безпеки та захисту даних

Мета та завдання дослідження

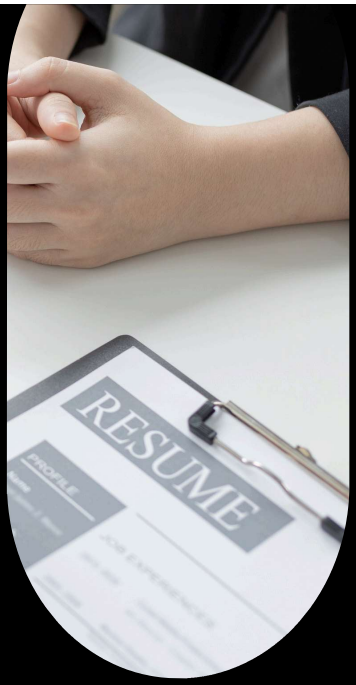
Мета дослідження

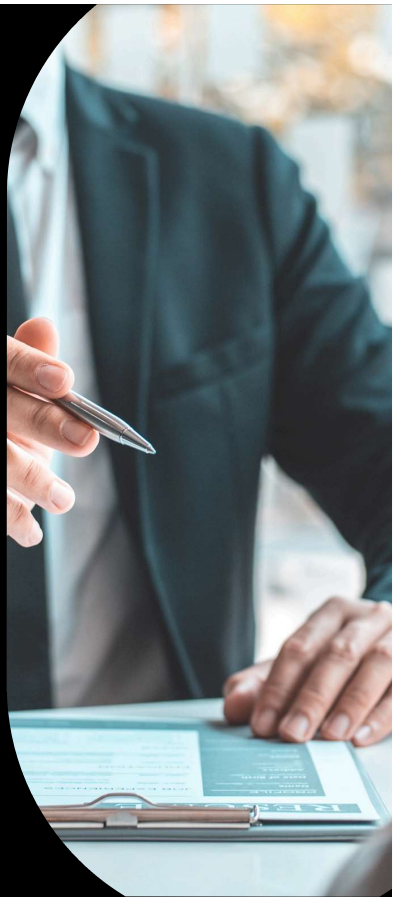
Метою цього дослідження є розробка веб-системи для автоматизації процесу надання пільгової продукції. Система має бути інтегрована з зовнішніми сервісами та базами даних через відкриті API, щоб забезпечити ефективний контроль над процесом доставки продукції. Основною метою є створення ефективного та прозорого механізму для перевірки прав користувачів та автоматизації обміну даними, що знижує ймовірність помилок і покращує точність обліку.

Завдання дослідження

- Розробка двох сайтів: створення двох основних сайтів для контролю фактичної доставки пільгових продуктів. Один з них служитиме для перевірки доступу користувачів, а інший — для адміністрування даних.
- Інтеграція з зовнішніми сервісами через API: забезпечення з'єднання з іншими зовнішніми платформами, які надають дані про користувачів, а також для верифікації пільгових осіб у реальному часі.
- Управління доступом та правами користувачів: створення механізму для автоматичної перевірки прав доступу та управління правами користувачів, щоб забезпечити точність та своєчасність надання пільгової продукції.

Архітектура веб-системи





Опис загальної архітектури

Веб-система складається з двох основних модулів: модуль перевірки даних через API та модуль роботи з базою даних. Вони взаємодіють через захищене API для забезпечення безпеки та швидкості обміну даними.

Модулі системи

- Модуль перевірки даних: взаємодіє з API для перевірки прав доступу користувачів.
- Модуль роботи з базою даних: зберігає дані та керує доступом до пільгової продукції

Вибір архітектурного стилю

Використовуваний архітектурний стиль — клієнт-сервер з мікросервісами для забезпечення масштабованості, надійності та гнучкості.

Вибір технологій та інструментів

Основні технології

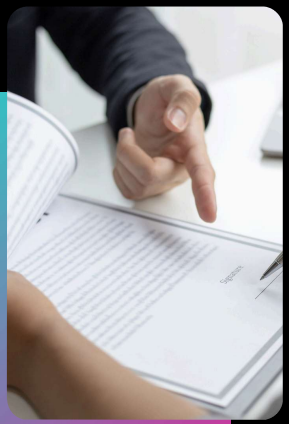
- Backend: ASP.NET
- Frontend: React
- База даних: MySQL

Інструменти для інтеграції з API

- RESTful API: для взаємодії між веб-системою та зовнішніми сервісами.
- GraphQL: для більш ефективного запиту даних, якщо система потребує гнучкості.
- OAuth, JWT: для забезпечення безпечної аутентифікації та авторизації.

Інструменти для безпеки

- TLS/SSL: для шифрування передачі даних.
- Firewall, Anti-DDoS: для захисту серверів від атак.
- OWASP ZAP: для тестування безпеки системи.



Безпека даних і контроль доступу

Аутентифікація та авторизація

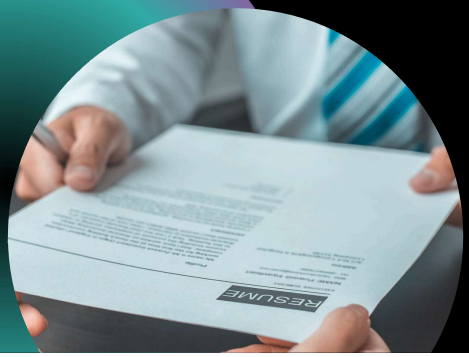
- OAuth 2.0
- JWT

Регулювання доступу

- рівні доступу
- Рольова модель доступу (RBAC)

Шифрування даних

- TLS/SSL.
- AES



Масштабування та ефективність роботи системи

Відмовостійкість та високодоступні архітектури

- Використовуються балансувальники навантаження для рівномірного розподілу трафіку.
- Резервні сервери забезпечують високу доступність та безперервну роботу системи навіть у разі відмови одного з серверів.

Оптимізація продуктивності

- Використовуються кешування для зберігання часто запитуваних даних.
- Використовуються асинхронні запити для покращення швидкості відповіді та зменшення навантаження на сервери.

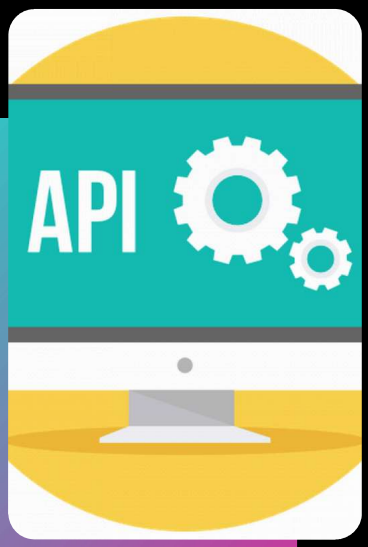
Автоматизоване тестування API

Автоматизоване тестування API:

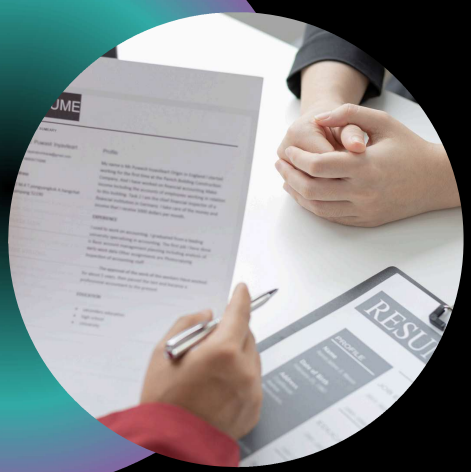
Postman
Swagger
JUnit

Моніторинг продуктивності в реальному часі

Grafana
Zabbix



Оновлення та підтримка системи



Механізми оновлення та релізу нових версій

CI/CD

Стратегії резервного копіювання та відновлення даних

автоматизовані бекапи