Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для A/B тестування товарів компанії
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-2
Юрченко В. Ю.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія

(повна назва освітньої програми)
Керівник ст.викл. Олійник О.В.
(посада, прізвище, ініціали)

Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 –Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Юрченко Валентину Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Програмна система для A/B тестування товарів компанії» затверджена наказом університету від 20 травня 2024 р. № 471Ст

2. Термін здачі студентом закінченої роботи 10 червня 2024 р.

3. Вихідні дані до роботи *В програмній реалізації мають бути передбачені всі необхідні аспекти готового продукту. Створення смарт-девайсу для збору інформації про відвідувачів магазинів, з використанням платформи Arduino, створення серверної частини програмного забезпечення для обробки отриманих з тестувань даних, побудова моделей та проекцій на їх основі, авторизації користувачів та реєстрації компаній, з використанням платформи Node.js, нереляційної бази даних MongoDB та мікросервісної архітектури. Створення клієнтської частини із сучасним зручним дизайном, виведенням графіків, ілюстрацій та сучасним підходом до серверного рендерингу.*

4. Перелік питань, що потрібно опрацювати в роботі: *вступ, аналіз предметної області, перелік вимог до програмної системи, архітектура та проектування програмної системи, опис прийнятих програмних рішень, впровадження програмного забезпечення, висновки, перелік джерел посилань, додатки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі, огляд існуючих рішень, розробка постановки задачі	12.04.2024	Виконано
2	Формування вимог до ПЗ	23.04.204	Виконано
3	UML-проектування, розробка архітектури ПЗ, розробка UI/UX	01.05.2024	Виконано
4	Проектування ключових алгоритмів ПЗ	07.05.2024	Виконано
5	Розробка та тестування ПЗ	17.05.2024	Виконано
6	Практичне впровадження ПЗ	25.05.2024	Виконано
7	Написання пояснювальної записки	28.05.2024	Виконано
8	Перевірка пояснювальної записки керівником, підготовка до перевірки на антиплагіат	30.05.2024	Виконано
9	Оцінка роботи рецензентом, отримання відзиву від керівника кваліфікаційної роботи, попередній захист роботи та проходження нормо контролю	07.06.2024	Виконано
10	Захист кваліфікаційної роботи	10.06.2024	Виконано

Дата видачі завдання 3 квітня 2024 р.

Студент _____ Юрченко В.Ю.

(підпис)

Керівник роботи _____ ст. викл. Олійник О.В.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 83 с., 13 рис., 10 джерел.

А/В ТЕСТУВАННЯ, АВТОРИЗАЦІЯ, БАЗА ДАНИХ, ВЕБ-ЗАСТОСУНОК, МІКРОСЕРВІСИ, СЕРВЕРНИЙ РЕНДЕРИНГ, ТОВАРИ, ШЛЮЗ, ARDUINO, NEXT.JS, NODE.JS, NOSQL, REST API, TYPESCRIPT.

Об’єкт розробки – програмна система для А/В тестування товарів компанії. Об’єктом дослідження є предметна область продажів у фізичних магазинів та рівень зацікавленості споживачів до окремих товарів на полицях. Вона охоплює як вплив зовнішнього вигляду товарів на їх «популярність» серед покупців, так і вплив вибору місця на продажі продукції.

Мета роботи – покращити процеси аналізу продажів та інтересу споживачів у фізичних товарах в магазинах, використовуючи в цій сфері такі відомі практики з інтернет-торвгівлі як А/В тестування чи дисперсний аналіз.

Методи рішення – платформа для розробки електронних пристроїв на базі мікроконтролерів та інтерфейсних плат Arduino, мова програмування зі статичною типізацією TypeScript, платформа для серверної розробки Node.js, NoSQL база даних MongoDB та прогресивний front-end фреймворк Next.js.

Результатом роботи є програмна система, яка дозволить менеджерам та користувачам бізнесів ще точніше визначати, які товари цікаві ринку, а також прогнозувати активність покупців по відношенню до певних товарів у майбутньому.

A/B TESTING, AUTHORIZATION, DATABASE, WEB APPLICATION, MICROSERVICES, SERVER RENDERING, GOODS, GATEWAY, AEDUINO, NEXT.JS, NODE.JS, NOSQL, REST API, TYPESCRIPT.

The object of development is a software system for A/B testing of company products. The object of the study is the subject area of sales in physical stores and the level of consumer interest in individual products on the shelves. It covers both the impact of the appearance of goods on their "popularity" among buyers, and the impact of the choice of location on product sales.

The purpose of the work is to improve the processes of analyzing sales and consumer interest in physical goods in stores, using in this field such well-known practices from the Internet trade as A/B testing or dispersion analysis.

The solution methods are a platform for the development of electronic devices based on microcontrollers and Arduino interface boards, a programming language with static typing TypeScript, a platform for server development Node.js, a NoSQL database MongoDB and a progressive front-end framework Next.js.

The result of the work is a software system that will allow business managers and users to more accurately determine which products are of interest to the market, as well as predict the activity of buyers in relation to certain products in the future.

Я, Юрченко Валентин Юрійович, студент гр. ПЗПІ-20-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для A/B тестування товарів компанії», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

HTTP – Hypertext Transfer Protocol

SEO – Search Engine Optimization

SSR – Server Side Rendering

JSON – JavaScript Object Notation

UML – Unified Modeling Language

SDK – Serial Development Kit

TCP – Transmission Control Protocol

SQL – Structured Query Language

ODM – Object Data Modelling

SSG – Static Site Generation

ISR – Incremental Static Regeneration

ВСТУП

Мета моєї кваліфікаційної роботи полягає у розробці програмної системи, яка допоможе компаніям проводити A/B тестування своїх продуктів в реальному житті. Вона пропонує власникам придбати розумний пристрій та використовувати його для аналізу зацікавленості покупців. Звичайні ж користувачі, які не мають своїх компаній, зможуть покращувати якість продукції, оцінюючи товари.

Сучасне підприємництво надзвичайно конкурентоспроможне, і невдалі рішення щодо вибору продукції можуть мати серйозні наслідки. На даний момент майже не існує таких же ефективних фізичних способів аналізу та покращення ефективності продажів, які доступні в онлайн-просторі, бо такі інструменти як A/B тестування не охоплюють матеріальні товари. Моя система спрямована на вирішення цієї проблеми, використовуючи розумні пристрої для збору даних, що допоможе компаніям приймати обґрунтовані рішення.

Суто кількість покупок того чи іншого товару не є повноцінним показником привабливості товару, адже чималу роль у підсумковій покупці завжди грає ціна. Моя ж система завдяки розумному датчику визначення руху зможе виключати цей фактор і отримувати більш справедливу картину якості маркетингових ідей, що приймаються.

Оскільки така система матиме актуальність у будь-якому куточку світу, вона вимагатиме підтримку інтернаціоналізації отже це ще один аспект, якому приділено велику увагу під час розробки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Основним контингентом користувачів, для яких створюється система, є власники та адміністратори офіційних копаній, продукти яких поширюються у звичайних фізичних магазинах та торгівельних центрах. Для цього необхідно володіти належним технічним оснащенням, а саме розумним пристроєм, що зможе фіксувати покупців. Саме тому одним з аналогів моєї системи можна вважати платформу Ignite Prism, що базується в Каліфорнії. Цінність їхніх послуг полягає у аналізі ефективності конфігурації приміщень магазинів за допомогою 360° камери, встановленої на стелі. Таким чином пристрій здатний реєструвати весь трафік у магазинах та відповідати на такі питання: «Як довго клієнти затримуються в магазинах», «Як часто відвідувачі потрапляють у певний ряд» та «Які ділянки магазинів є найактивнішими, а які відвідувачі ігнорують?» Для аналізу даних, отриманих із камер використовуються методи A/B-тестування, схожі на ті, які я також використовую у своїй системі, а також Ignite Prism має перевагу з точки зору візуалізації, адже ця система здатна складати теплові карти магазинів (див. рис. 1.1).

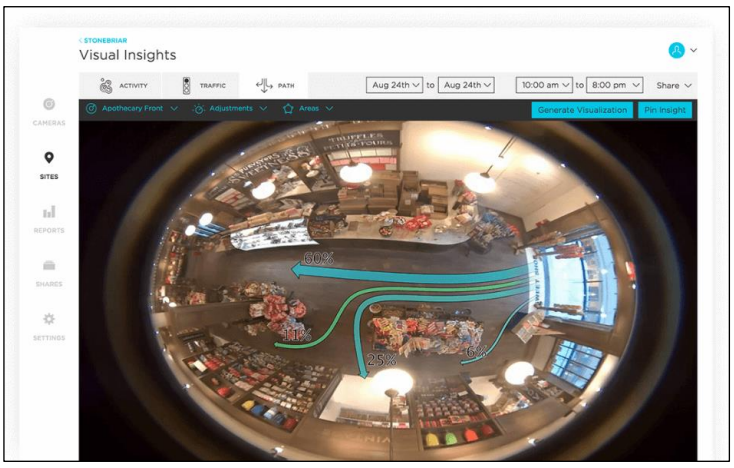


Рисунок 1.1 – Результат побудови карти приміщення в Ignite Prism

Однак перевагою мого продукту є те, що мій девайс дозволяє проводити аналіз конкретного продукту, ігноруючи товари на полицях, які на даний момент не входять в інтереси клієнта. Це дає можливість приймати конкретні рішення

щодо ефективності товарів з маркетингової точки зору та бачити загальну картину більш точно. Також моя система ніяк не залежить від масштабів магазинів, в яких проходять виміри, адже один девайс може бути прикріплений лише до одного товару. Проблема Ignite Prism полягає в тому, що камера не завжди здатна захоплювати все приміщення для аналізу.

Також існує чимало аналогів моєї системи, які виконують схожі функції для онлайн-магазинів.

Сервіс Dynamic Yield надає послуги A/B тестування для онлайн-платформ. Його великою перевагою є використання штучного інтелекту для обробки результатів, що робить систему дійсно потужною. Dynamic Yield надає велику кількість інформації про профілі клієнтів. Його переваги наступні:

- потужність систем;
- збір інформації про профілі клієнтів;
- зручний та зрозумілий веб-інтерфейс;
- можливість організації просунутих онлайн A/B тестувань [6];
- оптимізовані рішення відповідно до отриманої інформації.

Не дивлячись на те, що платформа блискуче вирішує систему A/B тестування для сайтів, саме зазначені в меті задачі він не виконує. До недоліків можна віднести:

- неможливість організації офлайн A/B тестувань;
- відсутність рейтингу товарів компаній;
- вузький діапазон магазинів, з якими співпрацюють;
- велика вартість послуг;
- відсутність ANOVA-аналізу.

Ще більш відомим ресурсом є Google Analytics. Це безкоштовний веб-аналітичний сервіс, що надається Google, який призначений для збирання та аналізу даних про відвідувачів сторінок. Перевагами цієї системи є:

- велика спільнота та ресурси в базах даних;
- доступність даних у режимі реального часу;

– зручний та зрозумілий веб-інтерфейс;
– API-інтеграція з іншими інструментами;
– велика кількість безкоштовних послуг.

Незважаючи на очевидні переваги, Google Analytics не вирішує саме моє завдання через такі недоліки:

– неможливість організації офлайн A/B тестувань;
– обмеження в обробці даних;
– проблеми з конфіденційністю даних;
– складність аналізу та налаштування;
– обмежені можливості для аналізу багатоканальних кампаній.

Платформою для мобільного маркетингу та взаємодії з користувачами, призначеною для тестування продукції, є Leanplum. Переваги такі:

– аналітика та сегментація;
– можливість організації онлайн A/B тестувань;
– пуші та повідомлення на телефон;
– зручний та зрозумілий інтерфейс;
– багатомовність та глобальність.

Недоліки цього мобільного додатка наступні:

– цінова політика для малих та середніх підприємств;
– залежність від сторонніх платформ;
– обмежені аналітичні можливості;
– неможливість обробки великих обсягів даних;
– проблеми з конфіденційністю даних.

1.2 Виявлення та вирішення проблем

Аналізуючи аналоги, ми визначили, що жоден з потенційних аналогів моєї системи і близько не вирішує поставлену проблему. Навіть ті компанії, які спеціалізуються на аналізах фізичних магазинів та закладів, концентрують увагу саме на аналізі приміщень і того, які ділянки вигідніше використовувати для

продажу компаній. Тим не менш, з використанням такого підходу неможливо відповісти на питання, наскільки привабливий для покупців конкретний товар із конкретною маркетинговою ідеєю. Це більшою мірою пов'язано з тим, що A/B тестування було придумано та впроваджено саме в інтернет-середовищі, і саме в таких умовах його робити найзручніше. Так що по-перше, ця ідея не часто переноситься в реальний світ через побоювання власників, що проведення A/B тестів офлайн може бути складним і вимагати значних зусиль та ресурсів. По-друге, власники можуть просто не мати належного розуміння процесу A/B тестування або не мати необхідних знань та ресурсів для його виконання [3].

Проте моя система дозволятиме користуватися всіма перевагами A/B тестування без необхідності значимих вливань. Незважаючи на часті помилки, для налаштування цього процесу зовсім не обов'язково повністю переобладнати магазин, витрачаючи значні ресурси. Достатньо лише вдосконалити свої полиці розумними компактними девайсами у кількості, що відповідає кількості тестів, що проводяться одночасно.

1.3 Постановка задачі

Метою роботи є розробка програмної системи, яка зможе надати користувачеві зручний веб-сайт, на якому він зможе придбати розумний девайс, а потім за допомогою якомога простішого інтерфейсу налаштувати його для проведення тестування. Власник компанії має отримувати звіти з результатами тестувань безпосередньо під час їх виконання, адже тести можуть тривати достатньо довгий період. Результати мають бути представлені у максимально зрозумілому людині вигляді – за допомогою графіків та гістограм. Крім того, окремий модуль має бути присвячений формуванню прогнозів системою на основі отриманих даних за допомогою призначених для цього математичних та статистичних моделей.

Звичайні користувачі зможуть користуватися даною програмною системою для визначення якості товарів і брендів, якими вони зацікавлені. Необхідна система оцінювання, у якій зможе взяти участь кожен авторизований користувач.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Постановка мети

Метою виконання кваліфікаційної роботи є створення програмного продукту для проведення різноманітних тестів товарів в умовах реального світу та прогнозування темпів продажів на основі отриманих результатів.

Розумний девайс буде сконструйований за допомогою датчиків і плат, що надаються платформою Arduino, і дані з нього будуть передаватися на серверну частину програми, написану на ефективній платформі Node.js з використанням багатошарової архітектури мікросервісу і статично типізованої мови програмування TypeScript. Дані зберігатимуться за допомогою розподіленої нереляційної бази даних MongoDB та спеціально створеної для неї ORM Mongoose.

Фронт-енд частина має бути написана за допомогою прогресивного React-фреймворку Next.js з використанням усіх прогресивних підходів до серверного рендерингу та оптимізації SEO. Повинна бути реалізована платіжна система для зручної оплати користувачами своїх покупок, а також авторизація за допомогою бібліотеки Next-Auth.

У будь-якій частині програми потрібно дотримуватися чистої архітектури: код, що легко читається, тестується і масштабується. В основу розробки мають бути закладені патерни SOLID та «банди чотирьох» (Gang of Four patterns) [4], які вважаються запорукою правильного коду. Додаток має бути розбитий на шар уявлення, шар бізнес-логіки та домен, а також шар доступу до даних. Частини, які повторюються в різних репозиторіях, мають бути винесені у загальний пакет та написані лише один раз.

2.2 Основний функціонал системи

Готовим продуктом буде веб-застосунок з розумним пристроєм, що залежить від нього. У системі повинні поєднуватися точний математичний аналіз із простотою та доступністю використання. Інтерфейс та функціонал системи повинен відрізнятися для різних користувачів залежно від ролі.

Звичайний авторизований користувач зможе переглядати та оцінювати товари та їх продукти. Визначати, у яких товарів найвищий рейтинг, а яких придивлятися не варто.

Для власників та адміністраторів, які зареєстрували компанію ж, можливості будуть значно розширені, адже саме вони є цільовою аудиторією, на яку орієнтована бізнес-ідея проекту. Їм будуть доступні великі налаштування для своїх компаній з можливістю створення, редагування та видалення тестів, поповнення баллансу купівля девайсів та перегляд усієї аналітики.

Найбільші можливості будуть надані системним адміністраторам платформи, які оновлюють наявні розумні пристрої, валідують компанії та вирішують будь-які питання власників бізнесів.

2.3 Припущення та залежності

Передбачається, що реальна оплата послуг та впровадження реклами в додаток будуть доступні лише на етапі запуску програми. До цього проводитимуться тестові транзакції.

Передбачається, що користувач буде знайомий із роботою в інтернет-браузері, оскільки програмна система матиме доступ лише через веб-сайт, а також з клавіатурою та мишею.

Передбачається, що користувач сам зможе встановити пристрій у місці, де він хотів би проводити тест, після його покупки та ознайомлення з інструкцією.

Передбачається, що при реєстрації корпоративного акаунту компанія є офіційно зареєстрованою. В іншому випадку послуги не надаватимуться.

Передбачається, що проведення A/B та інших видів тестувань не гарантує компанії моментального приросту продажів, а лише допоможе краще зрозуміти вподобання клієнтів.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Прецеденти використання

Спираючись на вимоги до програмної системи, сформульовані в розділі вище, було проведено моделювання сценаріїв її використання для трьох ключових ролей: користувач, власник компанії та адміністратор. Під час концептуального моделювання предметної області була створена наступна діаграма прецедентів, що зображує функціональне призначення (див. рис. 3.1).

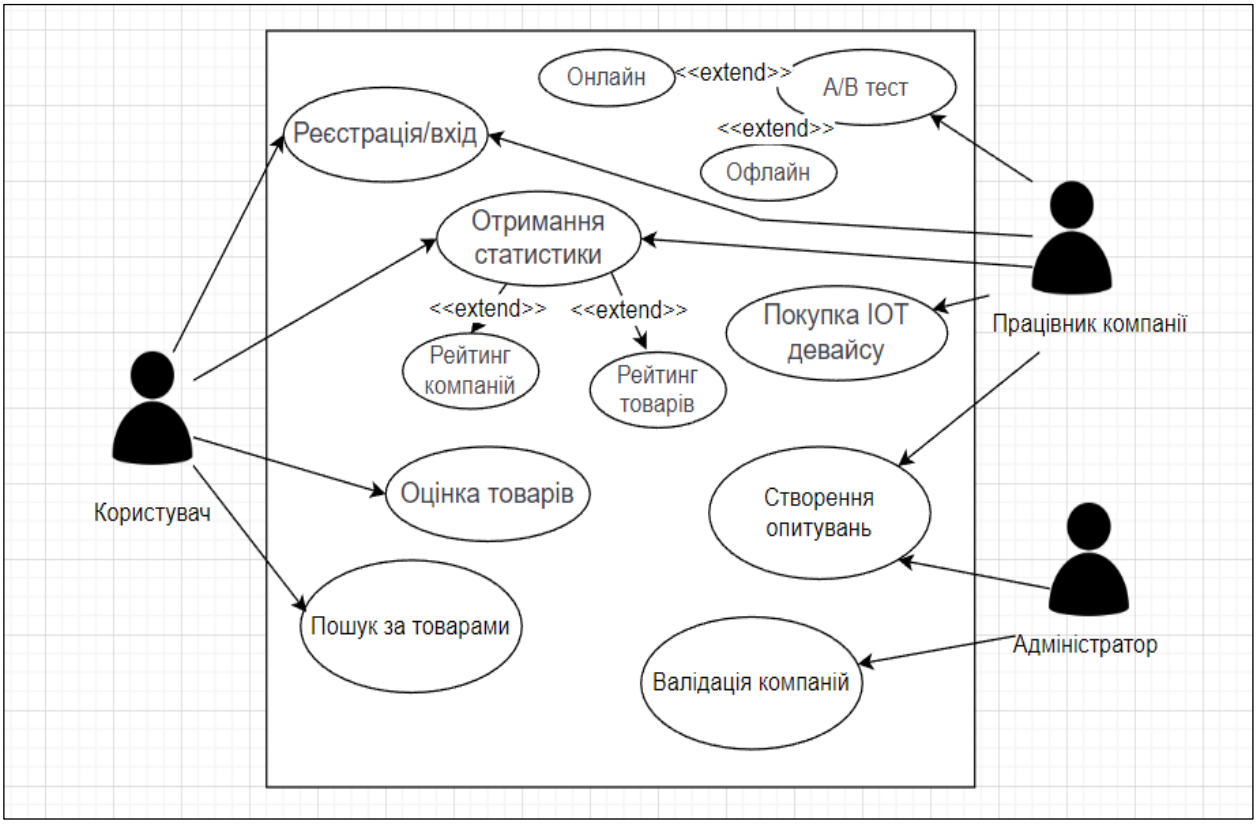


Рисунок 3.1 – UML діаграма прецедентів

Як видно з діаграми, головними сценаріями використання програми будуть дії, пов'язані з тестуванням, отриманням результатів тестів та розумним девайсом. Для доступу до будь-якої з ролей потрібна попередня авторизація.

Потрапляючи в обліковий запис, звичайний користувач опиняється на сторінці із зареєстрованими компаніями в системі, де він відразу може або додати свою компанію, або переглянути інформацію щодо інших. Звичайний користувач зможе переглянути лише поверхневу статистику про компанію, таку як середній

рейтинг компанії або кожного товару серед усіх користувачів, він також може оцінити їх сам. Тим не менш, просунуту статистику, пов'язану з тестами у своїй компанії, зможе отримувати лише користувачів за участю "власник". Користувач автоматично отримує таку роль під час реєстрації компанії. Адміністратори сайту займатимуться переважно валідацією компаній, що реєструються на справжність.

Також в системі буде присутня локалізація, як за мовою, так і за часом, що дозволити власникам легше обробляти великі обсяги даних, які отримуються з тестування.

3.2 Архітектура ПЗ

Було вирішено застосувати архітектуру на основі мікросервісів для розробки back-end частини застосунку для тестування товарів компаній. Цей підхід дозволяє нам додавати нові можливості до системи без змін вже існуючих елементів, знижує ризик поширення помилок на весь застосунок, інкапсулюючи кожний окремий сервіс, оскільки кожен мікросервіс може мати свій набір технологій і базу даних [1].

Під час створення моделі програмної системи для A/B тестування продукції компаній було розроблено кілька UML-діаграм. Архітектура, яку я розробив, має чотири рівні, які демонструються на діаграмі пакетів (див. рис. 3.2).

Шар інтерфейсу користувача забезпечує взаємодію з кінцевими користувачами і буде детальніше розглянутий у наступних розділах. Шар сервісів відповідає за оброблення запитів і виконання бізнес-логіки. Він містить сервіси, які пропонують функціонал системи зовнішнім користувачам і додаткам, обробляючи запити з шару інтерфейсу і викликаючи методи в шарі бізнес-логіки. Шар бізнес-логіки відповідає за виконання бізнес-правил і логіки системи. Він містить компоненти, які обробляють дані і реалізують логіку застосунку. Шар даних забезпечує збереження і доступ до даних для інших рівнів системи. Тут розміщені бази даних, системи файлового зберігання та інші механізми для зберігання даних.

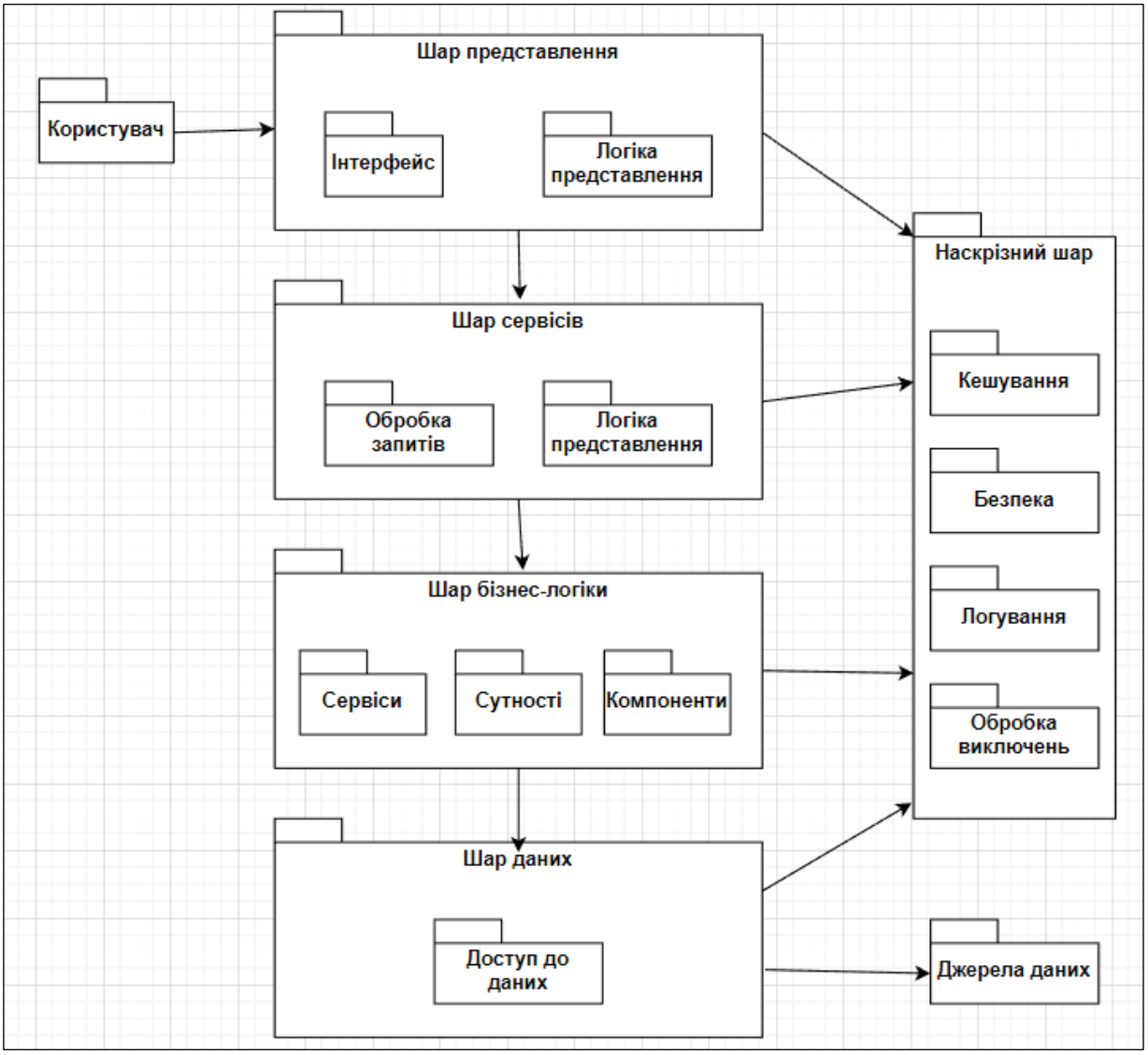


Рисунок 3.2 – UML діаграма пакетів додатку

На діаграмі розгортання (див. рис. 3.3) показано обчислювальні вузли, компоненти і об’єкти, які працюють на цих вузлах під час виконання програми.

Продумуючи архітектуру системи більш детально, слід дотримуватися принципів SOLID, що сприяє створенню гнучких, масштабованих і підтримуваних програмних систем. Ці принципи допомагають розділити функціональність на окремі модулі, забезпечити здатність на зміни і розширення, зберегти сумісність між класами і інтерфейсами, а також покладатися на абстракції, а не конкретні реалізації, які дуже часто можуть змінюватись відповідно до нестабільних тенденцій на ринку.

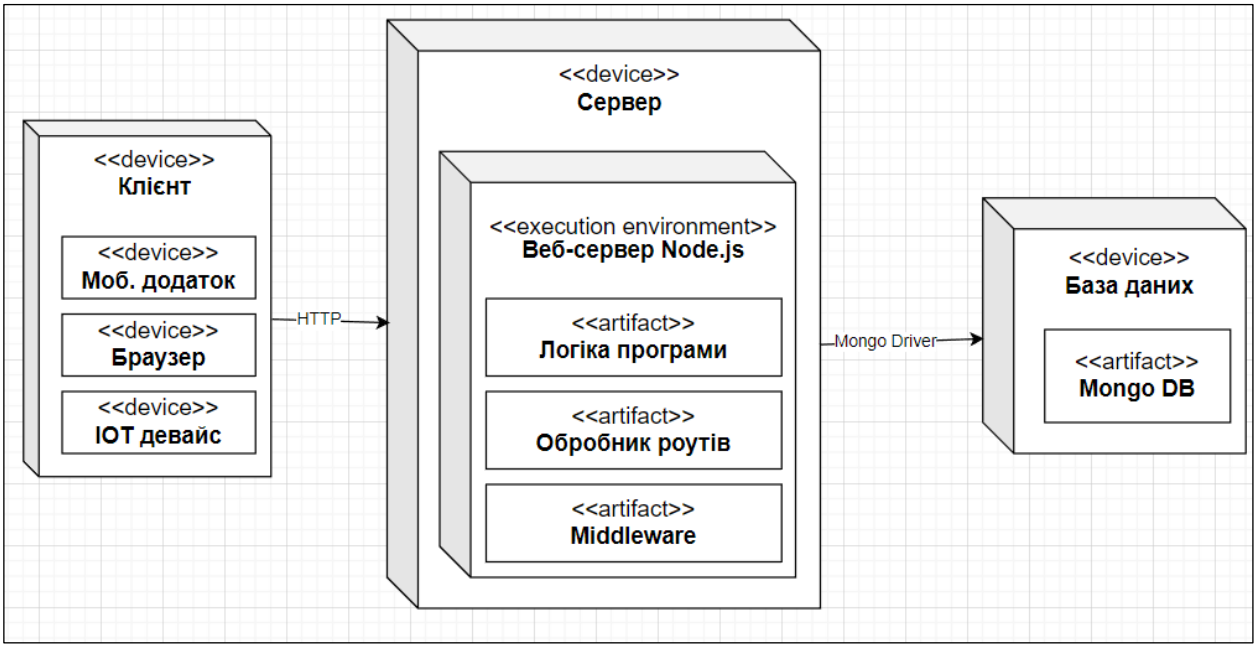


Рисунок 3.3 – UML діаграма розгортання

3.3 Проєктування структури бази даних

Роблячи вибір між реляційною та нереляційною базою даних у моєму проекті, я відштовхувався насамперед від гнучкості, необхідної для моєї системи. З часом структура зберігання таких даних, як результати тестів або смарт-девайси, може змінюватися у зв'язку з появою нових ідей або вимог до процесу тестування. У разі реляційного підходу до сховищ, у таких ситуаціях програмістам зазвичай доводиться здійснювати безпечні міграції, складність яких може ставати надмірною і зовсім необов'язковою для програмних систем зі швидко зростаючим обсягом даних. Саме в таких ситуаціях стаються в нагоді NoSQL бази даних [10].

Найпопулярнішою нереляційною СУБД на сьогоднішній день є MongoDB. MongoDB вирізняється своєю документ-орієнтованою структурою, яка дозволяє зберігати дані у вигляді гнучких документів, уникаючи необхідності встановлення жорсткої схеми даних наперед. Ця особливість сприяє швидкій адаптації до змін у вимогах і дозволяє розширювати можливості системи без складних процесів міграції даних.

MongoDB відома своєю здатністю ефективно обробляти великі масиви даних, забезпечуючи швидке зберігання, відшукання та обробку інформації, що є

критично важливим для систем з великою кількістю користувачів або високим рівнем навантаження. Додатково, MongoDB пропонує широкий спектр функціональності та інструментів, які полегшують процес розробки та управління базами даних. Наприклад, інтеграція з Node.js дозволяє зручно працювати з даними у форматі TypeScript об’єктів, що є моєю переважною мовою програмування, спрощуючи інтеракцію з базою даних у серверній частині застосунку.

Стержневими сутностями моєї бази даних є сутності «Користувач», «Компанія», «Продукт», «Девайс», «Тест», «Результат» та «Придбаний девайс».

Сутності «Компанія» необхідно надати доступ для придбання розумних пристроїв, представлення товарів, замовлення тестів та перевірки результатів (див. рис. 3.4).

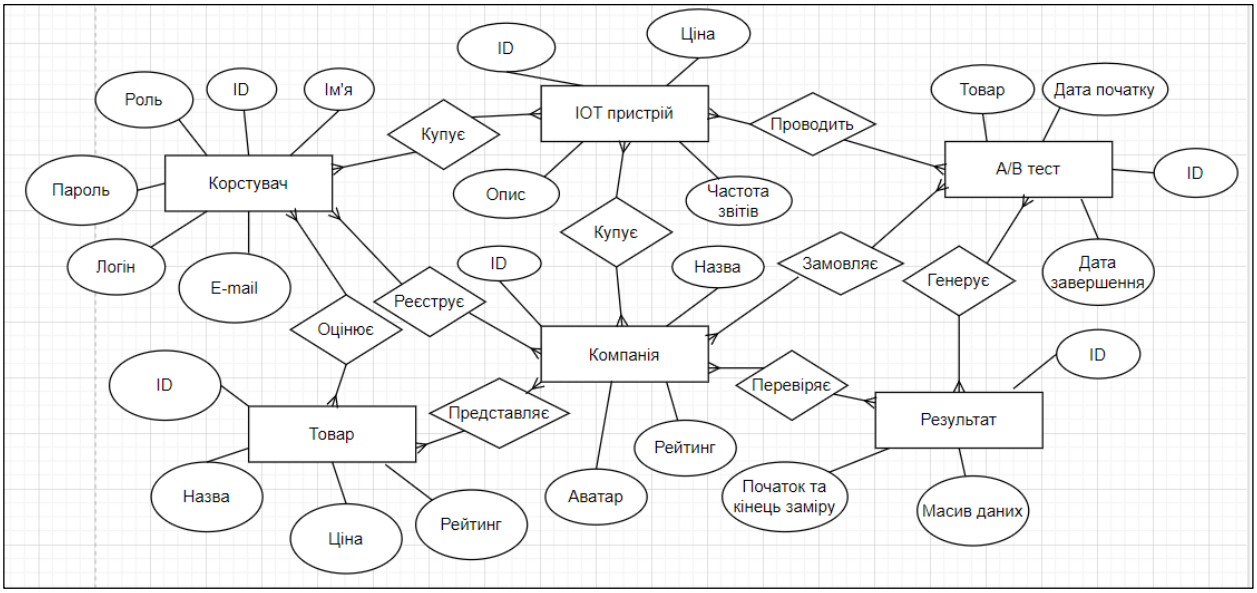


Рисунок 3.4 – ER-діаграма

Результати створюватимуться за допомогою A/B тестувань, які будуть проводитися з використанням спеціалізованих розумних пристроїв. Водночас, сутність “Користувач” матиме можливість реєструвати та оцінювати товари [7].

У системі MongoDB кожен окремий елемент даних відомий як колекція. Використовуючи мікросервісну архітектуру для розробки програмного забезпечення, ми маємо безліч незалежних компонентів, кожен з яких виконує

певні функції і має свої обов’язки. Це унеможливлює розміщення всіх колекцій у єдиній базі даних. Моя система складається з чотирьох сервісів.

Сервіс “User” займається управлінням користувачами системи, включаючи реєстрацію нових користувачів, аутентифікацію, авторизацію та управління профілями, такими як зміна пароля та вилучення даних користувачів. Основною колекцією цього сервісу є “users” у базі.

Сервіс “Test” безпосередньо відповідає за проведення тестувань продукції, запускаючи пристрої та збираючи результати. Він містить три основні колекції: “tests”, “results” та “products” (див. рис. 3.5).

Сервіс “Company” виконує функції керування компаніями та організаціями у системі. Він надає можливості для заснування нових компаній, модифікації даних про існуючі компанії, включно з фінансовими операціями, та закупівлею обладнання для тестувань. В базі даних цього сервісу розміщені такі колекції як “companies”, “devices” (доступні для купівлі пристрої) та “purchases” (придбане обладнання).

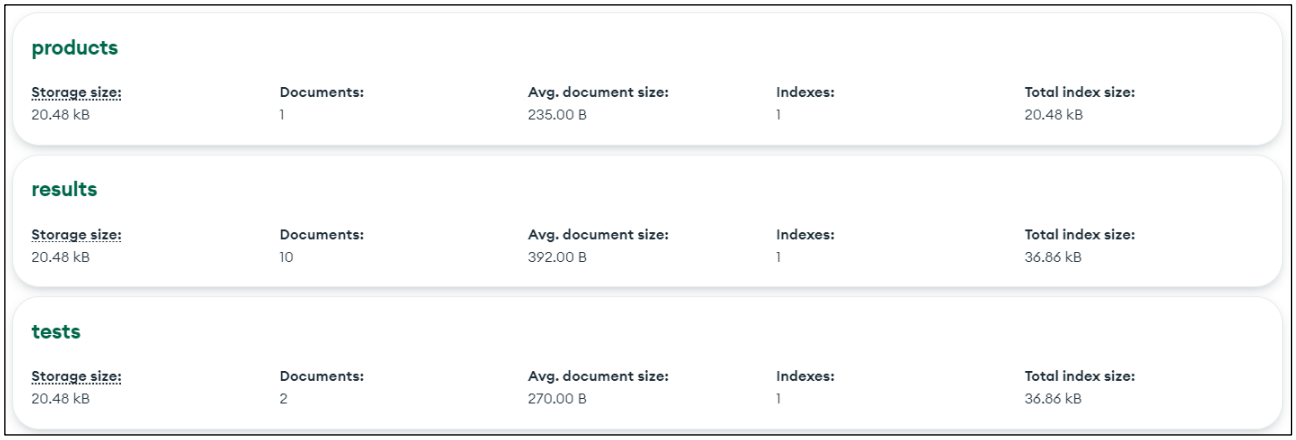


Рисунок 3.5 – Колекції сервісу «test»

Сервіс “Gateway” служить як вхідний портал до системи, забезпечуючи взаємодію між зовнішніми запитами та внутрішніми сервісами. Він обробляє запити клієнтів, направляє їх до відповідних сервісів і здійснює повернення відповідей назад до клієнтів.

3.4 Проєктування розумного девайсу ПЗ

Основна мета моєї розробки програмного забезпечення для A/B аналізу продукції - створення пристрою на базі Internet of Things (IoT), який би моніторив інтерес покупців до фізичних товарів. У цьому мені стане в нагоді відкрите обладнання, яке пропонує платформа Arduino. Перша плата Arduino була створена для того, щоб зробити процес виготовлення електроніки більш простим і доступним для широкого кола людей. Нині Arduino користується великою популярністю у світі і застосовується у численних проектах.

Головні переваги платформи Arduino - це її доступність і легкість у використанні. Arduino має інтуїтивно зрозумілий інтерфейс, що дозволяє ефективно почати роботу.

Для мого проекту я обрав плату Arduino UNO, яка є однією з найбільш вживаних плат для створення електронних пристроїв і програмування мікроконтролерів. Вона забезпечує підтримку великої спільноти, яка може надати необхідну інформацію з даної теми. Однією з переваг Arduino UNO є також можливість розширення завдяки численним портам вводу-виводу і доступним модулям.

Ключовим компонентом для збору даних слугуватиме ультразвуковий датчик HC-SR04 (див. рис. 3.6).

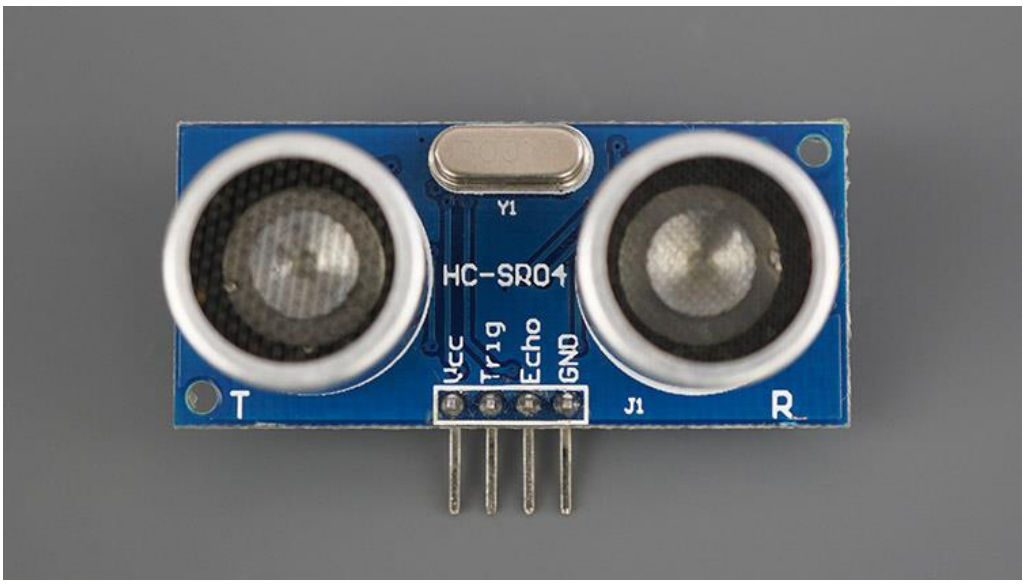


Рисунок 3.6 – Датчик HC-SR04

Даний датчик використовує принцип ехолокації для вимірювання відстані до предмета. Ультразвуковий передавач генерує імпульс високої частоти (40 кГц), який розповсюджується у повітрі і відбивається від об’єкта назад до датчика. Приймаючи звуковий сигнал, датчик фіксує час, потрібний для подорожі до об’єкта і назад, дозволяючи обчислити відстань у сантиметрах. Після цього залишається лише відсіяти непотрібні дані, видаляючи зайві відстані з записів.

Згідно з описаною логікою була розроблена діаграма станів, яка демонструє ключові стадії роботи інтелектуального пристрою (див. рис. 3.7).

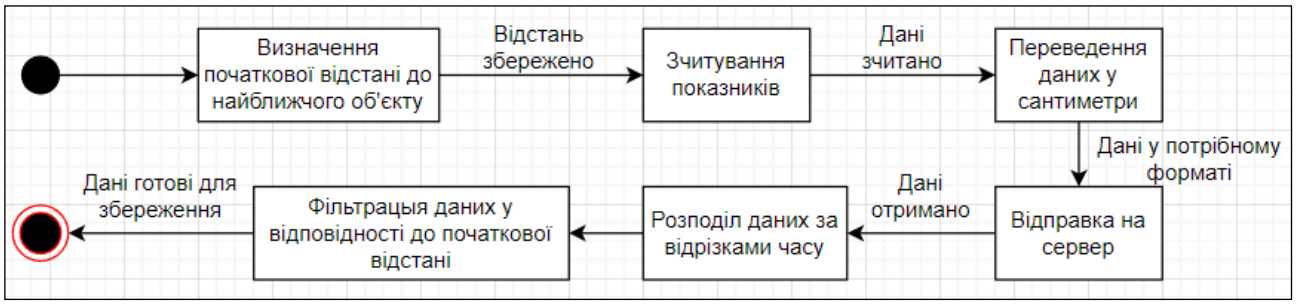


Рисунок 3.7 – Діаграма станів

Ультразвуковий датчик HC-SR04 оснащений чотирма контактами. Контакт VCC призначений для під’єднання до позитивного джерела живлення. Його слід з’єднати з 5V контактом на платі Arduino. Контакт GND використовується для з’єднання з негативним джерелом живлення і загальною «землею», і на платі є відповідний контакт для цього. Контакти Trig та Echo призначені для відправлення та прийому ультразвукових сигналів відповідно. Їх можна підключити до будь-яких цифрових контактів на Arduino UNO. Використовуючи цю конфігурацію, я зібрав схему для мого смарт-пристрою (див. рис. 3.8).

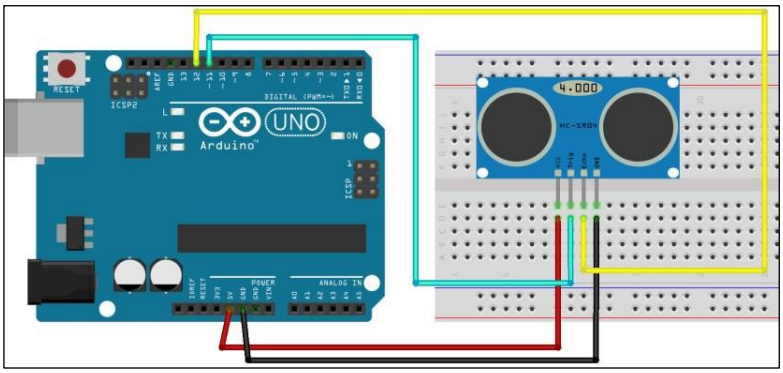


Рисунок 3.8 – Схема підключення

3.5 Проєктування серверної частини ПЗ

Як вже було сказано вище, для розробки серверної REST API (див. додаток Б) частини проекту використовується мікросервісна архітектура та платформа Node.js. Окрім же згаданих мікросервісів для користувачів, тестів і компаній, на бекенді необхідний ще один сервіс-шлюз, що буде слугувати точкою входу, об’єднуючі всі частини.

Як фреймворк для розробки був обраний Express, адже це один і найпростіших інструментів для побудови RESTful додатків з використанням middleware – механізм обробки запитів перед тим, як вони досягнуть основного обробника запиту, який часто використовують для таких цілей як логування, обробка помилок або авторизація. Це саме те, що потрібно в нашому випадку, адже однією з ключових цілей є створення безпечного та ефективного механізму обробки запитів від клієнтських додатків, забезпечення авторизації користувачів за допомогою сесій, які зберігаються в базі даних, та захист ендпоінтів від несанкціонованого доступу до цінної інформації з тестування.

Для досягнення належної якості програмного коду було розроблено модульну архітектуру, де кожен мікросервіс має власну базу даних та набір контролерів, які відповідають за обробку запитів та забезпечення взаємодії з базою даних. Моделі даних описані відповідно до функціональності кожного сервісу та знаходяться в окремих папках database. Контролери, які відповідають за обробку HTTP-запитів, розміщені в папках api, тоді як функції взаємодії з моделями знаходяться в папках services. Така структура проекту дозволяє забезпечити високу модульність та гнучкість системи.

Найбільш значущим аспектом розробленого REST API є його здатність до авторизації та захисту ендпоінтів від несанкціонованого доступу. Для цього використовується механізм сесій, де у базі даних користувача зберігається токен, який використовується для перевірки ідентифікації та авторизації користувача при кожному запиті до сервера. Такий підхід забезпечує безпеку та конфіденційність даних, що обробляються в системі, та відповідає сучасним стандартам розробки веб-додатків.

3.6 Проєктування клієнтської частини ПЗ

Front-end частина веб-додатку розробляється з використанням прогресивного фреймворку Next.js, який дозволяє використовувати всі переваги серверних компонентів React та серверного рендерингу (SSR) для оптимізації швидкості завантаження сторінок через сервер (не на стороні клієнта) та покращення їхнього індексування пошуковими системами. Одним із ключових принципів розробки є процес гідратації, який забезпечує перетворення статично згенерованих на сервері HTML сторінок у динамічні шляхом додавання JavaScript після первинного рендерингу. Зокрема, компоненти, що вимагають взаємодії з користувачем, виносяться як клієнтські і розташовуються якнайглибше в ієрархії компонентів сторінки, щоб максимально використовувати можливості SSR.

Додатково, для забезпечення безпеки авторизації використовується зберігання токенів в HttpOnly куках замість локального сховища браузера. Це забезпечує додатковий рівень захисту, оскільки HttpOnly куки неможливо зчитати за допомогою JavaScript, унеможливлюючи потенційний доступ до них з боку зловмисників [5].

У контексті реалізації платіжних операцій, для поповнення балансу аккаунту користувача обрано платформу Stripe. Використання даної платіжної системи забезпечує зручний та безпечний механізм проведення операцій, забезпечуючи високий рівень надійності та швидкості обробки транзакцій.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Вибір засобів програмної реалізації

Моя програмна система є комплексним програмним продуктом, що складається з трьох великих компонентів. Відповідно до потреб було обрано кілька ключових технологій, кожна з яких виконує свою специфічну роль у загальній архітектурі рішення.

Апаратну частину системи реалізовано за допомогою Arduino. Це платформа з відкритим вихідним кодом, яка забезпечує простий спосіб підключення різних датчиків і виконавчих механізмів. Вибір Arduino зумовлений його широкими можливостями, зручністю використання та доступністю великої кількості бібліотек, що спрощують розробку. Крім того, наявність великої спільноти користувачів Arduino забезпечує легкий доступ до документації, прикладів та підтримки, що значно полегшує процес розробки апаратної частини системи.

Серверна частина системи побудована на основі мікросервісної архітектури з використанням Node.js та фреймворку Express. Node.js було обрано завдяки його високій продуктивності, асинхронній природі виконання та здатності обробляти велику кількість одночасних запитів. Використання фреймворку Express дозволяє спростити створення RESTful API, що є важливим для забезпечення зв’язку між різними компонентами системи. Така комбінація технологій дозволяє досягти гнучкості та масштабованості серверної частини, що є критичним для комплексних програмних систем.

Фронтенд системи розроблено з використанням Next.js, що є популярним фреймворком для React. Next.js обрано через його підтримку сервер-сайд рендерингу (SSR) та інкрементального статичного регенерування (ISR). Ці можливості забезпечують швидке завантаження сторінок, покращують SEO та підвищують загальну продуктивність веб-додатку. Крім того, використання Next.js дозволяє легко інтегруватися з іншими компонентами системи та забезпечує зручність у розробці та підтримці коду.

На всіх рівнях системи, окрім апаратного, використовується мова

програмування TypeScript. TypeScript надає можливості статичної типізації JavaScript, що сприяє зменшенню помилок та полегшує рефакторинг коду. Крім того, використання TypeScript поліпшує процеси оптимізації та деоптимізації під час JIT-компіляції програми, що робить код мономорфним та поліпшує його продуктивність.

4.2 Опис Smart Device програмного застосунку

Через необхідність взаємодії з платформою Arduino при проектуванні розумного пристрою для моєї системи, данна частина проекту є єдиною де використовується не мова програмування TypeScript, а C++.

Насамперед у коді Arduino визначаються змінні портів для випромінювання та отримання ультрозвукових хвиль, необхідних для підрахунку відстані. Відповідно створюються змінні під назвою trigPin та echoPin для тригера та ехо-контакту зі значеннями 11 та 12, як це було зазначено на схемі вище. Також нам знадобляться додаткові службові змінні для запам'ятовування curDistance, previousDistance, previousTime і duration.

```
1. int trigPin = 11;
2. int echoPin = 12;
3. unsigned long duration, previousTime = 0;
4. int curDistance, previousDistance = 0;
```

У будь-якій програмі Arduino необхідно виконати етап ініціалізації в методі setup(). На цьому етапі визначається послідовний порт із передачею даних на швидкості 9600 бод, встановлюється тригерний контакт як вихідний, а ехо-контакт – як вхідний.

```
1. void setup() {
2.   Serial.begin (9600);
3.   pinMode(trigPin, OUTPUT);
4.   pinMode(echoPin, INPUT);
5. }
```

Основна ж робота програми описана в методі loop(). Представлений нижче

код виконує відправку сигналів. Для того щоб отримати чистий HIGH імпульс, спочатку надсилається короткий (5 мікросекунд) LOW імпульс. Потім, за допомогою функції pulseIn(), отримуємо час у мілісекундах, який затратив ехо-датчик на повернення імпульсу.

Це значення є часом, який сигнал провів у шляху до найближчого об'єкта і назад, тому його треба поділити на 2. Враховуючи швидкість звуку 343 м/с, це значення можна поділити на 29.1, щоб отримати відстань у сантиметрах. Оскільки виміри проводяться кожні півсекунди (500 мілісекунд), багато даних можуть бути ідентичними до попередніх і не мають значення. Тому з допомогою блоку if відсіваються однакові значення, і записуються лише ті, що відрізняються від попередньої ітерації хоча б на 5 сантиметрів.

```
1. void loop() {
2.   digitalWrite(trigPin, LOW);
3.   delayMicroseconds(5);
4.   digitalWrite(trigPin, HIGH);
5.   delayMicroseconds(10);
6.   digitalWrite(trigPin, LOW);
7.
8.   pinMode(echoPin, INPUT);
9.   duration = pulseIn(echoPin, HIGH);
10.
11.   curDistance = (duration/2) / 29.1;
12.
13.   if (abs(curDistance - previousDistance) >= 5) {
14.     unsigned long currentTime = millis();
15.     Serial.print(curDistance);
16.     Serial.print("cm ");
17.     Serial.print(currentTime);
18.     Serial.println("ms");
19.     previousDistance = curDistance;
20.     previousTime = currentTime;
21.   }
22.
23.   delay(500);
24. }
```

Запустивши цю програму і відкривши серійний монітор, можна побачити бажані результати у сантиметрах (див. рис. 4.1).

Усю подальшу обробку результатів, отримуваних за допомогою смарт-девайсу будо вирішено виконувати на платформі Node JS. Задля того, щоб отримати зв’язок між сервером Node.js та рядками, що надходять на серійний

монітор зі смарт-девайсу, я використовував npm-бібліотеку serialport. Ця бібліотека дозволяє підключатися до пристроїв, що підтримують передачу даних через серійний порт.

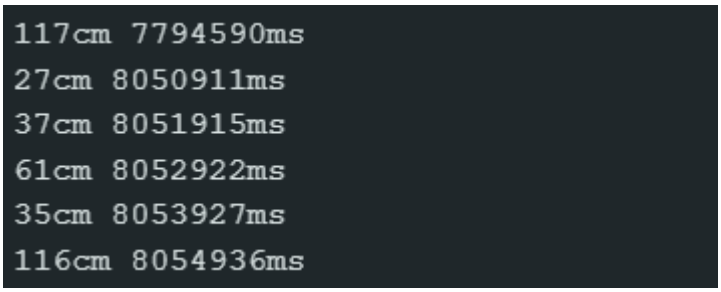


Рисунок 4.1 – Результати виконання програми

За допомогою класу SerialPort я створив екземпляр порту, вказавши необхідні параметри, такі як порт, швидкість передачі даних, біт даних, паритет та інші. Потім я встановив потік даних за допомогою методу pipe, який зв'язує порт із парсером даних. Для обробки отриманих даних я використовував парсер Readline, який розділяє дані на рядки.

При отриманні нових даних за допомогою події data я додавав їх до масиву результатів. Після завершення тестування я закривав порт і повертав масив результатів за допомогою функції resolve, надавану промісом.

```
1. export const runTestingForDuration = (duration: number) => {
2.  return new Promise((resolve, reject) => {
3.   let res = [] as string[];
4.   const parsers = SerialPort.parsers;
5.   const parser = new parsers.Readline({
6.    delimiter: "\r\n",
7.   });
8.   const port = new SerialPort("COM3", {
9.    baudRate: 9600,
10.    dataBits: 8,
11.    parity: "none",
12.    stopBits: 1,
13.    flowControl: false,
14.   });
15.   port.pipe(parser);
16.   parser.on("data", function (data) {
17.    res.push(data);
18.   });
19.   setTimeout(() => {
20.    port.close();
21.    resolve(res);
22.   }, duration);
```

```
23.  });
24. };
```

4.3 Опис серверної частини програми

У сучасному світі розробки програмного забезпечення мікросервісна архітектура набуває все більшої популярності завдяки своїй гнучкості, масштабованості та стійкості. Цей підхід дозволяє поділити велику систему на менші, автономні сервіси, кожен з яких виконує окрему функцію та може бути розроблений, розгорнутий і масштабований незалежно від інших.

Тим не менш, в кожному з сервісів мого серверного застосунку я використовуватиму одне й те саме стандартне розбиття коду на три шари: контролери, сервіси та репозиторій. Застосування такого підходу має кілька переваг. По-перше, стандартизація шарів дозволяє зберігати єдність у розробці, що полегшує підтримку і розширення системи. По-друге, розділення відповідальностей між контролерами, сервісами і репозиторіями сприяє кращій організації коду та знижує складність кожного окремого компонента.

Рівень контролерів відповідає за обробку запитів, які надходять від користувачів або інших сервісів, та перенаправлення цих запитів до відповідних сервісів. Сервісний шар містить бізнес-логіку, яка визначає, як повинні оброблятися дані, та забезпечує виконання основних операцій. Репозиторій даних забезпечує взаємодію з базою даних, абстрагуючи деталі доступу до даних від інших шарів [9].

4.3.1 Опис сервісу користувача

Сервіс user відповідатиме як за управління акаунтами в системі, так і за авторизацію. Саме тому в директорії сервісу знаходяться два контролери - user та auth.

Авторизація реалізована за стратегією зберігання токенів з поточними сесіями в базі даних, тож я не використовую JWT. Це надає впевненості, що токени не будуть перехоплені на стороні клієнта. Паролі користувачів хешуються

за допомогою алгоритму bcrypt, що робить їх захищеними від несанкціонованого доступу. У дотриманні всіх цих паттернів безпеки можна переконатися, глянувши на метод реєстрації користувача (див. додаток Г).

На окрему увагу заслуговує мідделвейр-функція для перевірки авторизації користувача, що надіслав запит. У цій функції checkAuth ми витягуємо токен з cookie запиту, відправленого користувачем, і порівнюємо його з діючим cookie сесії, записаної в базі даних. Така функція може бути використана перед обробкою будь-якого запиту, якщо його потрібно зробити обмеженим лише авторизованих користувачів.

```
1. export const checkAuth = async (
2.  req: express.Request,
3.  res: express.Response,
4.  next: express.NextFunction
5. ) => {
6.  try {
7.   const sessionToken = req.cookies.COOKIE_AUTH;
8.   if (!sessionToken) {
9.    throw AppError.unauthorised("You are not logged in.");
10.   }
11.   const existingUser = await UserModel.findOne({
12.    "authentication.sessionToken": sessionToken,
13.   });
14.
15.   if (!existingUser) {
16.    throw AppError.unauthorised("You are not logged in.");
17.   }
18.
19.   merge(req.body, {
20.    identity: existingUser._id,
21.    identityLogin: existingUser.login,
22.   });
23.   return next();
24.  } catch (err) {
25.   next(err);
26.  }
27. };
```

У класі UserService зберігаються методи для отримання всіх користувачів, отримання одного користувача за ID, логіном або токеном, а також отримання профілю за токеном. Ця логіка повністю відокремлена від логіки, пов’язаної з авторизацією і є набагато простішою:

```
async GetProfileByLogin(login: string) {
  try {
   const existingUser = await UserModel.findOne({ login });
   if (!existingUser) {
    throw AppError.badRequest("User doesn't exist");
   }
   return existingUser;
  } catch (err) {
   throw err;
  }
 }
```

4.3.2 Опис сервісу компаній

Різні мікросервіси системи мають бути ізольовані один від одного, що забезпечує підвищену стійкість і масштабованість системи в цілому. Ізоляція досягається завдяки тому, що кожен мікросервіс має власну базу даних і виконується в окремому середовищі. Тим не менш, сервісам також необхідно спілкуватися один з одним. Яскравим прикладом такої необхідності може бути автоматична зміна ролі користувача зі звичайного користувача до власника при реєстрації нової компанії в програмі. Обробник та логіка створення нової компанії повинна знаходитися в сервісі company, але метод зміни ролі користувача знаходиться в сервісі user. Одним зі способів спілкування між такими двома незалежними сервісами є протокол HTTP, який також використовується і у всіх інших запитах моєї системи. Після успішного створення компанії програма просто звертатиметься до необхідного методу з іншого сервісу за допомогою POST запиту HTTP:

```
const { name, avatarUrl, identity, description, identityLogin } =
    req.body;
const data = await service.CreateCompany({
    name,
    avatarUrl,
    description,
    owner: identityLogin,
});
const payload = {
    event: "CHANGE_ROLE",
    data: { newRole: "Owner", _id: identity },
};
await axios.post(`${BASE_URL}/user/app-events/`, { payload });
res.status(200).json(data);
```

Відповідно, для того, щоб диференціювати запити, необхідні для спілкування сервісів між собою в рамках мого back-end продукту від звичайних зовнішніх запитів до REST API у списку моїх контролерів, створюється окремий роут "/app-events", який перенаправлятиме ці запити до потрібних методів. , логіруючи звернення у рамках мікросервісної архітектури.

```
1. export const appEvents = async (app: express.Application) => {
2.   const serviceDevice = new DeviceService();
3.   const serviceCompany = new CompanyService();
4.
5.   app.use('/app-events', async (req: express.Request, res:
express.Response) => {
6.     const { payload } = req.body;
7.     const result =
8.       payload.event === 'GET_COMPANY_BY_ID'
9.         ? await serviceCompany.SubscribeEvents(payload)
10.          : await serviceDevice.SubscribeEvents(payload);
11.     console.log('============= App Event ===============');
12.     return res.status(200).json(result);
13.   });
14. };
```

Сам сервіс компаній містить у собі операції над трьома сутностями: компанії, девайси і покупки, що здійснювали компанії на платформі. На даний момент існує лише один варіант девайсів доступних для покупки, а саме датчик для вимірювання відстані до найближчого об'єкта. До кожного такого купленого девайсу буде застосовано наведену нижче схему, в якій записані налаштування пристрою за умовчанням, чи є він вільним від тестування в даний момент і чи доставлений він покупцю:

```
1. const PurchaseSchema = new mongoose.Schema({
2.  companyId: {
3.   type: String,
4.   required: true,
5.  },
6.  deviceId: {
7.   type: String,
8.   required: true,
9.  },
10.  isFree: {
11.   type: Boolean,
12.   default: true,
13.  },
14.  defaultReportingFrequency: {
```

```
15.   type: String,
16.   default: "Every hour",
17.  },
18.  defaultTrackingRange: {
19.   type: Number,
20.   default: 3,
21.  },
22.  delivered: {
23.   type: Boolean,
24.   default: false,
25.  },
26. });
```

Функціонал поповнення грошового балансу компаній для здійснення покупок всередині програми буде використовуватися стороння бібліотека платіжної системи Stripe. Згенерувавши новий API-ключ на офіційному сайті платформи, можна створити екземпляр класу Stripe, який володіє всім необхідним функціоналом для створення сесії поповнення рахунку банківською карткою (див. додаток Д). Варто зазначити, що API-ключ повинен зберігатися виключно в .env, файлі, недоступному стороннім.

4.3.3 Опис сервісу тестів

У сервісі управління тестуванням товарів компаній щодо інтересу з боку покупців повинні бути дві колекції даних: тести і результати тестів, що мають зв'язок "один до багатьох" по первинному полю id. Таке рішення призведе до нормалізації даних і зручнішого способу стеження за оновленнями результатів тестування в режимі реального часу.

Однією з найскладніших функцій даного сервісу є метод запуску тестування, адже коли користувач здійснить подібний запит, серверу Node.js буде необхідно самостійно зв'язатися з датчика і почати записувати дані, які він зчитує, в базу даних. Для цього спочатку необхідно встановити зв'язок з портом комп'ютера, до якого підключений датчик. З цим мені допоможе стороння бібліотека npm serialport. Користуючись можливістю створювати асинхронні дії за рахунок механізму Node.js під назвою promise, я створюю відкладену дію на тривалість тестування, задану користувачем. У цій функції я заношу дані, що надходять з датчика на порті, вказаному в конфігурації, в результуючу змінну, і

після заданого часу повертаю цю змінну за допомогою колбека Promise під назвою resolve.

```
1. export const runTestingForDuration = (
2.   duration: number
3. ): Promise<Approach[]> => {
4.   return new Promise((resolve, reject) => {
5.     let res: Approach[] = [];
6.     const parsers = SerialPort.parsers;
7.     const parser = new parsers.Readline({
8.       delimiter: "\r\n",
9.     });
10.     const port = new SerialPort("COM3", {
11.       baudRate: 9600,
12.       dataBits: 8,
13.       parity: "none",
14.       stopBits: 1,
15.       flowControl: false,
16.     });
17.     port.pipe(parser);
18.     parser.on("data", function (data) {
19.       const numbers = data.match(/\d+/g);
20.       res.push({ distance: numbers[0], time: numbers[1] });
21.     });
22.     setTimeout(() => {
23.       port.close();
24.       resolve(res);
25.     }, duration);
26.   });
27. };
```

Незважаючи на те, що в частині, що стосується розумного девайсу на Arduino, я вже згадував про первинну обробку даних, що зчитуються пристроєм, інформація про результати тесту, яку ми отримуємо на сервері на даному етапі, все ще досить "сира" з кількох причин. Перша полягає в тому, що при роботі датчик може повертати всього два принципово різних типи результатів: початкова відстань до найближчого об'єкта, яка одночасно є максимально можливою (наприклад, протилежна полиця в магазині) і змінена відстань, яка записується коли до датчика наближається людина. Очевидно, що перший тип результатів не цікавить кінцевого користувача, і всі рядки, в яких записано відстань за замовчуванням, потрібно відфільтрувати. Інша проблема у потокових результатах полягає в тому, що навіть серед тих результатів, які насправді є наближеннями щодо початкової дистанції, є ознакою зацікавленості насправді. Датчик зчитує

результати навколишнього світу кожні 500 мілісекунд, через що навіть ті випадки, коли людина просто пройшла повз продукт, можуть записуватися їм як наближення. Проте справжню зацікавленість людини можна фіксувати лише в тих випадках, коли вона затримується у товару хоча б на кілька секунд. Саме тому з підсумкових результатів також необхідно відфільтрувати ті наближення, тривалість яких не перевищує 2 секунди. Все це можна зробити за допомогою вбудованих JavaScript-функцій для обробки масивів (див. додаток Е).

4.3.4 Опис реалізації математичного аналізу результатів

Навіть належно оброблені результати тестрів, що приходять з датчиків, матимуть не настільки велику цінність для кінцевого користувача, як побудовані на їх основі математичні моделі, здатні допомагати з прийняття грамотних рішень щодо товарів і навіть прогнозувати подальші темпи продажу.

Для цієї мети мною були вивчені три різних способи математичного аналізу даних: біноміальний розподіл, розподіл Пуассона і дисперсний аналіз (ANOVA).

Біноміальний розподіл – це розподіл ймовірностей для дискретної випадкової величини, яка підраховує кількість успіхів у фіксованій кількості незалежних випробувань, кожне з яких має два можливі результати: успіх або невдача (див. рис. 4.2).

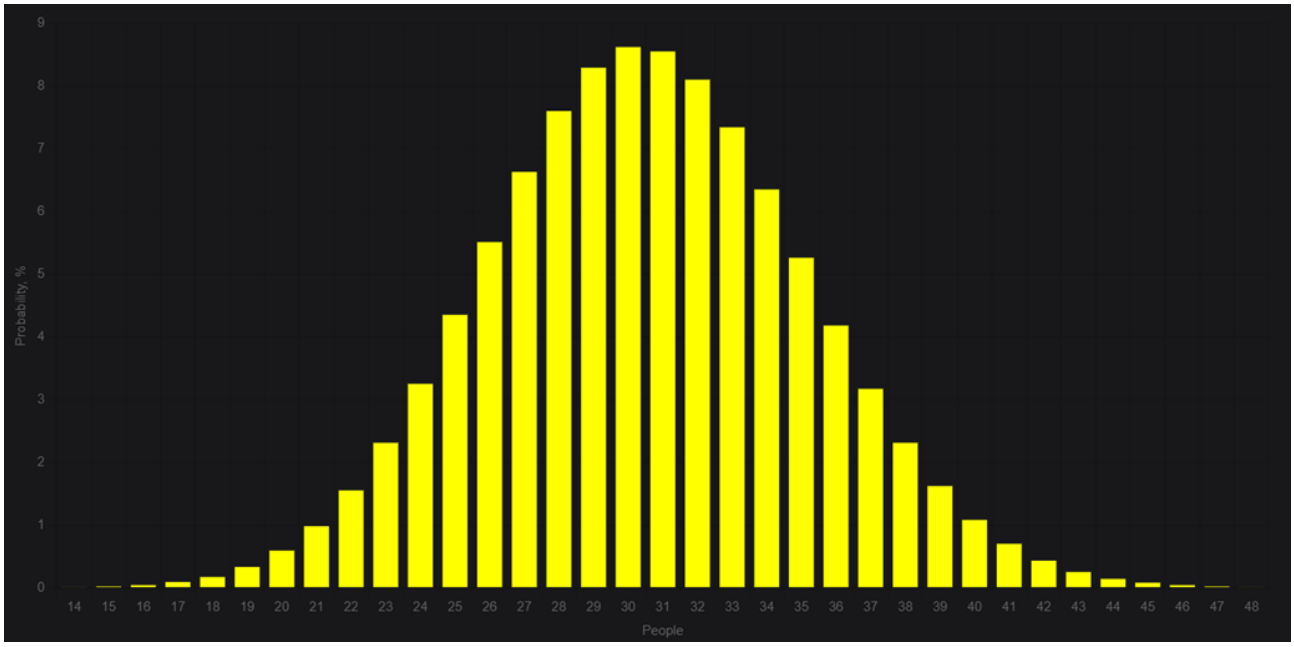


Рисунок 4.2 – Приклад біноміального розподілу для 100 випадків

Ймовірність отримати рівно k успіхів у n випробуваннях обчислюється за формулою (1):

$$P(X = k) = \binom{n}{k} \times p^k \times (1-p)^{n-k}, \quad (1)$$

де $\binom{n}{k}$ – кількість комбінацій k продажів при n користувачів,

p та $(1-p)$ – ймовірності успіху та невдачі при кожній спробі продажу.

Спираючись на цю формулу, нескладно розробити функцію саме для мого випадку випробувань:

```
1. async BinomialDistribution(
2.   testId: string,
3.   peopleNumber: number = 100
4. ): Promise<BinomialResult> {
5.          const   {   allClients,   allApproaches   }   =   await
this.GetFilteredResults(testId);
6.   const p = allApproaches / allClients;
7.   const distributionArray = [];
8.   for (let i = 0; i <= peopleNumber; i++) {
9.     const calculatedProbability =
10.       this.combinations(peopleNumber, i) *
11.       Math.pow(p, i) *
12.       Math.pow(1 - p, peopleNumber - i);
13.         const  probability  =  parseFloat((calculatedProbability  *
100).toFixed(2));
14.     if (probability > 0) {
15.       distributionArray.push({
16.         people: i,
17.         probability,
18.       });
19.     }
20.   }
21.   return distributionArray;
22. }
```

Розподіл Пуассона – це розподіл ймовірностей для дискретної випадкової величини, яка описує кількість подій, що відбуваються у фіксованому інтервалі часу або просторі при відомій середній частоті подій і за умови, що ці події відбуваються незалежно одна від одної [8].

Ймовірність того, що у фіксованому інтервалі відбудеться рівно k подій, обчислюється за формулою (2):

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad (2)$$

Вважаючи, що у випадку моєї програми подіями будуть наближення клієнтів до окремого товару, маючи результати тестів, можна реалізувати функцію розподілу Пуассона:

```
async PoissonDistribution(
  testId: string,
  start: string,
  end: string
): Promise<PoissonResult> {
    const lambda = await this.poissonLambdaCount(testId, start, end);
  let probabilities = [];
  let k = 0;
  while (true) {
    const probability = parseFloat(
                ((Math.pow(lambda, k) * Math.exp(-lambda)) / this.factorial(k)).toFixed(
        2
      )
    );
    if (k > lambda && probability === 0) {
      break;
    }
    probabilities.push(probability);
    k++;
  }
  const probabilitySums: number[] = probabilities
    .reverse()
    .reduce((acc: number[], cur: number) => {
      const lastSum = acc.length > 0 ? acc[acc.length - 1] : 0;
      acc.push(lastSum + cur);
      return acc;
    }, []);
  probabilitySums[probabilitySums.length - 1] = 1;
  return probabilitySums
    .reverse()
    .slice(1)
    .map((prob, i) => ({
      people: i + 1,
      probability: Math.round(prob * 100),
    }));
}
```

Дисперсний аналіз (ANOVA) – це статистичний метод, який використовується для визначення наявності статистично значущих відмінностей між середніми значеннями кількох груп. Існує два типи аналізу ANOVA.

Однофакторний ANOVA порівнює середні значення між кількома групами за одним фактором. Двофакторний ANOVA порівнює середні значення між групами за двома факторами, а також враховує можливу взаємодію між цими факторами [2].

Тож у моєму випадку цей метод може бути використаний саме у ситуаціях A/B тестувань, коли власнику товару потрібно визначити, чи існує статистично значуща різниця з погляду привабливості для покупців між товарами, які мають зовнішні відмінності. На відміну від попередніх двох методів, дисперсний аналіз у моїй програмі реалізовано за допомогою сторонньої JavaScript-бібліотеки "simple-statistics", що має вбудований метод для ANOVA.

4.3.5 Опис стратегії обробки помилок

У сучасних інформаційних системах надійність та стійкість до помилок є одними з найважливіших аспектів. Складність програмного забезпечення постійно зростає, і з нею збільшується ймовірність виникнення помилок. Навіть найретельніше спроектовані системи можуть зазнавати збоїв через непередбачувані фактори: збої в обладнанні, помилки користувачів, неочікувані вхідні дані або зовнішні атаки.

Одним із найефективніших методів боротьби з помилками, які можуть зупинити виконання програми, є конструкція try-catch. Всі операції, які можуть видати помилку при виконанні, аж до потенційного поділу на 0, необхідно заносити до блоку try, а в catch вказувати, що необхідно зробити у разі помилки для того, щоб після виконання цього блоку програма продовжила працювати без виходу з контексту виконання

Ще одним важливим концептом для обробки помилок у контексті середовища, в якому написано мою програму, а саме Node.js фреймворк під назвою Express, є концепт middleware. Він передбачає розбиття програми на компоненти, організовані у вигляді ланцюга обробників, через який проходить кожен запит. Кожен компонент може вирішити, чи передавати запит далі по цепі, чи обробляти його на місці. Це дозволяє на глобальному рівні кожного сервісу в

кінці всіх ланцюгів виконання програми розташувати універсальну функцію-обробник помилки, яка «відловить» усі викинуті програмою винятки, визначить, до якого саме типу належить помилка та прологує її.

```
1. export class AppError {
2.   message: string;
3.   statusCode: number;
4.   constructor(statusCode: number, message: string) {
5.     this.statusCode = statusCode;
6.     this.message = message;
7.   }
8.
9.   static badRequest(msg: string) {
10.     return new AppError(400, msg)
11.   }
12.
13.   static unauthorised(msg: string) {
14.     return new AppError(403, msg)
15.   }
16.
17.   static internal(msg: string) {
18.     return new AppError(500, msg)
19.   }
20. }
21.
22. export const ErrorHandler: express.ErrorRequestHandler =
async(err, req, res, next) => {
23.     console.error(err);
24.     if (err instanceof AppError){
25.         res.status(err.statusCode).json({error: err.message});
26.         return;
27.     }
28.     res.status(500).json({error: "Something went wrong"});
29. }
```

Аналогічна стратегія обробки винятків використовують у всіх сервісах системи.

4.4 Опис клієнтської частини програми

Front-end частина моєї програми написана за допомогою потужних технологій, на чолі яких стоїть фреймворк Next.js, створений на основі React. Головним перевагою Next у порівнянні з аналогічними JavaScript-фреймворками є його можливості серверного ретндерінгу та, відповідно, SEO оптимізації. Адже веб-сайт має бути не тільки зрозумілим і зовні привабливим для користувачів, а й оброблюваним внутрішніми алгоритмами пошукових систем. Після оновлення

Next до 14 версії минулого року всі компоненти цього фреймворку за замовчуванням стали серверними. Це дозволяє їм рендеруватися на сервери і приходити в браузер користувача вже в готовому вигляді, але позбавляє їх можливостей, доступних лише при виконанні JavaScript в браузері, наприклад, обробки кліків або in-memory кеша. На прикладі нижче наведено те, як легко і декларативно можна асинхронно зчитати масив користувачів з бази даних і відрендерити його в серверному компоненті:

```
1. const users = (await getUsers()) as IUser[];
2. return (
3. <>
4. <Headline color="yellow" classes="font-bold text-4xl mb-10 mt-16">
5. all profiles:
6. </Headline>
7. <div className="grid grid-cols-6 gap-3">
8. {users.map((user) => (
9.     <UserCard
10.     key={user._id}
11.     _id={user._id}
12.     login={user.login}
13.     avatarUrl={user.avatarUrl}
14.     role={user.role}
15.     />
16. ))}
17. </div>
18. </>
19. );
```

Тим не менш, якщо розробнику необхідно рендерити певний компонент саме в браузері клієнта, необхідно лише вказати директиву "use client" вгорі файлу компонента. Нижче наведено приклад мого кастомного компонента дропдауну, який може рендеритися з різними кольорами залежно від параметрів, що передаються. Цей компонент є клієнтським, тому здатний обробляти такі клієнтські події як onClick.

```
1. <div
2.   className={`w-full p-1 bg-gradient-to-r ${
3.   color === "blue"
4.     ? "from-blue-400 to-blue-600"
5.     : color === "red"
6.     ? "from-red-400 to-red-600"
7.     : "from-amber-400 to-amber-600"
8.   }`}
```

```
      9. >
      10.   <button
      11.   onClick={() => setIsSelect(!isSelect)}
      12.   className={`flex bg-zinc-900 w-full h-full items-center
justify-between p-2 ${
      13.     value === "" ? "text-zinc-400" : "text-zinc-50"
      14.   } font-medium`}
      15.   >
      16.   {value === "" ? placeholder || "Choose..." : value}
      17.   <FontAwesomeIcon icon={isSelect ? faChevronUp : faChevronDown}
/>
      18.   </button>
      19.
      20.   {isSelect &&
      21.   array?.map((item) => (
      22.     <button
      23.     key={item._id}
      24.     onClick={() => {
      25.       setValue(item.name);
      26.       setIsSelect(false);
      27.     }}
      28.     className="p-1 w-full mt-1 bg-transparent font-medium border-
4 border-zinc-900 rounded-xl text-zinc-900 hover:text-white hover:bg-zinc-
900 duration-300"
      29.     >
      30.     {item.name}
      31.     </button>
      32.   ))}
      33. </div>
```

Як помітно з коду вище, ще однією важливою технологією, що використовується в проекті, є CSS-фреймворк Tailwind, який забезпечує швидший та зручніший процес стилізації HTML за допомогою вбудованих класів.

Для представлення результатів виконання тестів, а також розподілів побудованих на їх основі необхідні графіки, і найвідомішим інструментом для цієї мети є бібліотека Chart.js. Я створив кастомний компонент для побудови графіків різних видів (наприклад, гістограма) залежно від даних, які передаються у форматі осей X і Y (див. додаток Є).

4.4.1 Реалізація інтернаціоналізації

Інтернаціоналізація програмного забезпечення означає його підготовку та налаштування для ефективної підтримки багатонаціональної аудиторії. У випадку мого веб-додатку першочерговою задачею є підтримка різних мов, адже багато аспектів системи можуть виявитися складними для сприйняття користувачів.

Так як на клієнтській частині системи використовується фреймворк Next.js, хорошим вфбором для інтернаціоналізації буде бібліотека "next-i18next". Її на відміну інших подібних бібліотек є підтримка інтернаціоналізації під час використання серверного рендерингу сторінок. Вона досягається завдяки тому, що поточна мова зберігається в URL-адресі, тобто всі ендпоінти обертаються в динамічний роут, що відповідає за мову, і саме завдяки ньому програма буде розуміти, якою мовою необхідно рендерувати поточну сторінку, а не завдяки механізму загального контексту компонентів, як це відбувається у випадках із виключно клієнтською стратегією рендерингу. Решта налаштування для здійснення перекладів інтерфейсу різними мовами здійснюється аналогічно решті бібліотек для інтернаціоналізації, а саме створюються JSON-файли перекладів, звідки і зчитується інформація.

Також для коректного та зручного форматування дати та часу залежно від часового поясу мною використовується бібліотека "date-fns". Вона надає величезний набір функцій перекладу часу, що звільняє розробників необхідність створювати свої утилітарні функції реалізації цього функціонала.

4.4.2 Опис вирішення проблеми дублюючого коду для типізації

Будуючи веб-додатки за допомогою архітектурного підходу REST API, будь-який розробник, що використовує мову TypeScript, стикається з проблемою типізації даних, яка повинна бути аналогічною на серверній та клієнтській стороні програми. Однак при передачі даних в JSON-форматі за допомогою протоколу HTTP типізація даних втрачається, і після отримання даних на front-end необхідно заново прописувати результату ті самі типи. Це не відповідає принципу проектування DRY (Don't Repeat Yourself).

Проблема, що утворилася, може бути вирішена шляхом винесення всіх типів та інших об'єктів програми, необхідних як на front-end так і на back-end, в окремий пакет, який у майбутньому може бути завантажений в мережу як спеціальна бібліотека. Такий підхід допоможе зовсім не писати типи в тих же репозиторіях, де знаходиться логіка, а винести його в окремий шар. У моїй

системі він називається "types-realtest" і крім зазначених вище технологій також використовує ідею реекспорту для того, щоб при імпорті типів в основному додатку не потрібно було посилатися на його будь-які внутрішні директорії.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, що допомагає забезпечити якість та надійність кінцевого продукту. Воно виявляє помилки та невідповідності на різних етапах розробки. У моєму випадку були обрані наступні два основні види тестування: модульне тестування, яке виконується програмно, та функціональне тестування, яке виконується вручну.

5.1 Модульне тестування

Модульне (Unit) тестування є процесом перевірки окремих модулів або компонентів коду. Його мета – переконатися, що кожен окремий модуль функціонує правильно ізольовано від інших частин системи. Модульне тестування виконується програмно за допомогою спеціалізованих інструментів та фреймворків.

Основна перевага модульного тестування полягає в можливості виявляти помилки на ранніх етапах розробки, що значно знижує вартість їх виправлення. Тестування ізольованих модулів дозволяє виявити проблеми в конкретних частинах коду без впливу на інші модулі, що є дуже доречним у випадку мікросервісного підходу моєї системи. Програмне модульне тестування також легко автоматизувати, що дозволяє швидко виконувати тести при кожній зміні коду. Крім того, написання тестів сприяє створенню більш структурованого та зрозумілого коду, що покращує загальний дизайн програмного забезпечення.

Для модульного тестування кожного сервісу back-end частини програми, написаної на JavaScript-платформі Node.js зручно використовувати бібліотеку Jest. Оскільки тестуватися будуть саме контролери, що отримують запити в форматі об'єкту Request і надсилають відповіті в форматі Response корисно буде також використати бібліотеку Supertest, що допомагає саме з обробкою HTTP запитів на рівні тестування:

```
1. describe("Test user service", () => {
```

```
2.   let s: any;
3.
4.   beforeAll(async () => {
5.     s = await server;
6.   });
7.
8.   afterAll(async () => {
9.     if (s) {
10.       s.close();
11.     }
12.   });
13.
14.   test("Get all users", async () => {
15.     const res = await request(s).get("/");
16.     expect(res.status).toBe(200);
17.     expect(Array.isArray(res.body)).toBe(true);
18.     res.body.forEach((obj: any) => {
19.       expect(obj).toHaveProperty("_id");
20.       expect(obj).toHaveProperty("name");
21.       expect(obj).toHaveProperty("email");
22.       expect(obj).toHaveProperty("login");
23.       expect(obj).toHaveProperty("role");
24.     });
25.   });
26. }
```

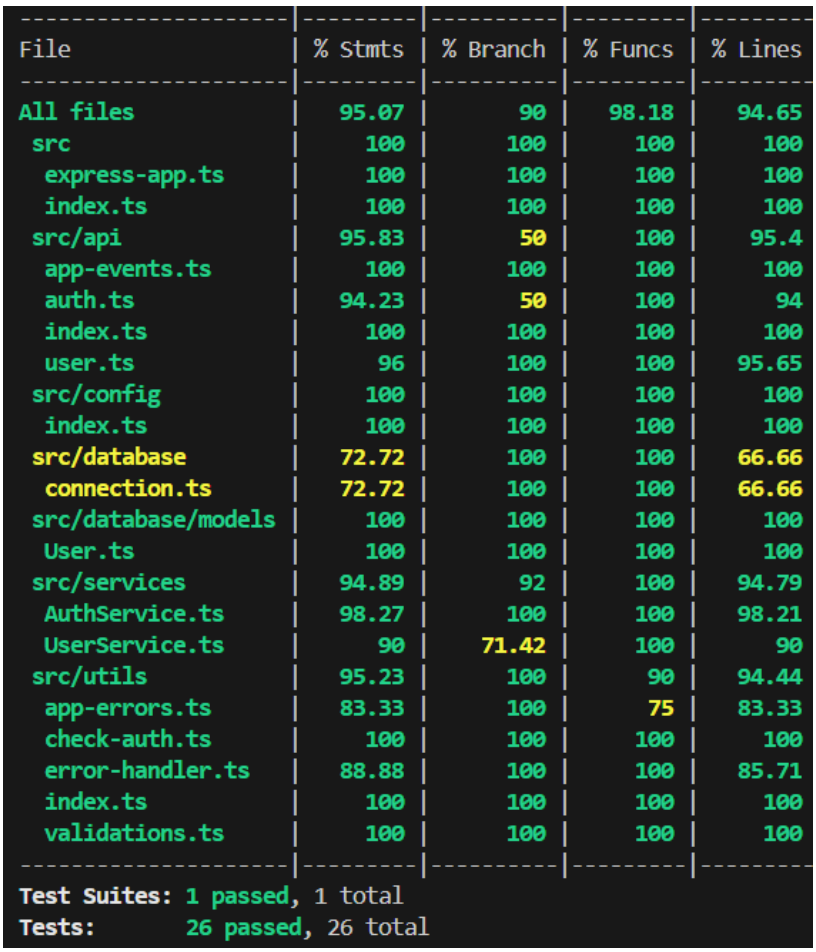
Результати тестування для сервісу користувача наведені нижче у вигляді таблиці 5.1 та у вигляді статистики покриття коду тестами (див. рис. 5.1).

Таблиця 5.1 – Результати тестування сервісу користувача

№	Тест	Результат	Висновок
1	Отримання всіх користувачів	Статус: 200 Дані: масив об’єктів User	Пройдено
2	Отримання одного користувача по правильному id	Статус: 200 Дані: об’єкт User	Пройдено
3	Отримання одного користувача по неправильному id	Статус: 400 Дані: {"error": "User doesn't exist" }	Пройдено
4	Оновити роль правильного профілю	Статус: 200 Дані: об’єкт з новою роллю	Пройдено
5	Оновити роль неправильного профілю	Статус: 400 Дані: {"error": "User doesn't exist" }	Пройдено
6	Отримати користувача за токеном	Статус: 200 Дані: об’єкт User	Пройдено

Кінець таблиці 5.1

№	Тест	Результат	Висновок
7	Отримати свій профіль	Статус: 200 Дані: об’єкт User	Пройдено
8	Отримати свій профіль без авторизації	Статус: 403 Дані: {"error": "You are not logged in" }	Пройдено
9	Авторизація з неправильним логіном	Статус: 400 Дані: {"error": "Wrong login, email or password" }	Пройдено
10	Авторизація з неправильним паролем	Статус: 400 Дані: {"error": "Wrong login, email or password" }	Пройдено
11	Коректна авторизація	Статус: 200 Дані: об’єкт User	Пройдено
12	Зміна пароля без авторизації	Статус: 403 Дані: {"error": "You are not logged in" }	Пройдено
13	Зміна пароля з неправильним старим	Статус: 400 Дані: {"error": " Wrong old password" }	Пройдено
14	Зміна пароля з неспівпадаючим новим	Статус: 400 Дані: {"error": " Passwords do not match" }	Пройдено
15	Коректна зміна пароля	Статус: 200 Дані: об’єкт User	Пройдено
16	Вихід закаунту	Статус: 200 Дані: {"success": "true" }	Пройдено
17	Вихід закаунту без авторизації	Статус: 403 Дані: {"error": "You are not logged in" }	Пройдено
18	Реєстрація з некоректною електронною поштою	Статус: 400 Дані: {"error": "Invalid email" }	Пройдено
19	Реєстрація з пропущеними полями	Статус: 400 Дані: {"error": " All fields should be filled" }	Пройдено
20	Коректна реєстрація	Статус: 200 Дані: об’єкт User	Пройдено



File	% Stmts	% Branch	% Funcs	% Lines
All files	95.07	90	98.18	94.65
src	100	100	100	100
express-app.ts	100	100	100	100
index.ts	100	100	100	100
src/api	95.83	50	100	95.4
app-events.ts	100	100	100	100
auth.ts	94.23	50	100	94
index.ts	100	100	100	100
user.ts	96	100	100	95.65
src/config	100	100	100	100
index.ts	100	100	100	100
src/database	72.72	100	100	66.66
connection.ts	72.72	100	100	66.66
src/database/models	100	100	100	100
User.ts	100	100	100	100
src/services	94.89	92	100	94.79
AuthService.ts	98.27	100	100	98.21
UserService.ts	90	71.42	100	90
src/utils	95.23	100	90	94.44
app-errors.ts	83.33	100	75	83.33
check-auth.ts	100	100	100	100
error-handler.ts	88.88	100	100	85.71
index.ts	100	100	100	100
validations.ts	100	100	100	100

Test Suites: 1 passed, 1 total
Tests: 26 passed, 26 total

Рисунок 5.1 – Статистика покриття сервісу тестами

Результати тестування сервісу компанії наведено в таблиці 5.2. Результати тестування сервісу тестів наведено в таблиці 5.3.

Таблиця 5.2 – Результати тестування сервісу компанії

№	Тест	Результат	Висновок
1	Створення компанії з закоротким описом	Статус: 400 Дані: {"error": "Company description should contain at least 10 symbols, maximum 1000 symbols" }	Пройдено
2	Створення компанії без посилання на зображення	Статус: 400 Дані: {"error": "All fields should be filled!"}	Пройдено
3	Створення компанії з дублюючою назвою	Статус: 400 Дані: {"error": " Company with the name already	Пройдено

Кінець таблиці 5.2

№	Тест	Результат	Висновок
		exists"}	
4	Коректне створення компанії	Статус: 200 Дані: об’єкт Company	Пройдено
5	Видалення неіснуючої компанії	Статус: 400 Дані: {"error": "You are deleting unexistent company"}	Пройдено
6	Видалення компанії користувачем що не є власником	Статус: 403 Дані: {"error": "You are not an owner"}	Пройдено
7	Коректне видалення компанії	Статус: 200 Дані: {"success": "true" }	Пройдено
8	Отримання компанії за неправильним id	Статус: 400 Дані: {"error": "The company doesn't exist"}	Пройдено
9	Отримання компанії за правильним id	Статус: 200 Дані: об’єкт Company	Пройдено
10	Отримання всіх компаній	Статус: 200 Дані: масив об’єктів Company	Пройдено
11	Оцінка компанії вище 5 або нижче 1	Статус: 400 Дані: {"error": "Wrong rating input"}	Пройдено
12	Дублююча оцінка компанії	Статус: 400 Дані: {"error": "You have already rated this"}	Пройдено
13	Коректна оцінка компанії	Статус: 200 Дані: оновлений об’єкт Company	Пройдено
14	Купівля девайсу користувачем, що не є власником	Статус: 400 Дані: {"error": " You don't work in the company"}	Пройдено
15	Купівля девайсу при недостатньому балансі	Статус: 400 Дані: {"error": "Your doesn't have enogh money on balance"}	Пройдено
16	Коректна купівля девайсу	Статус: 200 Дані: об’єкт Purchase	Пройдено
17	Поповнення балансу компанії	Статус: 200 Дані: {"success": "true" }	Пройдено

Таблиця 5.3 – Результати тестування сервісу тестів

№	Тест	Результат	Висновок
1	Створення тесту без авторизації адміністратора компанії	Статус: 403 Дані: {"error": "You are not an owner"}	Пройдено
2	Коректне створення тесту	Статус: 200 Дані: об’єкт Test	Пройдено
3	Видалення тесту при відсутності підтримки сервісу компанії	Статус: 500 Дані: {"error": "Unable to delete test"}	Пройдено
4	Видалення тесту з неіснуючим id	Статус: 400 Дані: {"error": 'Wrong id format!'}	Пройдено
5	Коректне видалення тесту	Статус: 200 Дані: {"success": "true" }	Пройдено
6	Отримати тест по id без авторизації адміністратора компанії	Статус: 403 Дані: {"error": "You are not an owner"}	Пройдено
7	Отримати тест по id	Статус: 200 Дані: об’єкт Test	Пройдено
8	Отримати всі тести товару з неправильним id	Статус: 400 Дані: {"error": 'Wrong id format!'}	Пройдено
9	Отримати всі тести товару з правильним id	Статус: 200 Дані: масив об’єктів Test	Пройдено
10	Зміна налаштувань неіснуючого тесту	Статус: 400 Дані: {"error": 'The test you are trying to edit does nor exist anymore!' }	Пройдено
11	Зміна налаштувань коректного тесту	Статус: 200 Дані: оновлений об’єкт Test	Пройдено
12	Запуск тестування з неправильним id	Статус: 400 Дані: {"error": "This test was not created"}	Пройдено
13	Запуск тестування з датою, меншою за дату початку	Статус: 400 Дані: {"error": "You must set the end time of your test properly"}	Пройдено
14	Запуск тестування не адміністратором компанії	Статус: 403 Дані: {"error": "You are not an owner"}	Пройдено
15	Коректний запуск	Статус: 200	Пройдено

Кінець таблиці 5.3

№	Тест	Результат	Висновок
	тестування	Дані: {"success": "true" }	
16	Отримання результатів не адміністратором компанії	Статус: 403 Дані: {"error": "You are not an owner"}	Пройдено
17	Отримання результатів адміністратором компанії	Статус: 200 Дані: масив Result	Пройдено
18	Отримання біноміального розподілу, розподілу Пуассона та дисперсного аналізу за результатами	Статус: 200 Дані: об’єкт з масивами даних для осей графіка	Пройдено

5.2 Функціональне тестування

Функціональне тестування це процес перевірки функціональності системи в цілому, щоб переконатися, що вона працює відповідно до вимог та специфікацій. Функціональне тестування виконується вручну, дозволяючи перевірити, як система взаємодіє з користувачем та чи відповідає вона його очікуванням. Основним завданням функціонального тестування є забезпечення відповідності функціональності системи заявленим вимогам та специфікаціям.

Мануальне тестування дозволяє оцінити зручність та коректність роботи користувацького інтерфейсу, а також виконувати сценарії, які відображають реальні дії користувачів. Це дозволяє виявити можливі проблеми у використанні системи, які можуть бути пропущені автоматизованими тестами. Ручне тестування також забезпечує гнучкість, дозволяючи швидко адаптуватися до змін у вимогах або інтерфейсі системи.

У даному проєкті функціональне тестування буде проводитися саме на клієнтській частині. Це рішення обґрунтоване кількома важливими факторами. По-перше, більшість інтерактивних дій та взаємодій з користувачем відбувається саме на клієнті, де важливо переконатися, що користувацький інтерфейс працює належним чином. Ручне тестування дозволяє тестувальникам оцінити зручність користування, інтуїтивність інтерфейсу та виявити можливі проблеми, які можуть

залишитися непоміченими автоматизованими тестами. По-друге, серверна частина системи вже покритий модульними тестами, що забезпечує високу надійність та правильність роботи його окремих компонентів. Завдяки програмному модульному тестуванню, back-end був ретельно перевірений на предмет помилок та невідповідностей на рівні окремих модулів. Це дозволяє впевнено стверджувати, що логіка та функціональність серверної частини працює коректно.

Трьома основними тест-кейсами, що необхідно перевірити є поповнення балансу компанії користувачем в системі (таблиця 5.4), створення та запуск тестів (таблиця 5.5) та аналіз результатів (таблиця 5.6). Саме ці події будуть найбільш затребуваними серед користувачів і є потенційно найуразливішими на клієнтській частині програми.

Таблиця 5.4 – Результати тестування поповнення балансу компанії

№	Опис випадку	Очікуваний результат	Висновок
1	Користувач, що не авторизований в системі або не є власником, відкриває сторінку компанії	З’являється лише загальна інформація про компанію	Пройдено
2	Власник компанії авторизований в системі	З’являється додатковий функціонал для редагування інформації про компанію	Пройдено
3	Користувач натискає на «Поповнення» та вводе необхідну суму	Перенаправлення на сторінку платіжної системи Stripe	Пройдено
4	Користувач повертається на сторінку компанії	Баланс компанії в системі не змінюється	Пройдено
5	На сторінці платіжної системи користувач вводе коректні дані про банківську картку та натискає «Підтвердити»	Перенаправлення на сторінку компанії з оновленим балансом	Пройдено
6	Користувач вводе некоректні дані про банківську картку та	Отримання помилки на сторінці платіжної системи	Пройдено

Кінець таблиці 5.4

№	Опис випадку	Очікуваний результат	Висновок
	натискає «Підтвердити»		
7	Користувач, що не авторизований в системі або не є власником, відкриває сторінку компанії	З’являється лише загальна інформація про компанію	Пройдено
8	Власник компанії авторизований в системі	З’являється додатковий функціонал для редагування інформації про компанію	Пройдено

Таблиця 5.5 – Результати тестування створення та запуску тестів

№	Опис випадку	Очікуваний результат	Висновок
1	Натискання на «Додати тест» на сторінці компанії	Відкриття форми створення тесту	Пройдено
2	Відсутність вибору вільних пристроїв або товарів для тестування	Кнопки для створення не з’являється	Пройдено
3	Натискання на «Новий тест» на сторінці переліку тестів, прив'язаних до девайсу	Відкриття форми створення тесту з вже вибраним девайсом	Пройдено
4	Введення назви тесту	Поява кнопки для створення	Пройдено
5	Підтвердження створення тесту	Перенаправлення на сторінку тесту	Пройдено
6	Натискання на кнопку «Запуск тесту»	Сплив діалогового вікна для останнього налаштування тестування	Пройдено
7	Вибір дати закінчення тестування	Активація кнопки підтвердження	Пройдено
8	Вибір тесту для пари A/B тестування у разі знаходження заздалегідь створеного тесту для того ж продукту	Прив'язка другого тесту в комірці «Пара»	Пройдено
9	Натискання на кнопку «Підтвердити»	Оновлення сторінки з появою дат старту та закінчення тесту	Пройдено

Таблиця 5.6 – Результати тестування процесу аналізу тестів

№	Опис випадку	Очікуваний результат	Висновок
1	Відкриття сторінки активного тесту	На сторінці тесту з’являються розділ результатів та чотири гістограми	Пройдено
2	У відповідності до частоти звітності, вказаної в налаштуваннях тесту, оновлення сторінки тесту	З’являється новий елемент у розділі результатів і новий стовпчик на кожній з гістограм, нові значення відповідають одне одному	Пройдено
3	Натискання на «Детальні результати»	Перенаправлення на сторінку детального аналізу з описом математичних методів	Пройдено
4	Вибір біномного розподілу	Побудова гістограми ймовірностей кількості підходів до товару для 100 клієнтів за замовчуванням	Пройдено
5	Зміна кількості клієнтів в моделі на 50	Перебудова гістограми	Пройдено
6	Вибір розподілу Пуассона	Побудова гістограми ймовірностей кількості підходів для аналогічного періоду часу за замовченням	Пройдено
7	Зміна часового інтервалу для моделювання	Перебудова гістограми	Пройдено
8	Тест позначено як парний	На сторінці детального аналізу також повинен знаходитися ANOVA аналіз	Пройдено
9	Вибір ANOVA аналізу	Відображення нульової гіпотези, висновку, порівняльних графіків	Пройдено

5.3 Навантажувальне тестування

Навантажувальне тестування є невід'ємною частиною процесу забезпечення якості програмного забезпечення. Воно спрямоване на визначення поведінки

системи під час роботи в умовах інтенсивного використання. Основною метою цього виду тестування є виявлення можливих проблем із продуктивністю, стійкістю та масштабованістю додатків, коли вони обробляють великі обсяги даних або одночасні запити.

Зручним інструментом для навантажувального тестування є програма Apache JMeter з відкритим кодом на Java. Він дозволяє імітувати різні сценарії користувацької активності, щоб оцінити продуктивність та стійкість системи під час роботи з великими обсягами запитів або даних.

Налаштувавши JMeter за допомогою необхідних ендпоінтів, я змоделював ситуацію використання програми великою кількістю користувачів (потоків) одночасно, які надсилають по 1800 запитів на 12 ключових ендпоінтів, мною були отримані наступні результати, розбиті на три категорії (див. рис. 5.2).

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**345785**	**14**	**0.00%**	**227.89**	**0**	**24129**	**97.00**	**403.00**	**466.00**	**1424.98**	**75.77**	**1683.22**	**1261.93**
get all users	1800	3	0.17%	492.14	125	12672	262.00	763.70	1403.90	6135.16	0.52	152.51	10.19
get one user	1800	0	0.00%	76.51	15	5080	28.00	76.00	182.95	1106.15	0.52	26.22	0.20
sign-up	1800	1	0.06%	62.96	2	5017	17.50	85.00	177.50	809.00	0.52	1.54	0.23
sign-in	1800	0	0.00%	100.73	2	5185	59.00	147.90	250.00	956.84	0.52	0.32	0.22
create company	1800	1	0.06%	110.64	3	5339	61.00	153.90	262.95	1056.91	0.52	0.47	0.24
create company product	1800	0	0.00%	112.96	3	5211	62.00	159.90	262.95	1037.55	0.52	0.33	0.23
rate company product	1800	0	0.00%	113.73	2	4848	62.00	158.00	262.90	1073.94	0.52	0.35	0.24
top up company balance	1800	0	0.00%	113.66	4	5282	61.50	154.90	262.00	1067.47	0.52	0.24	0.24
create test	1800	0	0.00%	96.00	15	5269	49.00	153.00	270.95	977.40	0.52	0.31	0.24
start test	1800	1	0.06%	102.93	2	5695	56.00	158.90	285.95	934.25	0.52	0.34	0.24
buy smart device	1800	0	0.00%	103.13	3	5723	54.00	168.00	277.35	968.89	0.52	0.26	0.24
top up company balance	1800	0	0.00%	97.88	2	5602	52.50	162.90	280.75	842.82	0.52	0.55	0.24
get test results	1800	0	0.00%	95.62	2	5608	51.00	159.00	278.00	825.00	0.52	0.30	0.24
get test distribution	1800	0	0.00%	89.82	2	4875	47.00	149.90	276.55	818.96	0.52	0.34	0.23

Рисунок 5.2 – Результати навантажувального тестування

Перша категорія демонструє кількість виконань: загальна кількість виконаних зразків, загальна кількість зразків, які завершилися помилкою та відсоток помилок.

Друга категорія відповідна за час відповіді в мілісекундах: середній витрачений час, мінімальний витрачений час, максимальний витрачений час,

медіанний витрачений час, 90-й перцентиль, 95-й перцентиль та 99-й перцентиль часу.

Останній розділ демонструє пропускну здатність в КБ/сек.: кількість отриманих та надісланих кілобайт на секунду.

З отриманих результатів ми можемо зробити висновок, що за невеликих темпів запитів система здатна обробляти їх із мінімальною кількістю помилок. Це свідчить про високу стабільність і надійність системи, а також про те, що вона добре масштабована і здатна витримувати значне навантаження без суттєвого погіршення продуктивності. Частково це пов'язано з рішенням вибрати мікросервісну архітектуру на back-end, що зробило різні частини програми максимально ізольованими один від одного аж до можливості розміщувати їх на різних серверах.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є програмна система, що забезпечує повний цикл організації та обробки A/B тестів для фізичних товарів компаній.

Як з'ясувалося, досі механізми A/B тестування та інших методів більш детального аналізу привабливості товарів для покупців, які існують у середовищі онлайн-продажів, досі не переносилися в умови реального світу, незважаючи на те, що вони здатні невеликі кошти надати найціннішу інформацію про профіль клієнтів. Моя програмна система є спробою вирішити цю проблему.

Чимало уваги було приділено проектуванню та архітектурі програмного забезпечення. В результаті цього етапу були побудовані наступні діаграми UML: діаграма прецедентів, діаграма пакетів, діаграма розгортання, ER-діаграма та діаграма станів.

Програма складається з трьох основних компонентів: серверної частини з використанням платформи Node.js у зв'язці з легковажним фреймворком, розумного пристрою на базі платформи Arduino та клієнтського веб-сайту, написаного за допомогою фреймворку Next.js. Крім ключових технологій, описаних вище, мною також були використані фреймворк Tailwind CSS для стилізації за допомогою класів на front-end частини, бібліотека Chart.js для швидкого та зручного представлення гістограм, ODM Mongoose для опірацій над базами даних, Jest для тестування та деякі інші JavaScript-залежності. Основною мовою розробки був TypeScript, який є обгорткою довкола JavaScript, що надає статичну типізацію. Будь-яка частина програми відповідає загальноприйнятим стандартам чистого коду завдяки патернам "банди чотирьох" та іншим корисним практикам, як, наприклад, винесення частин коду в загальний пакет для уникнення дублювання коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Лаврищева К. М. Програмна інженерія [Електронний ресурс] / Катерина Михаліївна Лаврищева // Київ. – 2008. – URL: https://web.archive.org/web/20120628103421/http://www.programsfactory.univ.kiev.ua/content/books/2 (дата звернення: 02.04.2024).

2. Суворов Д., Олійник Е. FEATURES OF ANOVA// SCIENCE, RESEARCH, DEVELOPMENT. 2019. № 24. С. 69-72.

3. Gui H. et al. Network a/b testing: From sampling to estimation //Proceedings of the 24th International Conference on World Wide Web. – 2015. – С. 399-409.

4. Refactoring Guru - Патерни проектування. URL: https://refactoring.guru/uk (дата звернення: 14.04.2023).

5. J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec. Can cascades be predicted? In Proceedings of the 23rd international conference on World wide web, pages 925-936. International World Wide Web Conferences Steering Committee, 2014.

6. Федорченко А. В. А/В-тестування як ефективний інструмент цифрового маркетингу / А. В. Федорченко, І. В. Пономаренко // Проблеми інноваційно-інвестиційного розвитку. – 2019. – № 19. – С. 36-42.

7. Gilotte A. et al. Offline a/b testing for recommender systems //Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. – 2018. – С. 198-206.

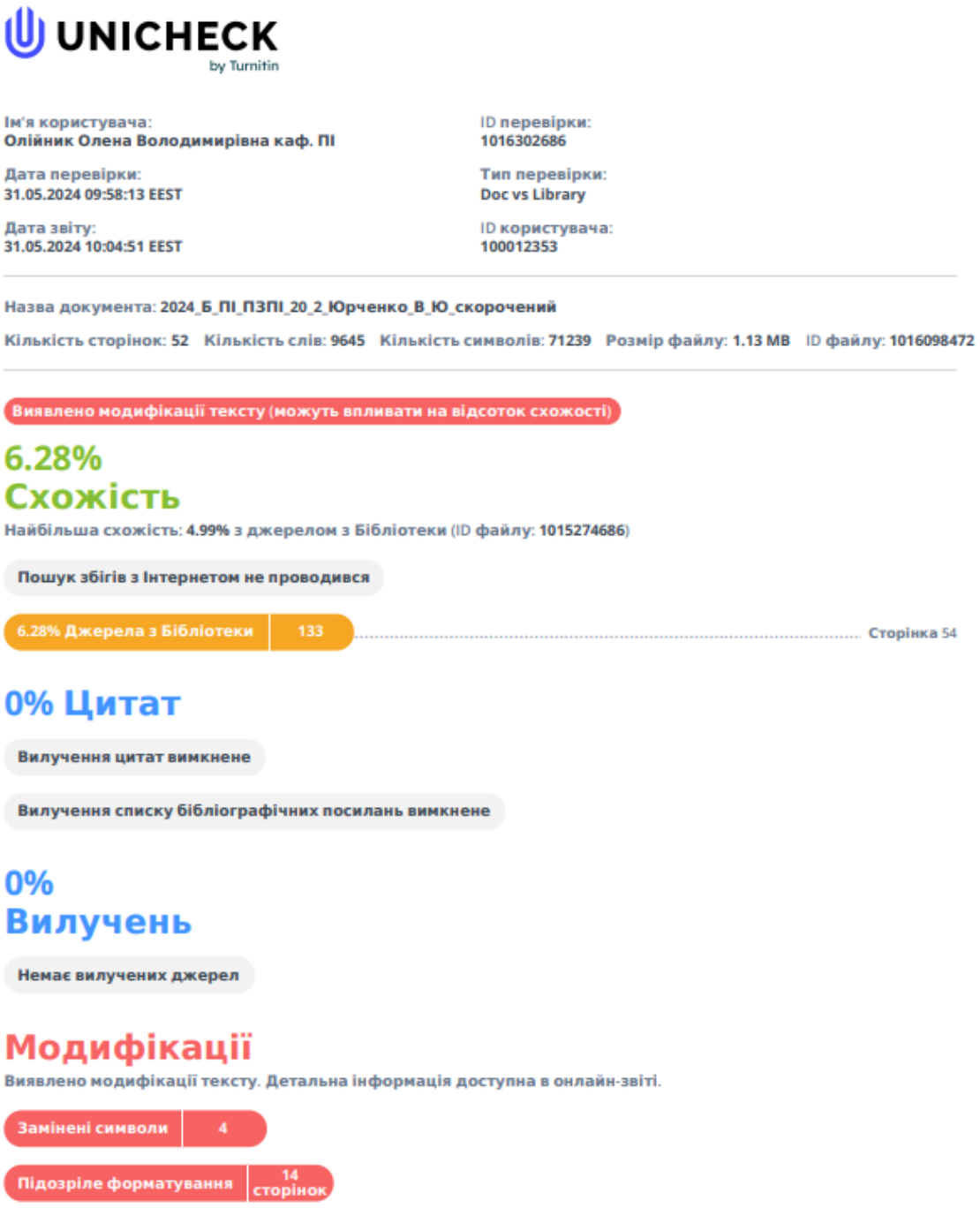
8. Yang, B., 2014. Research on Customer Value Evaluation Based on Data Mining and Mathematical Statistics. AMR 926–930, 3890–3893.

9. R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Pearson Education, 2009.

10. Брацький, В. О., and О. М. М’якшило. "Дослідження особливостей застосування реляційних і нереляційних баз даних на прикладі SQL Server та MongoDB." Наукові праці Національного університету харчових технологій 22,№ 5 (2016): 15-24.

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016302686

Дата перевірки:
31.05.2024 09:58:13 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
31.05.2024 10:04:51 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ_20_2_Юрченко_В_Ю_скорочений

Кількість сторінок: 52 Кількість слів: 9645 Кількість символів: 71239 Розмір файлу: 1.13 MB ID файлу: 1016098472

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.28%
Схожість

Найбільша схожість: 4.99% з джерелом з Бібліотеки (ID файлу: 1015274686)

Пошук збігів з Інтернетом не проводився

6.28% Джерела з Бібліотеки 133 ..... Сторінка 54

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 4

Підозріле форматування 14 сторінок

ДОДАТОК Б

REST API специфікація

Таблиця А.1 – Перелік ендпоінтів серверного застосунку.

Посилання на endpoint	**Метод**	**Політика доступу**	**Тіло запиту**	**Вихідні дані**
/upload	POST	user	Файл jpg/png	Посилання на файл
/user/register	POST	user	Ім’я, логін, пароль, email	Інформація про нового користувача
/user/login	POST	user	Логін/email та пароль	Інформація про користувача
/user/logout	POST	user	-	Повідомлення про успіх чи невдачу
/user/changePassword	PUT	user	Старий пароль, новий пароль	Повідомлення про успіх чи невдачу
/user/:id	GET	user	-	Інформація про користувача
/user	GET	user	-	Інформація про всіх користувачів
/company/create	POST	user	Назва, аватар	Інформація про нову компанію
/company/:id	GET	user	-	Інформація про компанію
/company/:id	DELETE	company admin	-	Повідомлення про успіх чи невдачу
/company/device/create	POST	admin	Назва, опис, зображення, ціна	Інформація про новий девайс
/purchase/create	POST	company	Id компанії,	Інформація про

Продовження таблиці А.1

Посилання на endpoint	**Метод**	**Політика доступу**	**Тіло запиту**	**Вихідні дані**
		admin	id девайсу	успішне придбання
/company/purchase/:id	PUT	company admin	Нова частота чи дальність	Повідомлення про успіх чи невдачу
/company/purchase/:id	GET	company admin	-	Інформація про девайс
/test/create	POST	company admin	Назва, id девайсу, id продукту	Інформація про новий тест
/test/:id	GET	company admin	-	Інформація про тест
/test/:id	DELETE	company admin	-	Повідомлення про успіх чи невдачу
/test/getByProduct/:id	GET	company admin	-	Інформація про всі тести
/test/:id	PUT	company admin	Нова частота чи дальність	Повідомлення про успіх чи невдачу
/test/product/create	POST	company admin	Компанія, назва, ціна та зображення	Інформація про новий продукт
/test/product/:id	GET	user	-	Інформація про продукт
/test/products/:id	GET	user	-	Інформація про продукти
/test/product/rate/:id	POST	user	Оцінка	Повідомлення про успіх чи невдачу
/test/product/rate/:id	DELETE	user	-	Повідомлення про

Кінець таблиці А.1

Посилання на endpoint	**Метод**	**Політика доступу**	**Тіло запиту**	**Вихідні дані**
				успіх чи невдачу
/test/:id/result	POST	company admin	Кінцевий час тестування	Повідомлення про успіх чи невдачу початку тестування
/test/:id/result	GET	company admin	-	Інформація про результати

ДОДАТОК В

Тези доповіді на конференції «Радіоелектроніка та молодь у XXI столітті»

Рисунок Б.1 – Обкладинка конференції «Радіоелектроніка та молодь у XXI столітті»

ДОДАТОК Г

Код функції реєстрації користувача

```
1. async RegisterUser(data: {
2.   email: string;
3.   login: string;
4.   password: string;
5.   name: string;
6.  }) {
7.   try {
8.    const { email, password, login, name } = data;
9.    if (!email || !password || !name || !login) {
10.     throw AppError.badRequest("All fields should be filled");
11.    }
12.    const existingEmail = await UserModel.findOne({ email });
13.    const existingLogin = await UserModel.findOne({ login });
14.    if (existingEmail || existingLogin) {
15.     throw AppError.badRequest("User with this email/login alredy
registered");
16.    }
17.    const salt = random() as string;
18.
19.    const user = new UserModel({
20.     email,
21.     login,
22.     name,
23.     authentication: {
24.      salt,
25.      password: authenticationToken(salt, password),
26.     },
27.    });
28.    await user.save();
29.    return {
30.     email: user.email,
31.     login: user.login,
32.     _id: user._id,
33.     role: user.role,
34.    };
35.   } catch (err) {
36.    throw err;
37.   }
38.  }
```

ДОДАТОК Д

Код обробки платежу на сервері за допомогою Stripe

```
1. const stripe = new Stripe(process.env.STRIPE_SECRET_KEY, {
2.   apiVersion: "2022-11-15",
3. });
4.
5. async CreatePaymentSession(data: {
6.     returnUrl: string;
7.     amount: number;
8.     companyId: string;
9.   }) {
10.     try {
11.       const session = await stripe.checkout.sessions.create({
12.         payment_method_types: ["card"],
13.         line_items: [
14.           {
15.             price_data: {
16.               currency: "usd",
17.               product_data: {
18.                 name: "Top up balance",
19.               },
20.               unit_amount: data.amount * 100,
21.             },
22.             quantity: 1,
23.           },
24.         ],
25.         mode: "payment",
26.         success_url: `${data.returnUrl}/company/${data.companyId}`,
27.         cancel_url: `${data.returnUrl}/company/${data.companyId}`,
28.       });
29.       return { id: session.id };
30.     } catch (err) {
31.       throw err;
32.     }
33.   }
```

ДОДАТОК Е

Код фільтрації результатів тестування

```
1. async GetAllResults(testId: string) {
2.   try {
3.     let results: Result[] = await ResultModel.find({ testId });
4.     let allClients = 0;
5.     if (!results.length) return { results, allClients };
6.     const defaultDistance = results[0].approaches[0].distance;
7.     let curApproachDistance = 0;
8.     let curApprocahTime = 0;
9.     for (let index in results) {
10.       let apps: Approach[] = [];
11.       const { approaches } = results[index];
12.       for (let i = 1; i < approaches.length; i++) {
13.         if (this.isApproach(approaches[i].distance, defaultDistance)) {
14.           if (this.isApproach(approaches[i - 1].distance,
defaultDistance)) {
15.             curApproachDistance = Math.min(
16.               curApproachDistance,
17.               approaches[i].distance
18.             );
19.             curApprocahTime += this.countTimeDifference(
20.               approaches[i - 1].time,
21.               approaches[i].time
22.             );
23.           } else {
24.             curApproachDistance = approaches[i].distance;
25.           }
26.         } else {
27.           if (curApproachDistance === 0) continue;
28.           apps.push({
29.             distance: curApproachDistance,
30.             time:
31.               curApprocahTime +
32.               this.countTimeDifference(
33.                 approaches[i - 1].time,
34.                 approaches[i].time
35.               ),
36.           });
37.           curApproachDistance = 0;
38.           curApprocahTime = 0;
39.         }
40.       }
41.       results[index].approaches = [...apps];
42.       allClients += apps.length;
43.     }
44.     return { results, allClients };
45.   } catch (err) {
46.     throw err;
47.   }
48. }
```

ДОДАТОК Ж

Код для побудови графіків на сайті

```
1. const CustomChart: FC<ApproachChartProps> = ({
2.   data,
3.   id,
4.   bgColor,
5.   labels,
6.   type,
7.   xName,
8.   yName,
9. }) => {
10.   useEffect(() => {
11.     const xLabels = labels || data.map((i, index) => index + 1);
12.     if (data.length > 1) {
13.       const canvas = document.getElementById(id) as
HTMLCanvasElement;
14.       var ctx = canvas.getContext("2d");
15.       const existingChart = Chart.getChart(ctx!);
16.       if (existingChart) {
17.         existingChart.destroy();
18.       }
19.       new Chart(ctx!, {
20.         type: type || "bar",
21.         data: {
22.           labels: xLabels,
23.           datasets: [
24.             {
25.               data: data.map((value, index) => ({
26.                 x: xLabels[index],
27.                 y: value,
28.                 r: (value * 20) / xLabels[index],
29.               })),
30.               backgroundColor: bgColor || "#14b8a6",
31.             },
32.           ],
33.         },
34.         options: {
35.           scales: {
36.             x: {
37.               title: {
38.                 display: true,
39.                 text: xName,
40.               },
41.             },
42.             y: {
43.               title: {
44.                 display: true,
45.                 text: yName,
46.               },
47.             },
48.           },
49.           plugins: {
50.             tooltip: {
51.               callbacks: {
```

```
52.                   label: function (context) {
53.                     const labelX = context.dataset.label || xName;
54.                     const labelY = context.dataset.label || yName;
55.                     const yData =
context.dataset.data[context.dataIndex] as {
56.                       y: number;
57.                     };
58.                     return `${labelX}: ${
59.                       xLabels[context.dataIndex]
60.                     }, ${labelY}: ${yData.y.toFixed(1)}`;
61.                   },
62.                 },
63.               },
64.               legend: {
65.                 display: false,
66.               },
67.             },
68.           },
69.         });
70.       }
71.     }, []);
72.
73.     if (data.length === 1) {
74.       return (
75.         <h1 className="flex items-center justify-center font-bold
text-2xl text-slate-200">
76.           No available data
77.         </h1>
78.       );
79.     }
80.
81.     return <canvas id={id}></canvas>;
82. };
```

ДОДАТОК К

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кваліфікаційна робота бакалавра

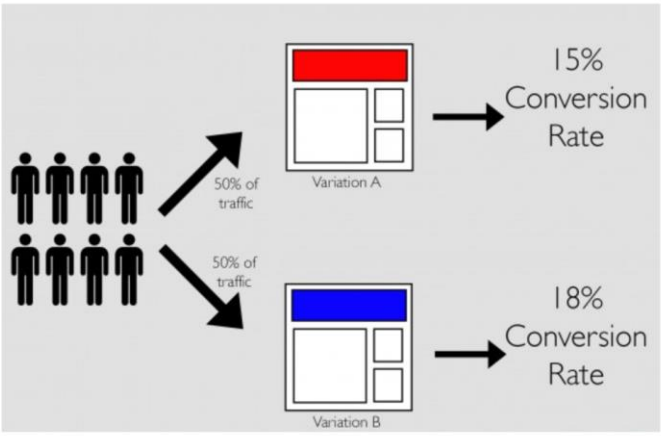
Програмна система для A/B тестування товарів компанії

ВИКОНАВ: СТ. ГР. ПЗПІ-20-2 ЮРЧЕНКО В.Ю.

КЕРІВНИК: СТ. ВИКЛ. ОЛІЙНИК О.В.

1

Аналіз проблеми



2

Аналіз існуючих рішень

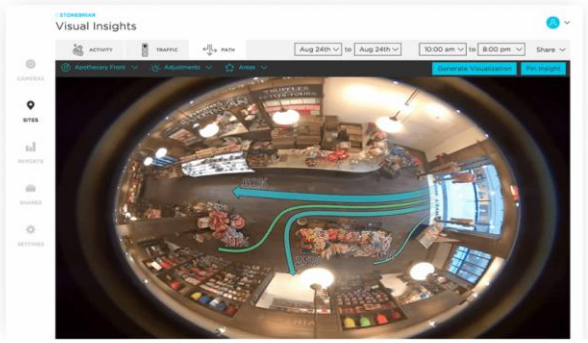
3

Аналіз конкурентів

	Офлайн-тести (5)	Діапазон магазинів (4)	Аналіз даних (4)	Продуктивність (4)	Дизайн (3)	Онлайн-тести (2)	Загалом
Ignite Prism	5	3	3	4	4	3	**83**
Dynamic Yield	3	2	5	4	3	5	**78**
Google Analytics	2	2	5	5	4	4	**78**
Leanplum	2	2	3	3	5	4	**65**
Optimizely	5	3	4	3	3	4	**82**
Моя система	5	5	5	4	4	2	**97**

4

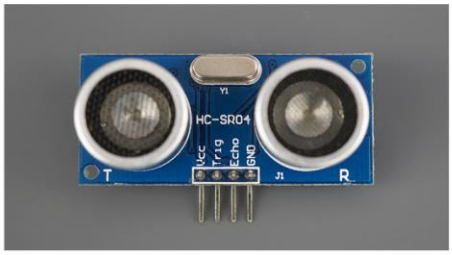
Проблеми офлайн-тестування

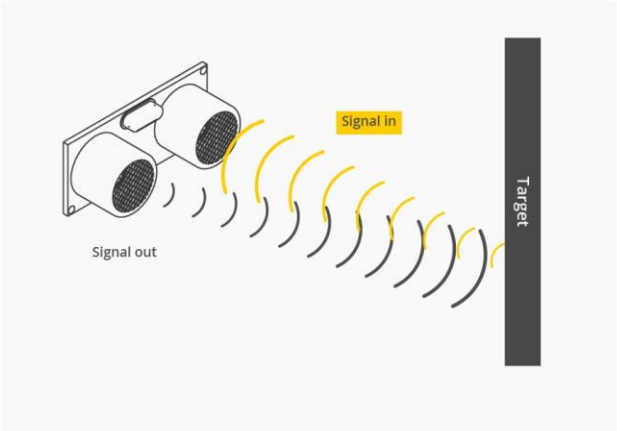


- Присутність інших статистичних вибірок
- Вартість обладнання
- Втрата офлайн-шопінгу домінації на ринку
- Неможливість точних вимірів за допомогою камер

5

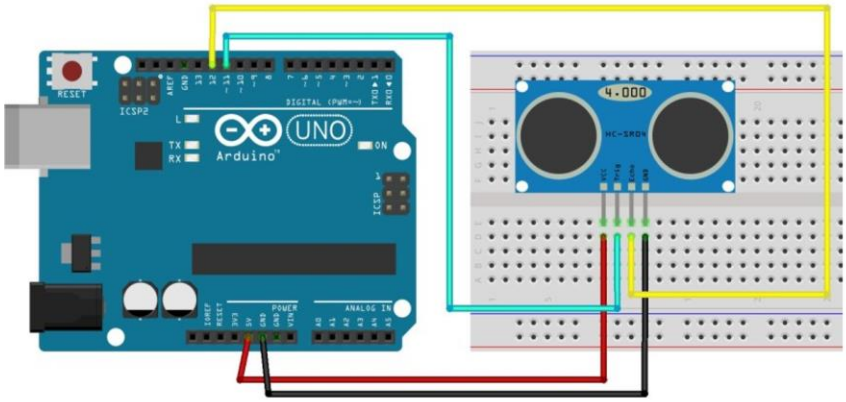
Ідея рішення





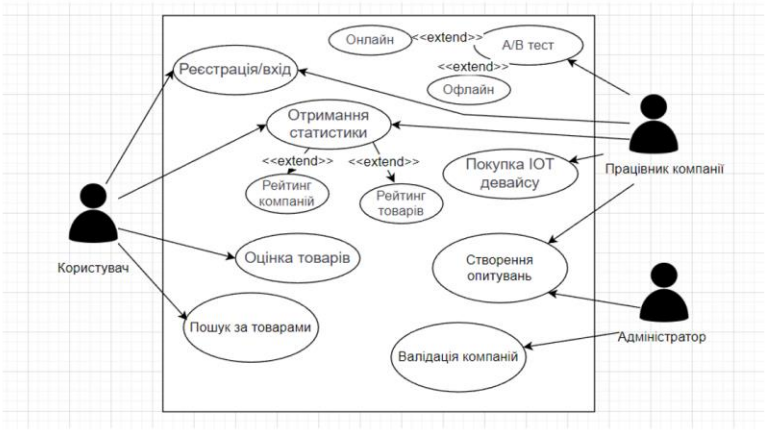
6

Схема підключення розумного пристрою



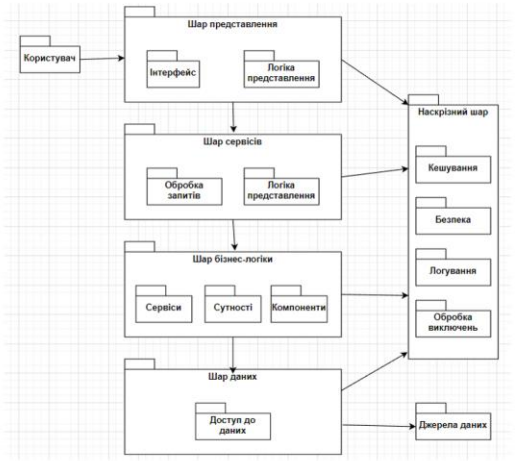
7

Діаграма прецендетів



8

Діаграма пакетів



Технології для розробки

ARDUINO

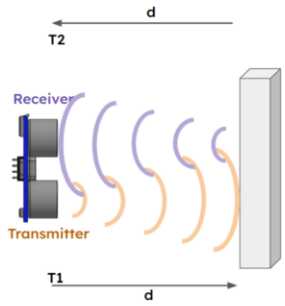
TypeScript

node

express

NEXT.JS

Первинна обробка даних

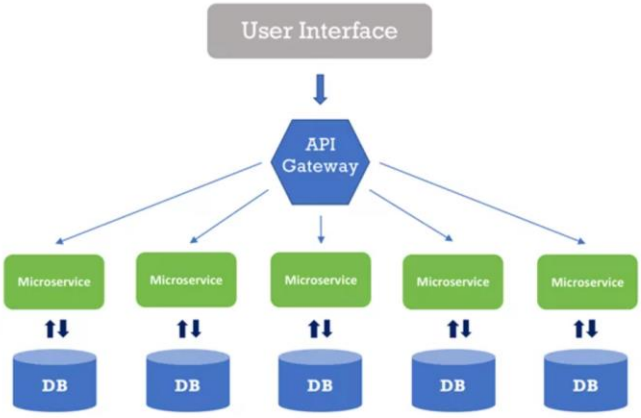


Швидкість звуку в повітрі: **c = 340 m/s**

Формула відстані: **d = c*ΔT/2**

```
void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);

  curDistance = (duration/2) / 29.1;

  if (abs(curDistance - previousDistance) >= 5) {
    unsigned long currentTime = millis();
    Serial.print(curDistance);
    Serial.print("cm ");
    Serial.print(currentTime);
    Serial.println("ms");
    previousDistance = curDistance;
    previousTime = currentTime;
  }

  delay(500);
}
```


Архітектура серверної частини



1. Сервіс користувача
2. Сервіс компанії
3. Сервіс тестів

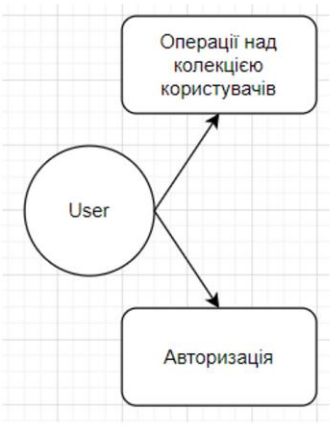
База даних

MongoDB – нереляційна база даних.

- Гнучкість
- Масштабованість
- Висока продуктивність

13

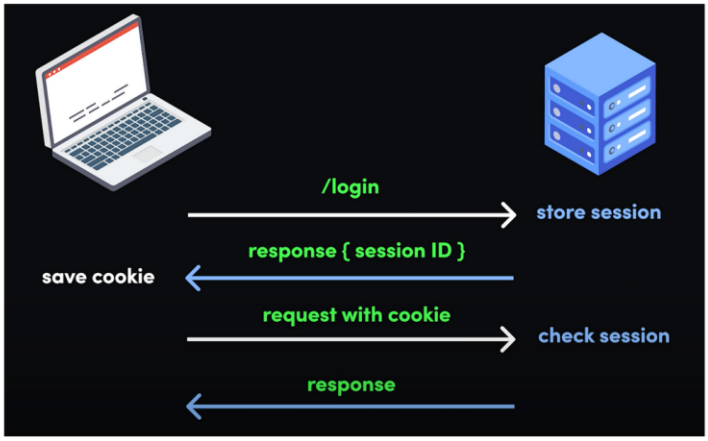
Сервіс користувача



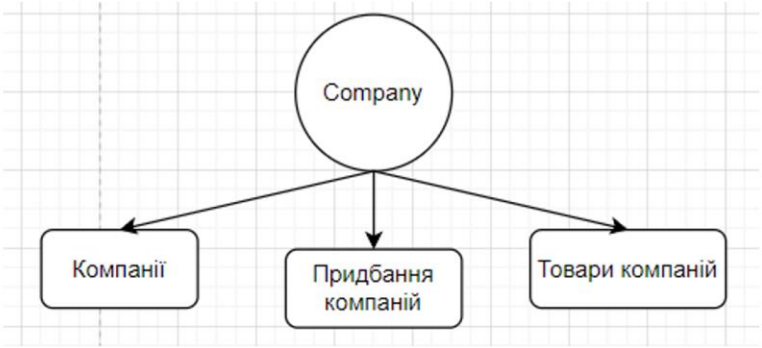

```
const UserSchema = new mongoose.Schema({
  login: {
    type: String,
    required: true
  },
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  avatarUrl: String,
  role: {
    type: String,
    default: "User"
  },
  authentication: {
    password: {type: String, required: true, select: false},
    salt: {type: String, select: false},
    sessionToken: {type: String, select: false},
  }
});
```

14

Авторизація



Сервіс компанії

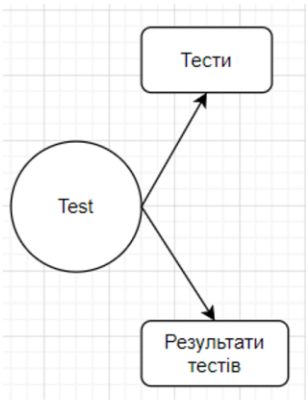


Stripe API

```
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY, {
  apiVersion: "2022-11-15",
});

async CreatePaymentSession(data: {
    returnUrl: string;
    amount: number;
    companyId: string;
  }) {
    try {
      const session = await
stripe.checkout.sessions.create({
        payment_method_types: ["card"],
        line_items: [
          {
```

```
price_data: {
            currency: "usd",
            product_data: {
              name: "Top up balance",
            },
            unit_amount: data.amount * 100,
          },
          quantity: 1,
        },
      ],
      mode: "payment",
      success_url: `${data.returnUrl}/company/${data.companyId}`,
      cancel_url: `${data.returnUrl}/company/${data.companyId}`,
    });
    return { id: session.id };
  } catch (err) {
    throw err;
  }
}
```


Сервіс тестів




```
export const runTestingForDuration = (
  duration: number
): Promise<Approach[]> => {
  return new Promise((resolve, reject) => {
    let res: Approach[] = [];
    const parsers = SerialPort.parsers;
    const parser = new parsers.Readline({
      delimiter: "\r\n",
    });
    const port = new SerialPort("COM3", {
      baudRate: 9600,
      dataBits: 8,
      parity: "none",
      stopBits: 1,
      flowControl: false,
    });
    port.pipe(parser);
    parser.on("data", function (data) {
      const numbers = data.match(/\d+/g);
      res.push({ distance: numbers[0], time: numbers[1]
}); });
    setTimeout(() => {
      port.close();
      resolve(res);
    }, duration);
  });
};
```

Основна фільтрація результатів

```
async GetAllResults(testId: string) {
  try {
    let results: Result[] = await ResultModel.find({ testId });
    let allClients = 0;
    if (!results.length) return { results, allClients };
    const defaultDistance = results[0].approaches[0].distance;
    let curApproachDistance = 0;
    let curApprocahTime = 0;
    for (let index in results) {
      let apps: Approach[] = [];
      const { approaches } = results[index];
      for (let i = 1; i < approaches.length; i++) {
        if (this.isApproach(approaches[i].distance, defaultDistance)) {
          if (this.isApproach(approaches[i - 1].distance, defaultDistance)) {
            curApproachDistance = Math.min(
              curApproachDistance,
              approaches[i].distance
            );
            curApprocahTime += this.countTimeDifference(
              approaches[i - 1].time,
              approaches[i].time
            );
          } else {
            curApproachDistance = approaches[i].distance;
          }
```

```
        } else {
          if (curApproachDistance === 0) continue;
          apps.push({
            distance: curApproachDistance,
            time:
              curApprocahTime +
              this.countTimeDifference(
                approaches[i - 1].time,
                approaches[i].time
              ),
          });
          curApproachDistance = 0;
          curApprocahTime = 0;
        }
      }
      results[index].approaches = [...apps];
      allClients += apps.length;
    }
    return { results, allClients };
  } catch (err) {
    throw err;
  }
}
```


Біноміальний розподіл

Біноміальний розподіл – ймовірнісний розподіл, який описує кількість успіхів у фіксованій кількості незалежних дискретних випробувань

$$P(X = k) = \binom{n}{k} \times p^k \times (1-p)^{n-k}$$

```
async BinomialDistribution(
    testId: string,
    peopleNumber: number = 100
  ): Promise<BinomialResult> {
    const { allClients, allApproaches } = await
this.getAllResultsByTestId(testId);
    const distributionArray = [];
    for (let i = 0; i <= peopleNumber; i++) {
      const calculatedProbability =
        this.combinations(peopleNumber, i) *
        Math.pow(p, i) *
        Math.pow(1 - p, peopleNumber - i);
      const probability = parseFloat((calculatedProbability * 100).toFixed(2));
      if (probability > 0) {
        distributionArray.push({
          people: i,
          probability,
        });
      }
    }
    return distributionArray;
  }
```

Розподіл Пуассона

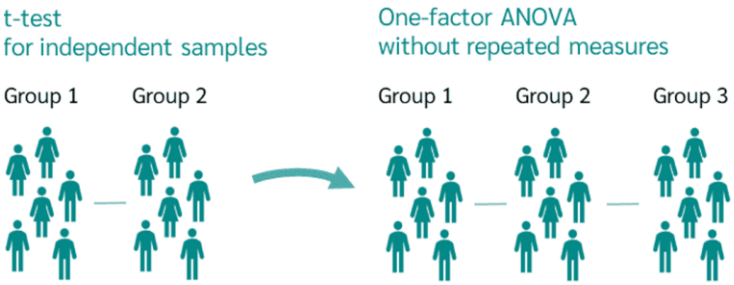
Розподіл Пуассона – ймовірнісний розподіл, який описує кількість незалежних подій, що відбуваються в фіксованому інтервалі часу.

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

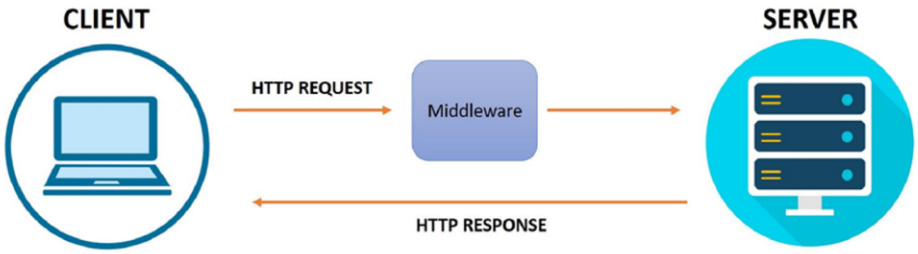
```
async PoissonDistribution(
  testId: string,
  start: string,
  end: string
): Promise<PoissonResult> {
  const lambda = await this.poissonLambdaCount(testId, start, end);
  let probabilities = [];
  let k = 0;
  while (true) {
    const probability = parseFloat(
      ((Math.pow(lambda, k) * Math.exp(-lambda)) / this.factorial(k)).toFixed(
        2
      )
    );
    if (k > lambda && probability === 0) {
      break;
    }
    probabilities.push(probability);
    k++;
  }
  const probabilitySums: number[] = probabilities
    .reverse()
    .reduce((acc: number[], cur: number) => {
      const lastSum = acc.length > 0 ? acc[acc.length - 1] : 0;
      acc.push(lastSum + cur);
      return acc;
    }, []);
  probabilitySums[probabilitySums.length - 1] = 1;
  return probabilitySums
    .reverse()
    .slice(1)
    .map((prob, i) => ({
      people: i + 1,
      probability: Math.round(prob * 100),
    }));
}
```


Дисперсний аналіз (ANOVA)

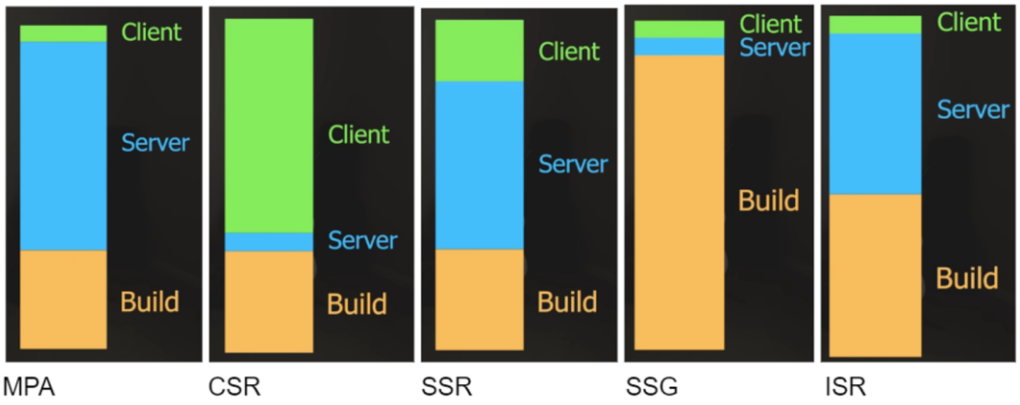
Дисперсійний аналіз – статистичний метод, який використовується для порівняння середніх значень кількох груп та визначення, чи є між ними статистично значущі відмінності.



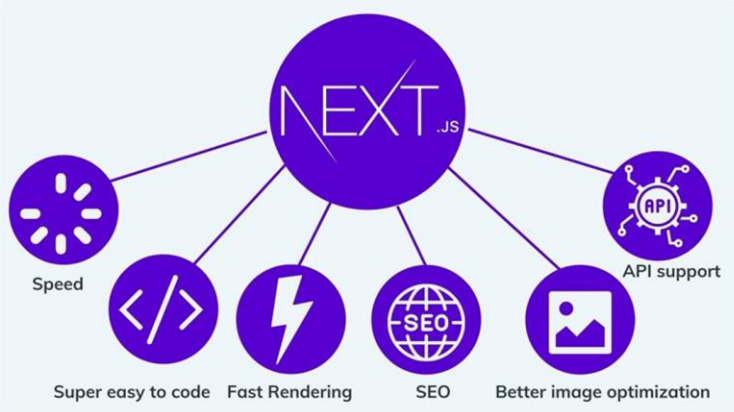
Middleware та спільні пакети



Клієнтська частина



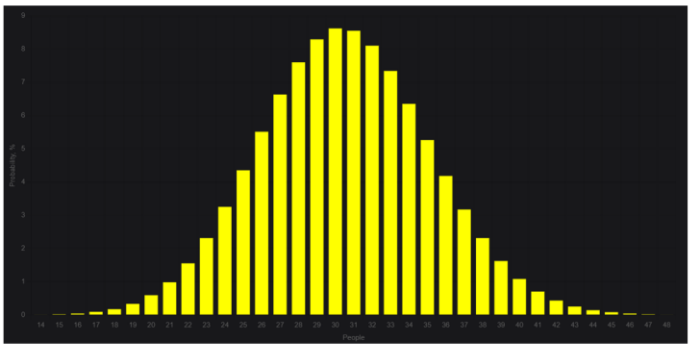
Інші переваги Next.js



25

Зображення графіків

Chart.js



26

Тестування

File	% Stmts	% Branch	% Funcs	% Lines
All files	95.07	90	98.18	94.65
src	100	100	100	100
express-app.ts	100	100	100	100
index.ts	100	100	100	100
src/api	95.83	50	100	95.4
app-events.ts	100	100	100	100
auth.ts	94.23	50	100	94
index.ts	100	100	100	100
user.ts	96	100	100	95.65
src/config	100	100	100	100
index.ts	100	100	100	100
src/database	72.72	100	100	66.66
connection.ts	72.72	100	100	66.66
src/database/models	100	100	100	100
User.ts	100	100	100	100
src/services	94.89	92	100	94.79
AuthService.ts	98.27	100	100	98.21
UserService.ts	90	71.42	100	90
src/utils	95.23	100	90	94.44
app-errors.ts	83.33	100	75	83.33
check-auth.ts	100	100	100	100
error-handler.ts	88.88	100	100	85.71
index.ts	100	100	100	100
validations.ts	100	100	100	100

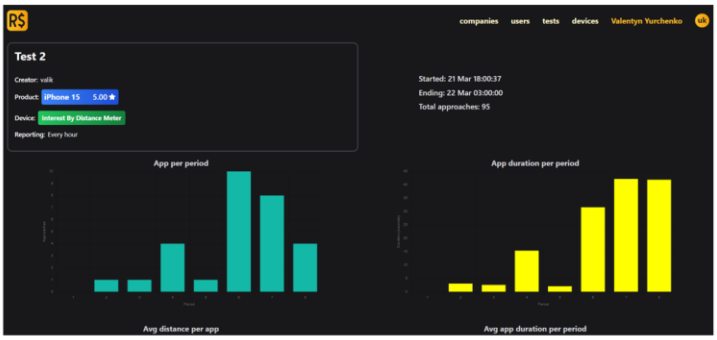
Test Suites: 1 passed, 1 total
Tests: 26 passed, 26 total

Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	345785	14	0.00%	227.89	0	24129	97.00	403.00	466.00	1424.98	75.77	1683.22	1261.93
get all users	1800	3	0.17%	492.14	125	12672	262.00	763.70	1403.90	6135.16	0.52	152.51	10.19
get one user	1800	0	0.00%	76.51	15	5080	28.00	76.00	182.95	1106.15	0.52	26.22	0.20
sign-up	1800	1	0.06%	62.96	2	5017	17.50	85.00	177.50	809.00	0.52	1.54	0.23
sign-in	1800	0	0.00%	100.73	2	5185	59.00	147.90	250.00	956.84	0.52	0.32	0.22
create company	1800	1	0.06%	110.64	3	5339	61.00	153.90	262.95	1056.91	0.52	0.47	0.24
create company product	1800	0	0.00%	112.96	3	5211	62.00	159.90	262.95	1037.55	0.52	0.33	0.23
rate company product	1800	0	0.00%	113.73	2	4848	62.00	158.00	262.90	1073.94	0.52	0.35	0.24
top up company balance	1800	0	0.00%	113.66	4	5282	61.50	154.90	262.00	1067.47	0.52	0.24	0.24
create test	1800	0	0.00%	96.00	15	5269	49.00	153.00	270.95	977.40	0.52	0.31	0.24
start test	1800	1	0.06%	102.93	2	5695	56.00	158.90	285.95	934.25	0.52	0.34	0.24
buy smart device	1800	0	0.00%	103.13	3	5723	54.00	168.00	277.35	968.89	0.52	0.26	0.24
top up company balance	1800	0	0.00%	97.88	2	5602	52.50	162.90	280.75	842.82	0.52	0.55	0.24
get test results	1800	0	0.00%	95.62	2	5608	51.00	159.00	278.00	825.00	0.52	0.30	0.24
get test distribution	1800	0	0.00%	89.82	2	4875	47.00	149.90	276.55	818.96	0.52	0.34	0.23

Висновки

- Розумний датчик, що визначає зацікавленість клієнтів до зовнішнього вигляду товару
- Аналіз отримуваних даних кількома математичними методами
- Сервер на Node.js із використанням архітектури мікросервісів
- Клієнт на Next.js з використанням прогресивних методів серверного рендерингу
- Зручне візуальне подання результатів
- Відповідність загальноприйнятим стандартам чистого коду



Дякую за увагу!