ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота

на тему: «**МАГАЗИН ВИСОКОНАВАНТАЖЕНОЇ ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА МОВІ ПРОГРАМУВАННЯ GO»**

на здобуття освітнього ступеня бакалавра
зі спеціальності __122 Комп'ютерні науки_____
(код, найменування спеціальності)
освітньо-професійної програми __Комп'ютерні науки _____
(назва)

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

_____ ___Їсян ВАН_____
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. __КНД-41_____

_____Їсян ВАН_____
(Ім'я, ПРІЗВИЩЕ)

Керівник: _Петро КРАВЧУК – PhD, доцент__
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

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

Київ – 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук
Ступінь вищої освіти − «Бакалавр»
Спеціальність підготовки − «122 Комп'ютерні науки»
Освітньо-професійна програма − «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Віктор Вишнівський
"____" _____2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Їсян ВАН_____
(прізвище, ім'я, по батькові)

1.Тема кваліфікаційної роботи: «Магазин високонавантаженої електронної комерції на мові програмування Go»
Керівник кваліфікаційної роботи __Петро КРАВЧУК – PhD, доцент_____ __,
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56_____.
2. Строк подання студентом роботи 30.05.2025 р.
3. Вихідні дані до роботи:
Тема: Магазин високонавантаженої електронної комерції на мові Go
Мета: Розробка вебсистеми, здатної обробляти великий потік одночасних запитів
Засоби реалізації: Go, Iris, Redis, MySQL, RabbitMQ, HTML/CSS/JS, Docker
Очікуваний результат: Стабільна, масштабована система онлайн-продажів із захистом від перевантаження
Об'єкт: Веб-магазин із високою конкурентністю покупок
Предмет: Програмна архітектура високонавантажених систем

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____

1. Аналіз предметної області та обґрунтування вибору технологій
2. Огляд мови програмування Go та фреймворку Iris
3. Проєктування архітектури системи (MVC, розподілена структура)
4. Розробка backend-частини
5. Розробка frontend-інтерфейсу користувача
6. Забезпечення конкурентності та оптимізація
7. Безпека
8. Docker-розгортання системи
9. Тестування системи
10. Висновки та перспективи подальшого розвитку

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання 25.02.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Визначення мети та технічного завдання		виконано
2	Аналіз предметної області та вибір технологій		виконано
3	Проєктування архітектури системи		виконано
4	Розробка серверної частини (Back-end)		виконано
5	Розробка клієнтської частини (Front-end)		виконано
6	Реалізація системної оптимізації та безпеки		виконано
7	Тестування та налагодження програмного забезпечення		виконано
8	Підготовка пояснювальної записки та захист роботи		виконано

Здобувач вищої освіти _____ __Їсян ВАН_____
(підпис) (Ім'я, ПРІЗВИЩЕ)

Керівник
кваліфікаційної роботи _____ ___Петро КРАВЧУК____
(підпис) (Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Бакалаврська робота присвячена розробці системи електронної комерції з високим рівнем навантаження, реалізованої мовою програмування Go. У межах роботи створено програмне забезпечення для інтернет-магазину, здатного ефективно обробляти велику кількість одночасних запитів під час флеш-розпродажів. Основна мета — забезпечити високу продуктивність, масштабованість і стійкість системи при навантаженнях.

У ході роботи розроблено як бекенд, так і фронтенд частини системи. На серверній частині застосовано фреймворк Iris та архітектурний шаблон MVC, використано бази даних MySQL і Redis для обробки та кешування даних, а також систему обміну повідомленнями RabbitMQ для зменшення навантаження на сервер. На стороні клієнта реалізовано адаптивний інтерфейс за допомогою HTML, CSS, JavaScript та фреймворку Bootstrap. Особлива увага приділена безпеці, контролю кількості замовлень та захисту від перевантажень.

Для досягнення високої продуктивності впроваджено механізми розподіленого хешування, оптимізовано логіку обробки замовлень, впроваджено обмеження доступу до функцій користувача через cookies та AES-шифрування. Також проведено тестування системи на предмет функціональності, безпеки та стійкості до великого трафіку.

У результаті розроблено інтернет-магазин, який може обслуговувати значну кількість користувачів одночасно, зберігаючи стабільність і високу швидкість реагування. Проєкт має перспективу подальшого розвитку та впровадження у сфері електронної комерції.

ЗМІСТ

Вступ **9**

1 Документація користувача **11**

1.1 Система закупівель з високим рівнем паралельності 11

1.2 Основні інструменти та методи 12

1.2.1 Back-end.....12

1.2.2 Front-end.....15

1.2.3 Середовище..... 16

1.2.4 Підготовка середовища 17

1.2.5 Встановлення та запуск проєкту..... 17

1.2.6 Розгортання Docker 18

1.3 Результат роботи системи закупівель з високою паралельністю 19

1.3.1 Перспектива користувача..... 19

1.3.2 Перспектива адміністратора 21

2 Документація для розробників **25**

2.1 Розробка системи закупівель з високою конкурентністю 25

2.1.1 Back-end Розробка управління продуктами 25

2.1.2 Back-end Розробка управління замовленнями 31

2.1.3 Fronted-end Розвиток ключових функцій 35

2.1.4 Логічне керування кількісними параметрами..... 42

2.2 Оптимізація для системи закупівель з високим рівнем паралельності 44

2.2.1 Front-end Оптимізація 44

2.2.2 Back-end Оптимізація..... 49

2.3 Тестування..... 55

2.3.1 Плани тестування 56

2.3.2 Письмові тести..... 56

2.3.3 Результати тестування 60
2.3.4 Висновки тестування 63
Висновок **65**
Перелік посилань **67**
Презентація **69**

ВСТУП

З розвитком інформаційних технологій зростають вимоги до програмного забезпечення з високою актуальністю, високою продуктивністю та високою доступністю. Коли ми стикаємося з такими ситуаціями, способи вирішення технічних вимог стають все більш важливими та популярними. Останнім часом багато провідних IT-компаній вирішили використовувати Go [1] в програмній інфраструктурі та бізнес-системах. Go підтримує мільйони TCP [2] з'єднань та має низьку продуктивність та споживання пам'яті. Крім того, Go підтримує паралельність на рівні мови програмування, використовуючи goroutines та канали [3]. Вищезазначені функції Go можуть дуже допомогти при розробці високопродуктивного програмного забезпечення. Завдяки належній архітектурі програмного забезпечення нам не потрібно писати багато коду, щоб впоратися з великим потоком даних у ситуації високої продуктивності, що виправдовує філософію Go «Менше - це експоненціально більше» [4]. Аналогічно, хороший вибір та архітектура розробки програмного забезпечення, ймовірно, дозволять досягти більшої кількості технічних вимог з меншими витратами на серверні ресурси та розробку, експлуатацію та обслуговування програмного забезпечення.

Інтернет-шопінг стає все більш популярним у наш час з розвитком суспільства. Оскільки в інтернет-магазинах немає фактичних дверей, вони, як правило, можуть вмістити більше покупців, ніж фізичні магазини. Однак ми можемо відчути, наскільки переповнено торговельні центри під час фестивалю шопінгу, але чи можемо ми уявити, що інтернет-магазини можуть витримувати більший тиск з мільярдами клієнтів одночасно? Якщо ми хочемо уникнути збоїв або випадкового закриття інтернет-магазинів, нам потрібно створити потужну систему для таких веб-сайтів, щоб вони могли витримувати великий потік даних та постійно стабільно працювати.

У цій дипломній роботі буде розроблено систему з високою

конкурентністю покупок для флеш-продажів на веб-сайтах електронної комерції, яка працюватиме з високою швидкістю потоку даних та великим трафіком. Крім того, вона зможе вирішувати проблеми перепроданості та матиме функції для масштабування. Завдяки спеціальним інструментам, ефективним алгоритмам та зручній архітектурі програмного забезпечення ми зможемо реалізувати вищезазначені функції та відповідати вимогам.

Як було проілюстровано вище, передумови та мотивація програмного забезпечення, у наступних розділах, у другому та третьому, ми ознайомимося з процесом розробки програмного забезпечення з точки зору користувача та розробника.

У другому розділі буде представлено документацію користувача з мотивацією та вимогами до програмного забезпечення. Крім того, будуть розглянуті основні методи та інструменти, що стосуються бекенду, фронтенду, середовища та операцій. Користувач зможе успішно запускати систему з високою конкурентністю за допомогою детальних методів та процедур, а також перевіряти результати роботи.

У третьому розділі документація розробника буде присвячена торговим системам для адміністратора бекенду та клієнта фронтенду. По-перше, присвячений розробці веб-сайту електронної комерції з основними функціями. Потім ми виконуємо оптимізацію, щоб оснастити систему функціями високого паралельного виконання та масштабування. Ми проведемо тестування, щоб перевірити, чи є наша система зручною для використання, стабільною, безпечною та відповідає всім вимога.

1 ДОКУМЕНТАЦІЯ КОРИСТУВАЧА

У цьому розділі буде надано короткий опис аналізу вимог. Крім того, ми представимо основні методи та інструменти, що використовуються в програмному проекті, з детальною інформацією про їх використання. Читач зрозуміє програмне забезпечення та навчиться працювати з ним за допомогою правильних інструкцій.

1.1 Система закупівель з високим рівнем паралельності

З розвитком інтернету спостерігається зростаюча тенденція до онлайн-покупок. Загалом, онлайн-покупки надають клієнтам більше зручності та вибору. Крім того, клієнти, ймовірно, отримають нижчу ціну, ніж на ті ж товари у фізичних магазинах.

Коли настають свята покупок, такі як Чорна п'ятниця чи Різдво, на товари зазвичай діють значні знижки у всіх магазинах (фізичних та онлайн). Оскільки під час таких свят у фізичних магазинах завжди трапляється багатолюдність, чи можемо ми припустити, що в інтернет-магазинах набагато більше людей? Відповідь – однозначно так! Веб-сайти покупок матимуть набагато більше відвідувачів та клієнтів завдяки високим знижкам на розпродажі товарів. Потім виникає друге питання: як веб-сайт покупок справляється з великим трафіком даних, а як система покупок справляється з величезними замовленнями?

Це програмне забезпечення вирішить вищезазначені проблеми з великим потоком даних та вирішить проблеми перепроданості у ситуаціях з високим рівнем завантаження.

1.2 Основні інструменти та методи

1.2.1 Back-end

Back-end система розроблена з використанням мови програмування Go [1] з фреймворком Iris [5] на основі шаблону Model-View-Controller (MVC) [6] та великими базами даних з MySQL [7] та Redis [8]. RabbitMQ [9] обробляє черги повідомлень [10] для обміну повідомленнями (у режимах надсилання та отримання).

Go

Go (також званий Golang або мовою Go) — це мова програмування з відкритим вихідним кодом, багатопарадигмальна, статично типізована, компільована, розроблена Google. Go, на відміну від інших мов програмування, надає прості та відстежувані функції паралельної роботи, що дозволяє розробникам додатків зручно взаємодіяти із запитами між виділеними ресурсами, серверами, базами даних та мережами. Крім того, вона надає потужну стандартну бібліотеку та корисні інструменти, такі як Gofmt, Gorun, Godoc. Go також має функції збору сміття та підтримує можливості тестування.

Iris

Iris — це веб-фреймворк, що слугує зручною та ефективною кросплатформною системою з потужним набором функцій. За допомогою Iris ми можемо створювати публічні або приватні веб-додатки та API з високою продуктивністю, портативністю та потенціалом.

Як веб-фреймворк Go з відкритим кодом, Iris був створений Герасимосом Маропулосом для розробки сучасних веб-додатків. За допомогою фреймворку Go Iris розробники програмного забезпечення, ймовірно, зможуть створювати надшвидкі веб-додатки з меншими зусиллями.

Порівняно з іншими модулями Go, Iris підтримує розробників з високоякісним архітектурним шаблоном MVC модель-вид-контролер у веб-додатках як на рисунок 1.1 показано.

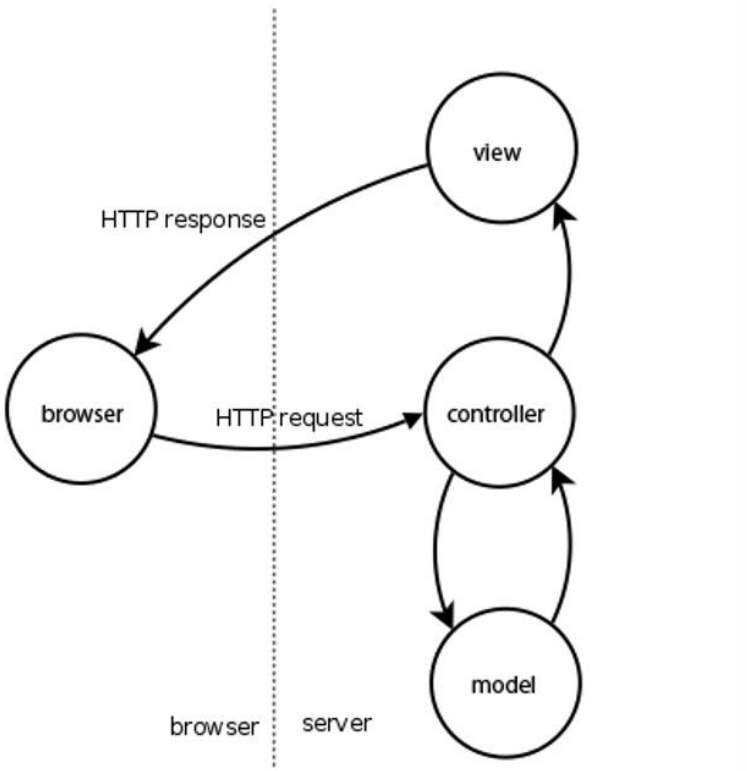


Рисунок 1.1 – Структура MVC

Щоб використовувати цей архітектурний шаблон MVC (Model View Controller), вам просто потрібно імпортувати пакет MVC в iris.

RabbitMQ

RabbitMQ — це популярний брокер повідомлень з відкритим кодом, орієнтований на повідомлення, проміжне програмне забезпечення. Він був розроблений для підтримки протоколу черги розширених повідомлень (AMQP) і згодом розширений за допомогою архітектури плагінів для підтримки протоколу текстово-орієнтованих повідомлень STOMP, транспорту телеметрії MQ (MQTT) та інших протоколів.

Нам потрібно встановити пакет abmq, щоб використовувати RabbitMQ. Як показано на рисунок 1.2, «P» працює як продуцент, а «C» — як споживач. Середній блок — це черга, яка служить буфером повідомлень [10], який RabbitMQ зберігає, представляючи споживача.

Рисунок 1.2 – RabbitMQ

MVC

Коли ми вивчаємо програмування або працюємо в процесі розробки програмного забезпечення, іноді трапляються поширені ситуації, коли деякі кодові бази легко перевірити. Водночас деякі є заплутаними та важкими для сприйняття, підтримки та оновлення. У більшості випадків це залежить від організації та структури коду [6]. Знання відповідної частини коду буде дуже корисним, коли ми захочемо додати нові функції чи можливості, виправити помилку або виконати будь-які інші операції з програмою. Якщо структури програми добре розроблені та представлені, ми, ймовірно, здогадаємося про розташування коду для роботи, навіть якщо ніколи їх не бачили. MVC, що представляє Model-View-Controller, — це шаблон проектування програмного забезпечення, який в основному використовується для розробки графічних інтерфейсів користувача (GUI) та веб-застосунків.

Модель [11] є центральною частиною шаблону MVC, яка представляє динамічну структуру даних та відображає інтерфейс користувача програми. Вона безпосередньо підключається та керує даними, логікою та правилами програми. [12] Як правило, вона взаємодіє з базами даних, а в інших випадках взаємодіє з даними з інших API або сервісів.

Представлення [13] представляє інформацію діаграми, схеми або таблиці.[14] Основна функція представлень полягає у відображенні даних, тобто, маючи певну сторінку з пов'язаними даними для відображення, ми можемо використовувати представлення для генерації правильного виводу. Якщо ми хочемо реалізувати застосунок за допомогою фреймворку MVC, ми можемо повернути HTML-код, відображений сервером, до браузера кінцевого користувача. Крім того, ми також можемо досягти цього за допомогою обробки

типів даних відображення, таких як XML, JSON тощо. Важливо зазначити, що ми повинні додавати якомога менше логіки до наших кодів, щоб уникнути неочікуваних проблем. Оскільки представлення в основному використовується для відображення даних, воно повинно мати чітку структуру з нещільним зв'язком.

Контролер [15] зазвичай отримує вхідні дані та перетворює їх на команди як сигнал для моделей та представлень. Замість безпосереднього запису в базу даних або встановлення HTML-відповідей, контролери [16] безпосередньо взаємодіють з вхідними даними за допомогою відповідних моделей, представлень та інших пакетів. Оскільки головною метою контролера є аналіз даних та їх надсилання іншим типам і функціям для обробки, нам не слід вкладати в контролер забагато логіки.

1.2.2 Front-end

У front-end розробці ми використовуємо HTML, CSS та JavaScript як мови програмування, а Bootstrap — як фреймворк.

HTML

Мова гіпертекстової розмітки (HTML) [17] є одним із найбазовіших компонентів побудови веб-сторінок як мови розмітки. Вона визначає значення, структуру та вміст веб-сторінки. Окрім HTML, ми використовуємо CSS для відображення зовнішнього вигляду або презентації веб-сторінки та JavaScipt для опису функціональності та поведінки веб-сторінки. «Гіпертекст» – це текст, який використовується для створення посилань, що з’єднують веб-сторінки з іншими веб-сайтами. «Розмітка» – це теги HTML для анотації зображень, тексту, структури та вмісту того, що відображається веб-браузером.

CSS

Каскадні таблиці стилів (CSS) [18] – це мова таблиць стилів, яка визначає, як HTML або XML відображаються у веббраузері. Вона показує, як елементи відображаються на екрані [18], і може керувати макетами кількох вебсторінок

одночасно. [19] Як основна мова програмування для відкритого Інтернету, CSS стандартизований і має три офіційні версії специфікацій: CSS1, CSS2.1 та CSS3.

JavaScript

JavaScript (JS) [20] — це інтерпретована мова програмування клієнтських скриптів з легкими кросплатформними функціями. Як одна з найпопулярніших мов програмування, JS широко використовується у веб-розробці. Крім того, вона також дуже популярна в розробці програмного забезпечення не в веб-браузерах, таких як Node.js, Apache CouchDB та Adobe AcrobatBy. [21] Більше того, JS можна використовувати як у клієнтській, так і в серверній розробці. На стороні клієнта об'єкти зможуть керувати браузером або моделлю об'єктів документа (DOM). Наприклад, корисні клієнтські бібліотеки, такі як AngularJS, VueJS та ReactJS, можна використовувати для розміщення елементів HTML-форми та взаємодії з подіями користувача.

Bootstrap

Bootstrap [22] — це потужний набір інструментів, що містить інструменти HTML, CSS та JavaScript для веб-розробки. Він розміщений на GitHub та створений Twitter як фреймворк з відкритим кодом та зручний для користувача. Перевага адаптивного дизайну забезпечує стабільну роботу браузера та можливість його узгодженого проектування з компонентами, що повторно використовуються. Завдяки багатому розширенню JavaScript, Bootstrap підтримує вбудовані функції для плагінів jQuery та JS API. Оскільки ми можемо використовувати Bootstrap у будь-якому IDE або редакторі, він забезпечує веб-дизайнерам та розробникам значну зручність та інтерес до роботи.

1.2.3 Середовище

Середовище розробки зазвичай служить робочою станцією, де розробники можуть створювати та розробляти свої програми. Тут ми покажемо детальні вимоги до середовища розробки програмного забезпечення, які мають виконувати користувачі. Специфікація системи, що використовується в проекті,

буде наведена нижче:

Операційна система: Windows 10

Пам'ять: 16,0 ГБ

Процесор: 1,80 ГГц

Редактор коду: Goland

Версія Go: go1.15.8 windows/amd64

Фреймворк: iris12

1.2.4 Підготовка середовища

Користувач повинен завантажити наступну програму та пакети, щоб налаштувати середовище та використовувати це програмне забезпечення.

Адреса для завантажити Go: https://go.dev/dl/

Адреса для завантажити Goland: https://www.jetbrains.com/go/download. Студенти можуть зареєструвати освітній обліковий запис і користуватися безкоштовним сервісом під час навчання.

Адреса для завантажити RabbitMQ: https://www.rabbitmq.com/download.html

Адреса для завантажити MySQL: https://www.mysql.com/cn/downloads Адреса для завантажити Navicat: https://www.navicat.com/en/products

1.2.5 Встановлення та запуск проєкту

1. Встановлення проекту

2. Підготовка бази даних MySQL для проекту: Слід вказати налаштування бази даних. Існує багато способів створення конкретної бази даних. Тут ми лише представимо спосіб створення бази даних за допомогою клієнта командного рядка MySQL. Коли ми завантажуємо базу даних MySQL, ми встановлюємо пароль root на 1. У цьому випадку ми вводимо командний рядок бази даних і виконуємо команди для створення бази даних.

3. Щоб запустити проект: Нам потрібно виконати наступний код разом, щоб побачити весь інтерфейс і перевірити основні функції цього програмного забезпечення для успішного запуску проекту.

4. Щоб запустити RabbitMQ.

5. Щоб запустити MySQL.

6. Щоб запустити систему керування серверною частиною: Відкрийте термінал, перейдіть до каталогу серверної частини та запустіть головний файл (для розгортання серверної частини).

7. Щоб запустити функції фронтенду: Відкрийте термінал, перейдіть до каталогу фронтенду та запустіть головний файл (для входу користувача).

8. Щоб запустити getOne.go (Зменшення кількості продукції та уникнення проблеми перепроданості).

9. Запустити consumer.go (Для використання користувачем).

10. Запустити validate.go (Використовується для узгодженої перевірки, перевірки безпеки, підтвердження успішної покупки).

1.2.6 Розгортання Docker

1. Нам потрібно скомпілювати головний файл та перетворити один з фронтендів та бекендів на виконуваний бінарний файл програми Linux. Нам потрібно встановити змінні середовища GOARCH та GOOS, а потім використати команду go build для компіляції програми.

2. Нам потрібно завантажити контейнер net Ubuntu, увійти в систему та скопіювати наші бінарні файли в цей контейнер.

3. Команда cp у docker буде використана для переміщення наших бінарних файлів на останньому кроці до контейнера net Ubuntu.

4. Контейнер з нашим проектом має бути імпортований командою export у docker. Результат експорту буде названо product.tar.

5. Нарешті, ми використаємо команду import у docker, щоб імпортувати product.tar як новий образ docker, який можна завантажити та впровадити на

інших машинах без додаткових налаштувань.

1.3 Результат запуску системи закупівель з високою паралельністю

У цьому розділі буде показано скріншоти системи з високою кількістю одночасних закупівель після запуску. Результати запуску як для користувача, так і для адміністратора будуть відображені наступним чином.

1.3.1 Перспектива користувача

Якщо користувач хоче отримати доступ до нашого сайту покупок з високою ймовірністю одночасного використання ресурсів, спочатку необхідно зареєструватися як легальний користувач. Користувачам потрібно ввести свій нікнейм, ім'я користувача та пароль для завершення реєстрації, як показано на рисунок 1.3.

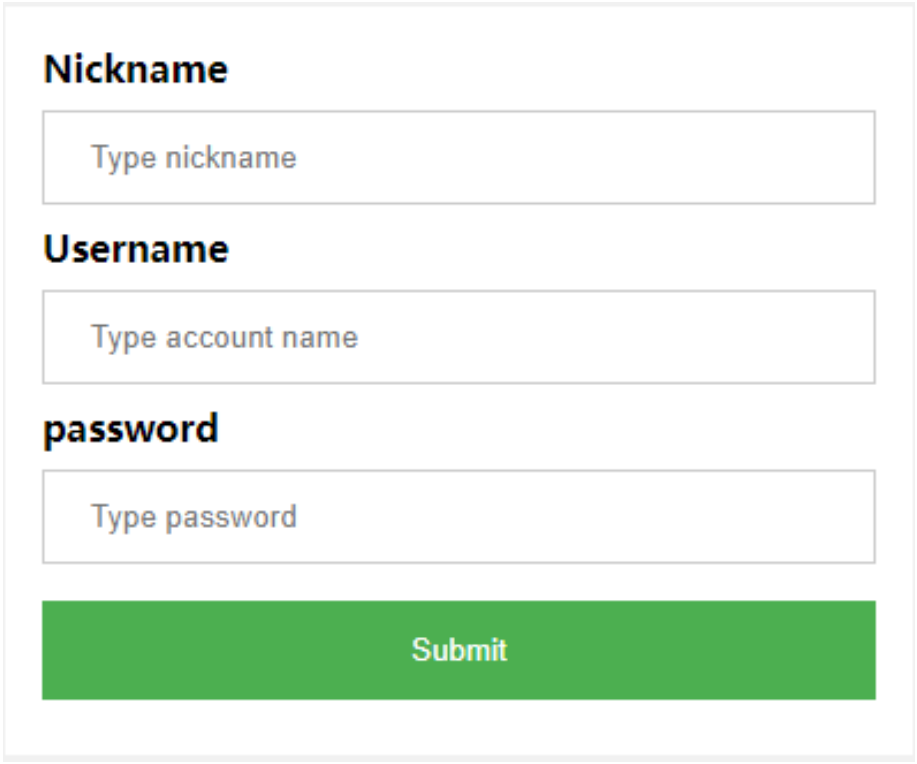


Рисунок 1.3 – Зображення реєстру користувача

Користувач переходить на сторінку входу та вводить номер свого

облікового запису й пароль для входу після завершення реєстрації як на рисунок 1.4 показано. Внутрішня система перевірить та визначить, чи є користувач легітимним, і відповідна логіка буде показана пізніше.

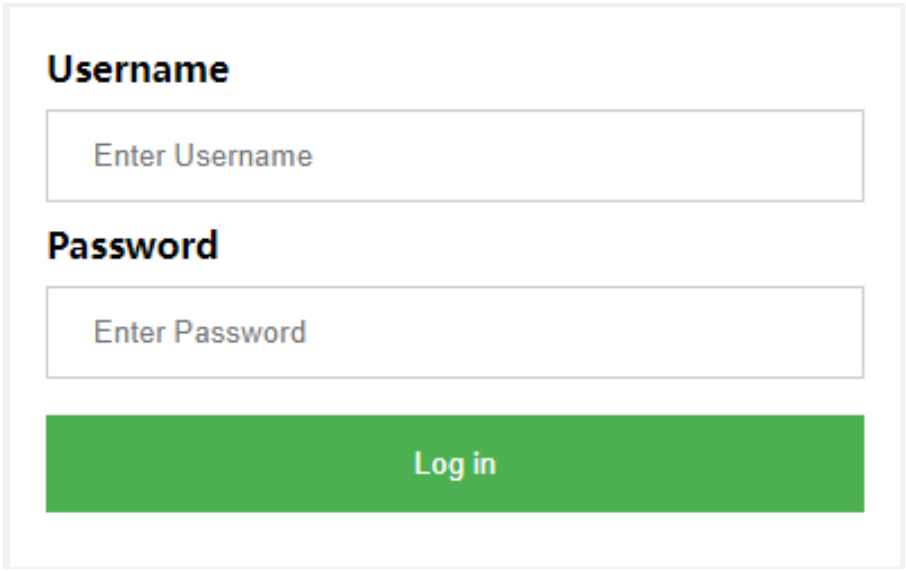


Рисунок 1.4 – Зображення входу користувача

Увійшли в систему користувачі побачать деталі продукту, як показано на рисунку 1.5, і після перегляду деталей продукту вони зможуть натиснути кнопку «Buy Now!», щоб здійснити покупку.

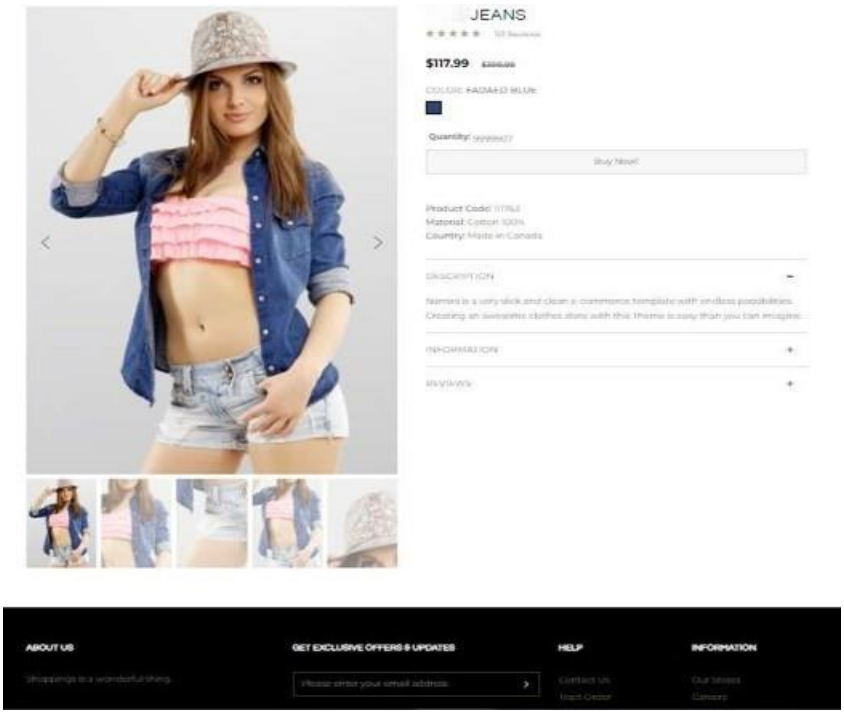


Рисунок 1.5 – Зображення сторінки покупки товару

Front end матиме 10-секундне обмеження інтервалу покупки до наступного запиту на покупку, як показано на рисунку 1.6.

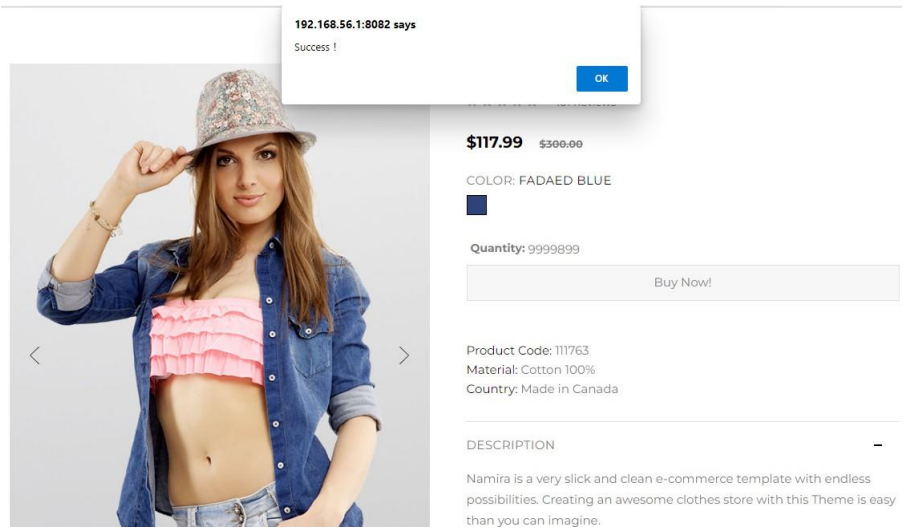


Рисунок 1.6 – Цифра успішної покупки

Front end обмежить наступний запит користувача на покупку, доки кнопка таймера «Buy Now!» не перевищить 10 секунд, як показано на рисунку 1.7.

Рисунок 1.7 – Рисунок паузи для купівлі

1.3.2 Перспектива адміністратора

Адміністратор зможе керувати продуктами та замовленнями в системі після запуску функції main у каталозі бекенду, як показано на рисунку 1.8.

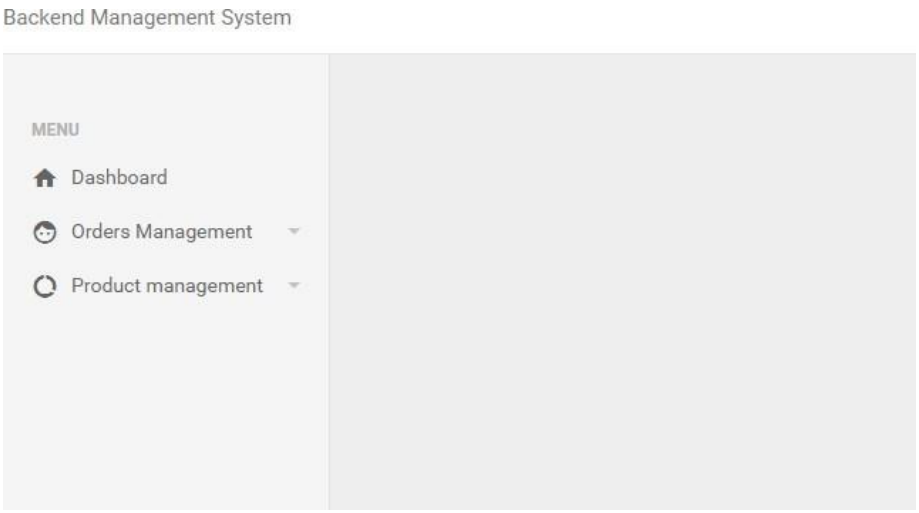


Рисунок 1.8 – Інтерфейс сервера Figure backend

Адміністратор побачить список продуктів після натискання кнопки керування продуктами, яка містить інформацію про кожен продукт, таку як ідентифікатор продукту, номер продукту, зображення продукту тощо. Адміністратор має право змінювати інформацію про продукт як на рисунок 1.9 показано.

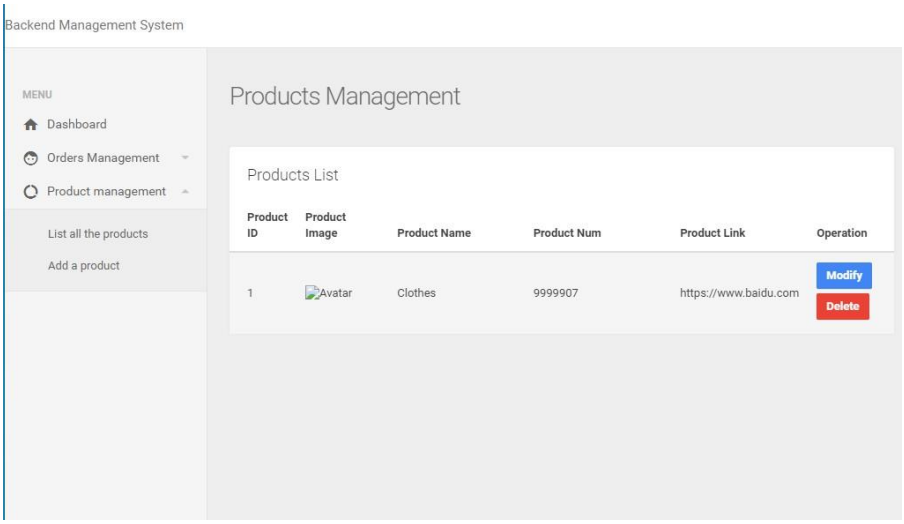


Рисунок 1.9 – Зображення управління продуктом

Адміністратори можуть додавати продукти, натискаючи кнопку «Add». Інформація про продукт містить назву продукту, номер продукту, адресу зображення продукту та посилання на продукт. Як показано на рисунку 1.10, адміністратор може ввести інформацію в порожнє поле та натиснути кнопку «Add», щоб завершити операцію додавання продукту.

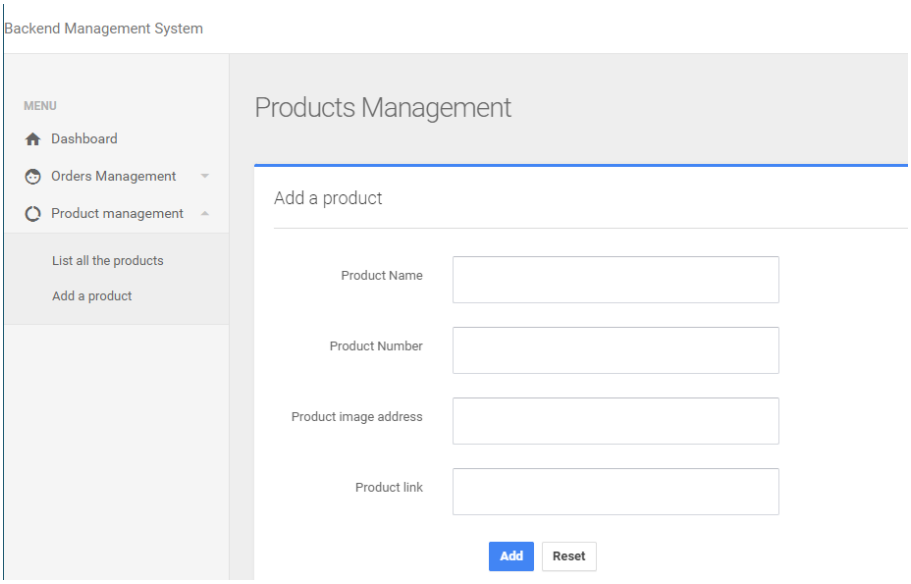


Рисунок 1.10 – Зображення додавання добутку

Адміністратор зможе переглянути всю інформацію про замовлення після натискання кнопки «List all the orders» в блоці керування замовленнями, як показано на рисунку 1.11.

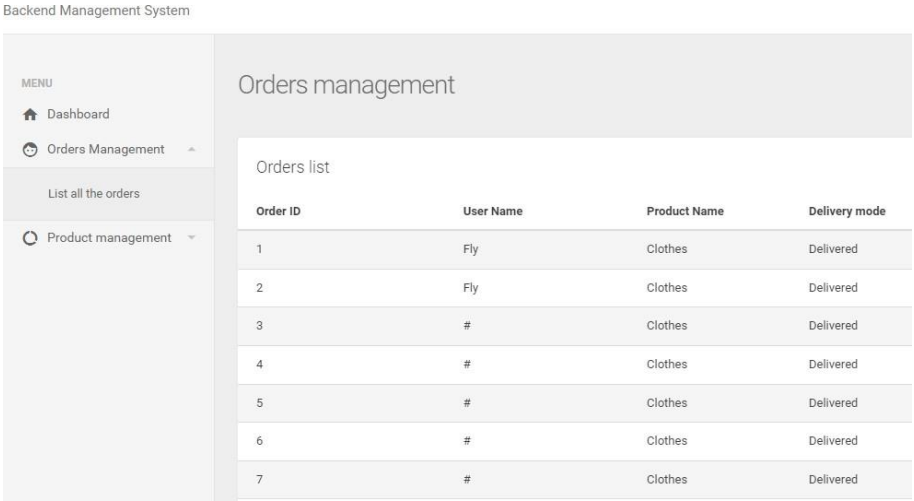


Рисунок 1.11– Фігура порядку

На завершення, ми показали інтерфейси кожної сторінки як на front-end, так і на back-end. Клієнти повинні зареєструвати свої облікові записи перед покупками, щоб забезпечити кращий досвід покупок та захистити веб-сайт від атак. Крім того, процес реєстрації та входу на фронтенді обмежить недійсні відвідування через великий потік даних, який надсилає багато недійсних запитів, що зменшить навантаження на сервер та бази даних. Дійсні клієнти

зможуть робити покупки та купувати свої товари з 10-секундним інтервалом, що дозволяє контролювати великий потік даних та захищати можливості покупок інших клієнтів. Адміністратори зможуть керувати продуктами, наприклад, додавати новий продукт, перевЙіряти та видаляти існуючий продукт. Крім того, адміністратори також можуть керувати базами даних через MySQL для роботи з даними.

Документація користувача вже завершена, і ми представили передумови та мотивацію цього програмного забезпечення з точки зору користувача. Оскільки представлені методи, інструменти та варіанти використання, потенційні користувачі зможуть знати, як правильно працювати з ним, виконавши достатні вимоги. У наступному розділі буде представлено детальний процес розробки з точки зору розробника, що стосується мотивації програмного забезпечення, аналізу вимог та архітектури програмного забезпечення.

2 Документація розробника

Цей розділ відрізняється від документації користувача, і ми представимо процес розробки з точки зору розробника. Після аналізу вимог до програмного забезпечення ми почнемо розробку програмного забезпечення з бекенд-керування та фронтенд-шопінгу за допомогою фреймворку Iris та шаблону MVC. Крім того, оптимізована розробка надасть програмному забезпеченню можливості роботи з великим потоком даних, високою продуктивністю та масштабуванням. Тестова частина перевірить, чи наше програмне забезпечення успішно відповідає всім вимогам.

2.1 Розробка системи закупівель з високим рівнем паралельності

2.1.1 Розробка управління Back-end продуктами

У цьому розділі ми починаємо розробку цього програмного забезпечення з розробки системи управління серверними продуктами, яка складається з трьох частин:

1. Розробка дизайну моделей продуктів.
2. Розробка функцій продуктів (з чотирма основними функціями: вставка, видалення, зміна та перевірка).
3. Інтерфейс керування backend продуктів.

Ми використовуємо framework Iris з моделлю MVC та мову програмування Go для проектування системи управління back-end продуктами.

Ми дотримуємося фреймворку MVC, щоб створити каталог програмного забезпечення наступним чином.

Створення структури каталогів

По-перше, ми розміщуємо функції управління продуктами на серверній частині до back-end/main.go. Усередині каталогу web ми розміщуємо контролери, пов'язані з ресурсами (css, img, js, lib) та представленнями, де знаходиться HTML. По-друге, ми створюємо каталог datamodels як модель MVC для моделей продуктів.

Створити модель даних

Після створення каталогу models ми створюємо файл models.go для кодування структури продукту. Призначаємо ідентифікатор продукту, назву, номер, зображення, URL-адресу з необхідними змінними та типами, після чого частина моделей продукту завершена ,код показано на рисунок 2.1. Діаграма E-R об'єктів, розроблених для нашої системи, детально описана на рисунку 2.2.

```
package models

type Product struct {
  ID           int64  `json:"id"`
  ProductName  string `json:"ProductName"`
  ProductNum   int64  `json:"ProductNum"`
  ProductImage string `json:"ProductImage"`
  ProductUrl   string `json:"ProductUrl"`
}
```

Рисунок 2.1– models.go

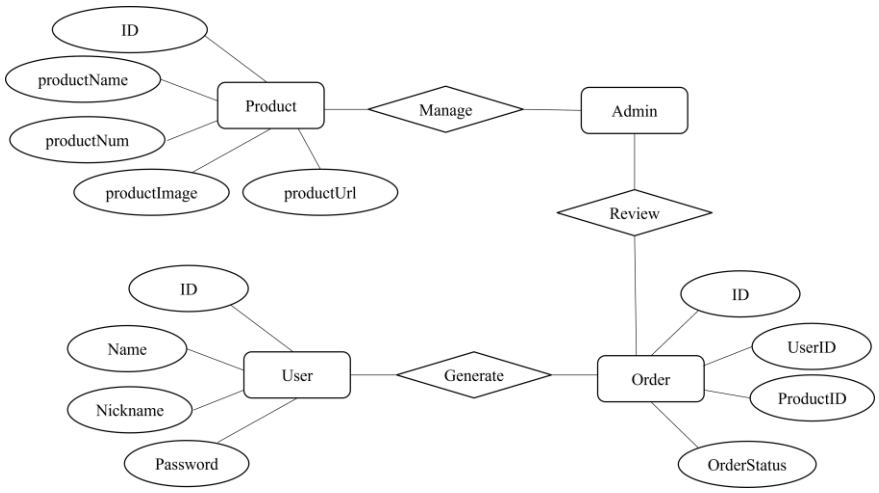


Рисунок 2.2 – Діаграма E-R

Створення інтерфейсу даних

Нам потрібно створити інтерфейс для розробки, а потім реалізувати відповідний інтерфейс для створення структури бази даних продукту. Ми додаємо такі функції, як вставка, видалення, оновлення та вибір, після підключення до баз даних, код показано на рисунок 2.3.

```
type IProduct interface {
  Insert(*models.Product) error
  Delete(int64) bool
  Update(*models.Product) error
  SelectByKey(int64) (*models.Product, error)
  SelectAll() ([]*models.Product, error)
  SubProductNum(productID int64) error
}
```

Рисунок 2.3– Інтерфейс продукту

Інтерфейс – це тип, визначений як набір сигнатур методів, що є узагальненням та абстракцією поведінки інших типів. Інтерфейс – це абстрактний тип, який не надає внутрішніх операцій на основі цього точного макета.

Особливості інтерфейсу можна підсумувати наступним чином:

- Інтерфейс має лише оголошення методів, не має реалізації та має поля даних.

- Інтерфейси можна анонімно вбудовувати в інші інтерфейси або структури.

Інтерфейс – це тип, що використовується для визначення поведінки. Ці визначені поведінки реалізуються не безпосередньо інтерфейсами, а визначеними користувачем типами, і конкретний тип, який реалізує ці методи, є екземпляром цього типу інтерфейсу.

Якщо користувацький тип реалізує набір методів, оголошених типом інтерфейсу, тоді значення користувацького типу можна присвоїти значенню типу інтерфейсу. Це присвоєння збереже значення користувацького типу в типі інтерфейсу.

Створіть функцію ініціалізації

Однак, реалізація інтерфейсу не зникне в мові програмування Go. Нам потрібно створити функцію-конструктор, щоб перевірити, чи реалізовано в коді потрібну структуру, як показано на рисунок 2.4 у функції NewProductManager.

```
1 func NewProductManager(table string,db *sql.DB) IProduct{
2   return &ProductManager{table: table,mysqlConn: db}
3 }
```

Рисунок 2.4 – Новий менеджер продукту

Створити логіку сервісу на основі структури операцій бази даних

Якщо ми хочемо розробити логіку служби бази даних, першим кроком є розробка відповідного інтерфейсу. Всередині інтерфейсу продукту ми підключаємося до бази даних і додаємо відповідні функції вставки, видалення, оновлення та вибору. Після визначення інтерфейсу нам потрібно його реалізувати, код показано на рисунок 2.5.

```go
package services

import (
  "product/data"
  "product/repositories"
)

type IProductService interface {
  GetProductByID(int64) (*data.Product,error)
  GetAll() ([]*data.Product,error)
  DeleteProductByID(int64) bool
  InsertProduct(product *data.Product) (int64,error)
  UpdateProduct(product *data.Product) error
  SubProductNum(productID int64) error
}

type ProductService struct {
  productRepos repositories.IProduct
}
```

Рисунок 2.5 – IProductService

Розробка контролера

Ми розробили моделі та представлення в архітектурі MVC на основі вищезазначеного. Тепер наступним кроком є розробка частини контролера. Після створення пакета контролера ми створюємо структуру ProductController, щоб розмістити iris та сервіс продукту всередину. Потім нам потрібно виконати дію структури, тобто отримати продукти для користувачів. Коли користувач відвідує наш веб-сайт, він чи вона, ймовірно, хоче перевірити детальну інформацію про продукт та зробити покупку. Потім має бути перерахована детальна інформація про продукт та функції. Усередині контролера повинні бути відображені представлення для реалізації вищезазначених потреб, код показано на рисунок 2.6.

```
type ProductController struct {
  Ctx iris.Context
  ProductService services.IProductService
}

func (p *ProductController) GetAll() mvc.View{
  products,_ := p.ProductService.GetAll()
  return mvc.View{
    Name: "product/view.html",
    Data: iris.Map{"productArray":products},
  }
}
```

Рисунок 2.6 – Отримати все

Зареєструйте контролер та підключити його до бази даних у backend/main.go, код показано на рисунок 2.7.

```
productRepository := repositories.NewProductManager("product",db)
productService := services.NewProductService(productRepository)
productParty := app.Party("/product")
product := mvc.New(productParty)
product.Register(ctx,productService)
product.Handle(new(controllers.ProductController))
```

Рисунок 2.7 – Реєстр продукції

Записати файл шаблону моделі MVC

Під час створення динамічного веб-сайту або показу користувачеві налаштованого виводу ми зазвичай використовуємо шаблон Golang для реалізації. Зазвичай Golang має два пакети шаблонів:

- text/template – він надає необхідні функції для розбору нашої програми.
- html/template – надає пакети для підтримки рендерингу шаблонів.

Зазвичай, обидва ці шаблони можуть генерувати однаковий інтерфейс,

тоді як шаблон html-l/template є безпечнішим, оскільки він реалізує шаблони на основі даних для генерації виводу HTML, код показано на рисунок 2.8.

```
return mvc.View{
    Name: "product/manager.html",
    Data: iris.Map{
        "product": product,
    },
}
```

Рисунок 2.8 – Представлення MVC

2.1.2 Розробка управління back-end замовленнями

У попередньому розділі ми вже продемонстрували, як розробити систему управління серверною частиною продукту. У цьому розділі ми продовжуємо розробку системи управління серверною частиною замовлень, яка подібна до системи управління серверною частиною продукту, дотримується архітектури MVC та має три частини, як показано нижче:

1. Розробка дизайну моделей замовлень.
2. Розробка функцій замовлень (з чотирма основними функціями: вставка, видалення, зміна та перевірка).
3. Інтерфейс замовленнями на back-end.

Створення моделі управління замовленнями в моделях даних

Ми створюємо файл models.go для кодування структури продукту після створення каталогу models. Призначаємо ідентифікатор замовлення, ідентифікатор користувача, ідентифікатор продукту та статус замовлення з типом int64, після чого частина моделей замовлення завершена, код показано на рисунок 2.9.

```
type Order struct {
  ID int64 `sql:"ID" order:"ID"`
  UserId int64 `sql:"userID" order:"UserID"`
  ProductId int64 `sql:"productID" order:"ProductId"`
  OrderStatus int64 `sql:"orderStatus" order:"OrderStatus"`
}
var(
  OrderSuccess int64 = 0
  OrderFail int64 = 1
)
```

Рисунок 2.9 – Модель даних замовленн

Створення структури операцій бази даних моделі замовлення

Розробка логіки обслуговування бази даних замовлень аналогічна тій, що ми робили з продуктом, і першим кроком для нас є розробка відповідного інтерфейсу. Всередині інтерфейсу продукту ми підключаємося до бази даних і додаємо відповідні функції вставки, видалення, оновлення та вибору. Після визначення інтерфейсу нам потрібно реалізувати його, код показано на рисунок 2.10.

```
type IOrderRepository interface{
    Insert(*models.Order) error
    SelectAllWithInfo() (map[int]map[string]string,error)
}

func NewOrderManagerRepository(table string) IOrderRepository{
  return &OrderManagerRepository{
    table: table,
  }
}

type OrderManagerRepository struct {
  table string
}
```

Рисунок 2.10 – Інтерфейс замовлення

Написати логіку сервісу на основі структури операцій бази даних

Нам потрібно реалізувати інтерфейс з відносними функціями на сервісі замовлень після створення функції конструктора замовлень. Спочатку нам потрібно підключитися до бази даних за допомогою функції Conn(). Потім ми виконаємо основні функції вставки, видалення, оновлення та вибору, код показано на рисунок 2.11.

```
func (o *OrderMangerRepository) Insert(order *models.Order) error {
    sql := "INSERT `order` set userID=?,productID=?,orderStatus=?"
    err := Product.Exec(sql, order.UserId, order.ProductId, order.
        OrderStatus).Error
    if err != nil {
        log.Log.Error(err)
        return err
    }
    return nil
}

func (o *OrderMangerRepository) SelectAllWithInfo() (OrderMap map[
    int]map[string]string, err error) {
    sql := "SELECT\n\to.ID,\n\tu.userName,\n\tp.productName,\n\to.
        orderStatus \nFROM\n\t`order` AS o\n\tLEFT JOIN product AS p
        ON o.productID = p.ID\n\tLEFT JOIN user AS u ON o.userID =
        u.ID"
    rows, err := Product.Raw(sql).Rows()
    if err != nil {
        return nil, err
    }
    return GetResultRows(rows), err
}
```

Рисунок 2.11 – Робота з базою даних замовлень

Розробка контролера

Розробка контролера замовлень подібна до розробки продукту. Після

створення пакета контролерів ми створюємо структуру OrderController, щоб розмістити контекст iris та сервіс продукту всередину. Потім нам потрібно виконати дію структури, тобто отримати замовлення для менеджерів. Усередині контролера повинні бути відображені представлення для реалізації вищезазначених потреб, код показано на рисунок 2.12.

```
type OrderController struct {
  Ctx          iris.Context
  OrderService services.IOrderService
}

func (o *OrderController) Get() mvc.View {
  orderArray, err := o.OrderService.GetAllOrderInfo()
  if err != nil {
    log.Log.Errorf("Failed to check orders info %s", err.Error())
    return mvc.View{}
  }
  return mvc.View{
    Name: "order/view.html",
    Data: iris.Map{
      "order": orderArray,
    },
  }
}
```

Рисунок 2.12– MVC порядку

Розробити реєстр контролера замовлень у функції main, код показано на рисунок 2.13.

```
  orderRepository := mysql.NewOrderMangerRepository("order")
  orderService := services.NewOrderService(orderRepository)
  orderParty := app.Party("/order")
  order := mvc.New(orderParty)
  order.Register(ctx, orderService)
  order.Handle(new(controllers.OrderController))
```

Рисунок 2.13 – Реєстр замовлень MVC

Шаблон замовлення розробляється так само, як і продукт, код показано на

рисунок 2.14.

```
return mvc.View{
  Name: "order/view.html",
  Data: iris.Map{
    "order": orderArray,
  },
}
```

Рисунок 2.14 – Шаблон замовлення

Наразі ми завершили розробку бекенд-керування продуктами та замовленнями. Після запуску backend/main.go ми зможемо побачити інтерфейс керування backend, код показано на рисунок 2.15.

```
addr := host + ":18080"
err := app.Run(
  iris.Addr(addr),
  iris.WithoutServerError(iris.ErrServerClosed),
  iris.WithOptimizations,
)
```

Рисунок 2.15– Виконано код backend

2.1.3 Розробка ключових функцій front-end

У цьому розділі ми починаємо розробку функцій фронтенду. Оскільки сторінки фронтенду призначені переважно для клієнтів, не слід враховувати менеджера порівняно з проектуванням інтерфейсу бекенду. Користувач є найважливішим учасником фронтенду, і з точки зору користувача, ми перераховуємо інтерфейс користувача та обробник на рисунку 2.16. Розробка наших основних функцій фронтенду складається з трьох частин, а саме:

1. Розробка інтерфейсу входу користувача
2. Розробка функцій відображення фронтенд-продуктів
3. Розробка контролю даних про покупки

Однак, це ще не все для нашого програмного забезпечення, яке не має характеристик високої продуктивності та здатності справлятися з великими потоками даних. Ми поговоримо про основну системну частину в наступному розділі, а зараз почнемо з базового інтерфейсу входу користувача.

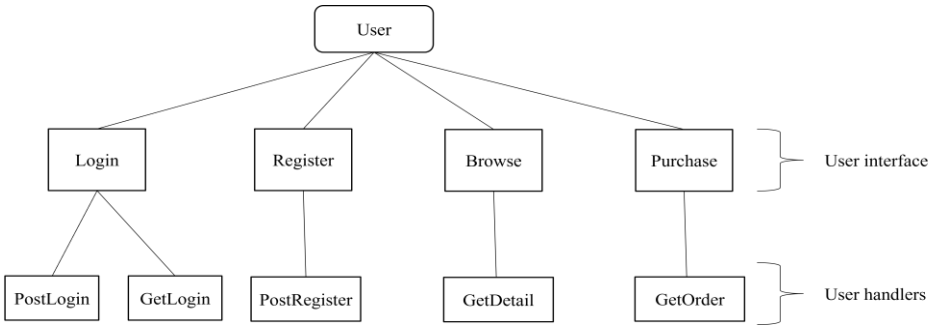


Рисунок 2.16– Інтерфейс та обробники користувача

Впровадження AES

У наступних розділах ми реалізуємо автентифікацію користувача для його автентифікації. Для реєстрації користувача, що увійшов у систему, буде використано механізм на основі файлів cookie, і оскільки файли cookie зберігаються на стороні клієнта, зловмисник може їх змінити. Тому перед реалізацією функції входу користувача необхідно реалізувати метод шифрування AES для запобігання атакам зловмисників.

Ми реалізували 128-бітний шаблон доповнення AES та метод шифрування/дешифрування. Спочатку ми визначаємо байт доповнення AES як довжину вихідного тексту, а потім повторюємо цей байт кілька разів, доки доповнений байт не буде успішно згруповано. Потім викликаємо інтерфейс AES для шифрування, а після цього додаємо шар шифрування base64 як зашифроване значення файлу cookie, що надсилається клієнту. Ми перераховуємо лише три функції шифрування з міркувань обсягу, а дешифрування – це зворотний процес до вищезазначених кроків.

Оскільки зловмисник не може знати ключ шифрування AES, він хоче видати себе за законного користувача за допомогою підробленого файлу cookie,

необхідного для перерахування зашифрованого рядка, а це дуже дорого. Ми надамо безпеку в наступному розділі тестування після використання шифрування AES, код показано на рисунок 2.17.

```
func pKCS7Padding(ciphertext []byte, blockSize int) []byte {
  padding := blockSize - len(ciphertext)%blockSize
  return append(ciphertext, bytes.Repeat([]byte{byte(padding)},
      padding)...)
}

func aesEcrypt(origData []byte, key []byte) ([]byte, error) {
  block, err := aes.NewCipher(key)
  if err != nil {
    return nil, err
  }
  blockSize := block.BlockSize()
  origData = pKCS7Padding(origData, blockSize)
  blocMode := cipher.NewCBCEncrypter(block, key[:blockSize])
  crypted := make([]byte, len(origData))
  blocMode.CryptBlocks(crypted, origData)
  return crypted, nil
}

func EnCode(pwd []byte) (string, error) {
  result, err := aesEcrypt(pwd, PwdKey)
  if err != nil {
    return "", err
  }
  return base64.StdEncoding.EncodeToString(result), err
}
```

Рисунок 2.17 – Впровадження AES

Основний процес реєстрації користувача

Інтерфейс входу користувача також відповідає архітектурі MVC, подібно до системи управління замовленнями продуктів. Детальний процес розробки MVC для входу користувача не буде представлено. Тепер почнемо з основного процесу реєстрації користувача.

Оскільки інтерфейс користувача відрізняється від сторінок бекенду з різними функціями, статичні веб-сторінки, шаблони та патерни, які ми використовуємо, також будуть різними. Ми кодуємо всі сторінки фронтенду в папку фронтенду, щоб відрізнити вищезгаданий пункт, тоді як функції бекенду знаходяться всередині папки бекенду, код показано на рисунок 2.18.

```
func (c *UserController) GetRegister() mvc.View {
    return mvc.View{
        Name: "user/register.html",
    }
}
```

Рисунок 2.18 – MVC регістру користувача

Ми розміщуємо всі статичні ресурси в папку public, де зберігаються CSS, шрифти, js та зображення. Крім того, нам потрібна папка views для зберігання файлів шаблонів та HTML-файлів. Нам потрібно підготувати форму реєстрації та створити новий контролер типу Get. Нам потрібно зареєструвати користувача за допомогою функції register, і інтерфейс буде відображатися на сторінці register.html після створення контролера користувача. Потім ми створюємо fronted/main.go для запуску сторінок fronted.

Потім ми створюємо контролер форми, що приймає запити, а назва методу контролера має починатися з Post. Після завершення login.html та register.html ми можемо отримати інтерфейс входу, як показано нижче. Однак нам потрібно обробити запит post у функції PostRegister(). Діаграма логіки виконання register показанана рисунок 2.19.

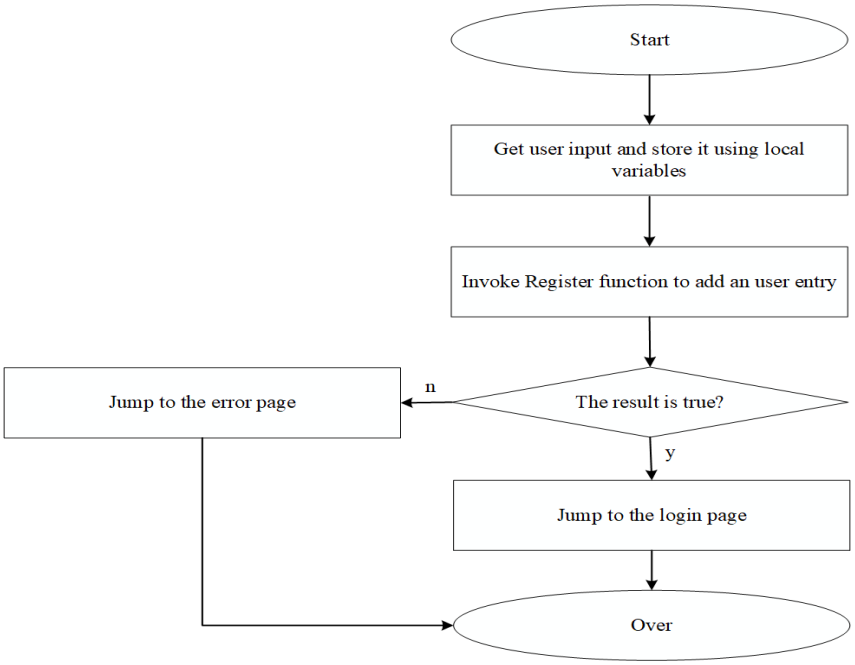


Рисунок 2.19 – Схема логіки виконання регістра

Ми отримуємо вміст форми та створюємо екземпляр структури користувача після вищезазначених операцій, код показано на рисунок 2.20.

```
var (
    nickName = c.Ctx.FormValue("nickName")
    userName = c.Ctx.FormValue("userName")
    password = c.Ctx.FormValue("password")
)
user := &models.User{
    UserName:     userName,
    NickName:     nickName,
    HashPassword: password,
}
```

Рисунок 2.20– Отримання вмісту форми

Ми вже розробили моделі користувачів, зіставили зв'язки та сервіси. Розробка бази даних ще не завершена, але метод схожий на продукт та замовлення. Ми зашифруємо пароль під час реєстрації, код показано на рисунок

2.21.

```
func GeneratePassword(userPassword string) ([]byte,error){
    return bcrypt.GenerateFromPassword([]byte(userPassword),bcrypt.
        DefaultCost)
}
```

Рисунок 2.21 – Отримати зашифрований пароль

Процес входу користувача в систему

Процес входу користувача можна завершити наступними чотирма кроками: Першим кроком буде відображення сторінки входу користувача. Потім буде запущено контролер користувача для обробки запиту користувача. На третьому кроці контролер перевірить введені дані, включаючи ім'я користувача та пароль. Нарешті, контролер перейде до користувача на основі результату автентифікації. Якщо користувач увійшов у систему, інформація про користувача буде зашифрована та збережена в cookie для повернення та переходу на сторінку покупки. Якщо користувач не зможе увійти, його буде перенаправлено на сторінку реєстрації, якщо ідентифікатор користувача дорівнює 0, або на сторінку помилки, якщо пароль користувача не дорівнює паролю, збереженому в базі даних, код показано на рисунок 2.22. Повна схема логіки виконання показана на рисунок 2.23.

```
func (c *UserController) GetLogin() mvc.View {
    return mvc.View{
        Name: "user/login.html",
    }
}
```

Рисунок 2.22 – MVC входу

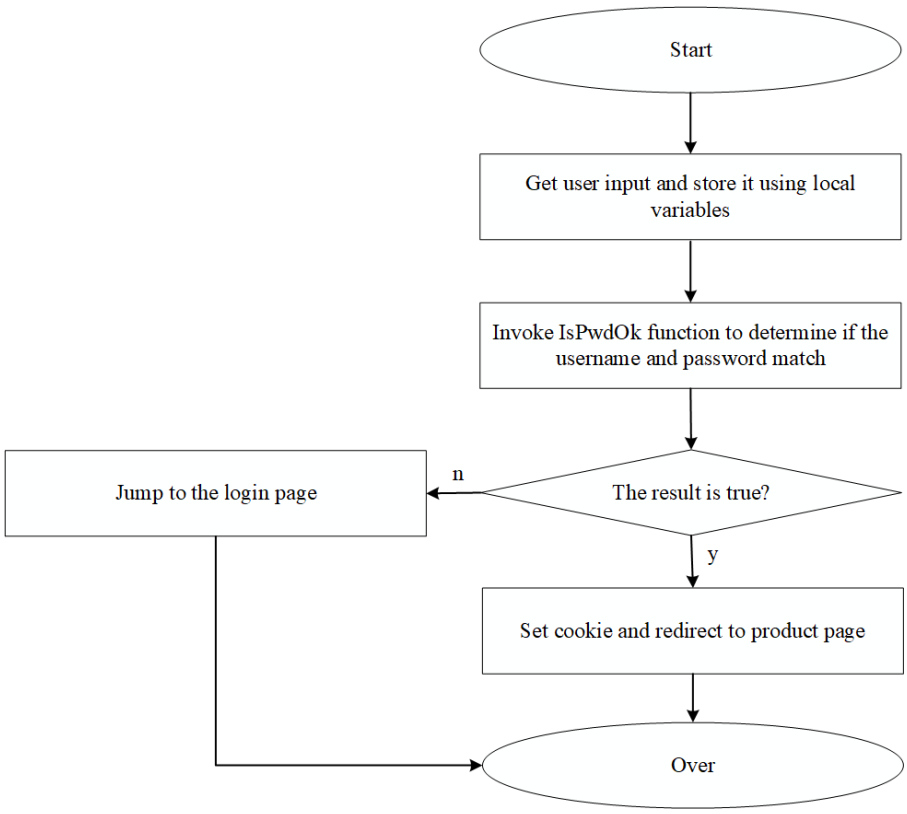


Рисунок 2.23 – Діаграма логіки виконання входу в систему

Відображення деталей продукту

Ми можемо використовувати бекенд-сервіси для розробки інтерфейсу обробки деталей продукту на основі попередньої розробки бекенд-управління. Аналогічно, розробка має три основні частини, а саме:

1. Запит інформації на основі продукту-послуги.
2. Розробка контролера продукту.
3. Розробка інтерфейсу відображення на фронтальній стороні продукту.

По-перше, нам потрібно створити контролер продукту всередині папки fronted/web/controllers. Нам потрібен контекст Iris та узгодження служби продукту з попереднім інтерфейсом. Крім того, нам потрібен тимчасовий сеанс для запуску програми. Однак, цей сеанс створюватиме високе навантаження на сервер і буде складним для сервера в управлінні, що не дозволить досягти наших високих поточних вимог до програмного забезпечення. Отже, ми змінимо та оновимо сеанс для cookie в наступному розділі.

Після розробки бази даних нам потрібно запитувати інформацію про наш продукт з бази даних за допомогою функції запиту.

2.1.4 Логічне керування кількісними параметрами

Нам потрібно розробити систему контролю даних про покупки після розробки детальної інформації про продукт, що є однією з основних функцій нашого програмного забезпечення та складається з трьох основних частин:

1. Отримання інформації про продукт
2. Перетворення ідентифікатора продукту
3. Запит щодо продуктів

По-перше, нам потрібно отримати ідентифікатор продукту з параметра URL-адреси та призначити його нашій програмі. По-друге, ми використовуємо файл cookie для отримання ідентифікатора користувача. Крім того, нам потрібно виконати перетворення ідентифікатора продукту для досягнення вищезазначених потреб. Після отримання ідентифікатора продукту та користувача нам потрібно запитати інформацію про продукт, щоб перевірити, чи є у нас ще достатньо продуктів на складі через службу підтримки продуктів.

Тепер перевіримо кількість продуктів, щоб побачити, чи відповідає вона вимогам продажу. Якщо номер продукту більший за нуль, користувач зможе здійснити покупку. Щоразу, коли користувач успішно купує продукт, ми віднімаємо кількість продукту на одиницю та оновлюємо кількість у базі даних. Аналогічно, якщо ми створюємо замовлення, кількість замовлень збільшується.

Redis для управління номерами продуктів

Redis — це високопродуктивна база даних типу «ключ-значення», яка має такі переваги порівняно з іншими базами даних.

1. Redis має дуже високу продуктивність, а швидкість читання та запису краща, ніж в інших баз даних.

2. Redis підтримує багаті структури даних, такі як множини, рядки та списки.
3. Операції з базою даних у Redis є атомарними.

Ми використовуємо Redis для кешування даних продукту, щоб досягти високопродуктивної оцінки, коли користувачі отримують продукт. Ми реалізуємо операцію додавання та видалення ключів для Redis. Ми визначаємо глобальну змінну підключення Redis для Redis, а потім створюємо функцію ініціалізації для виконання тестів підключення на Redis. Далі ми створюємо функції SetCache, DelCache та GetCache для певного ключа наступним чином.

Зверніть увагу, що в Redis ефект додавання та оновлення однаковий, тому SetCache використовується однаково для реалізації.

Діаграма логіки виконання для нашої системи закупівель з високим рівнем паралельності, що використовує Redis, показана на рисунок 2.24.

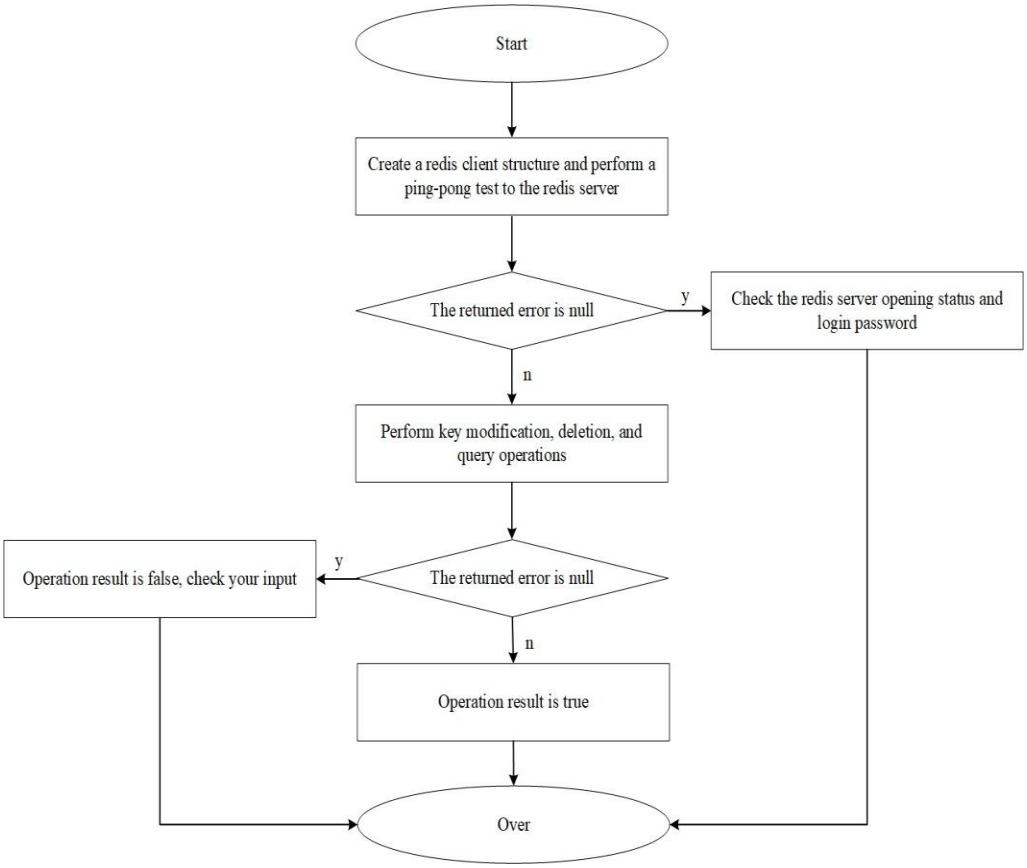


Рисунок 2.24 – Діаграма логіки виконання redis

2.2 Оптимізація для системи закупівель з високим рівнем паралельності

2.2.1 Оптимізація front-end

Під час розробки фронтенд-частини ми вже проаналізували архітектуру основних функцій на рисунок 2.25:

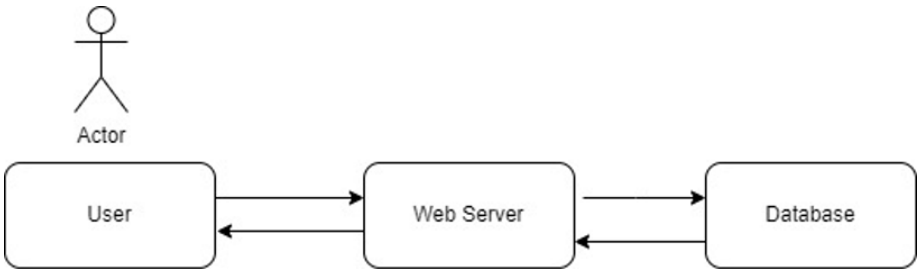


Рисунок 2.25 – вибір

Коли користувач відвідує наш веб-сайт, він спочатку потрапляє на наш веб-сервер, де міститься вся інформація про продукт. Усередині веб-сервера (сторінка з детальною інформацією про продукт) функції запитів тісно пов'язані з базами даних. Після того, як веб-сервер взаємодіє з базою даних, база даних отримує детальну інформацію про продукт, а потім надсилає дані назад на веб-сервер для відображення HTML-сторінки для користувачів.

Оптимізуйте архітектуру

Однак, ця проста фундаментальна архітектура не може задовольнити потреби високострумової системи нашого програмного забезпечення. Основні проблеми будуть продемонстровані наступним чином:

1. Веб-сервер матиме високе навантаження, а вартість перевірки безпеки буде високою. Серверу потрібно обробляти велику кількість запитів користувачів та виконувати дії для перевірки безпеки. Коли всі дії потрібно виконувати на одному сервері, високе навантаження збільшує фінансові витрати.

2. Отримання даних у режимі реального часу призведе до високого навантаження на бази даних. Веб-сервер отримає один запит, коли один користувач виконує дії на сторінці інтерфейсу. Коли користувачів багато, наприклад, 10000 відвідувачів нашої веб-сторінки та виконують відповідні операції, сервер отримає 10000 запитів та виконає 10000 запитів.
3. Важко підтримувати узгодженість даних у ситуаціях високого струму. У ситуаціях високого струму зазвичай виникає проблема перепроданості продуктів, і нам доводиться виконувати дії з кожним вузлом тиску.

Ми будуємо наступну архітектуру, як показано нижче, щоб вирішити вищезазначені проблеми.

1. Коли користувач відвідує нашу систему, він відвідує всі статичні ресурси, де знаходяться js, CSS, зображення продуктів та HTML-файли.
2. Коли користувач хоче придбати продукт, і після натискання кнопки «Купити», дані в режимі реального часу надсилаються до SLB (балансувальник навантаження сервера). SLB розподіляє запити від веб-серверів.
3. Розподілена перевірка безпеки в основному використовується для перевірки перехоплення даних на дійсність запиту. Тим часом розподілена система може виконувати горизонтальне масштабування.
4. Після розподіленої перевірки буде здійснюватися контроль кількості закуплених товарів, щоб уникнути проблем перепроданості. Водночас це також покращить продуктивність програмного забезпечення.
5. Коли контроль кількості закуплених товарів отримає запит на придбання товару, він надішле цей запит на веб-сервер.
6. Веб-сервер запише повідомлення про покупку на RabbitMQ після отримання запиту.
7. RabbitMQ – це брокер повідомлень. Коли виникає трафік даних,

MySQL отримає кілька запитів і буде зайнятий. RabbitMQ служитиме проміжним програмним забезпеченням для зберігання всіх запитів, допомагаючи зменшити навантаження на базу даних. Схема показана на рисунок 2.26.

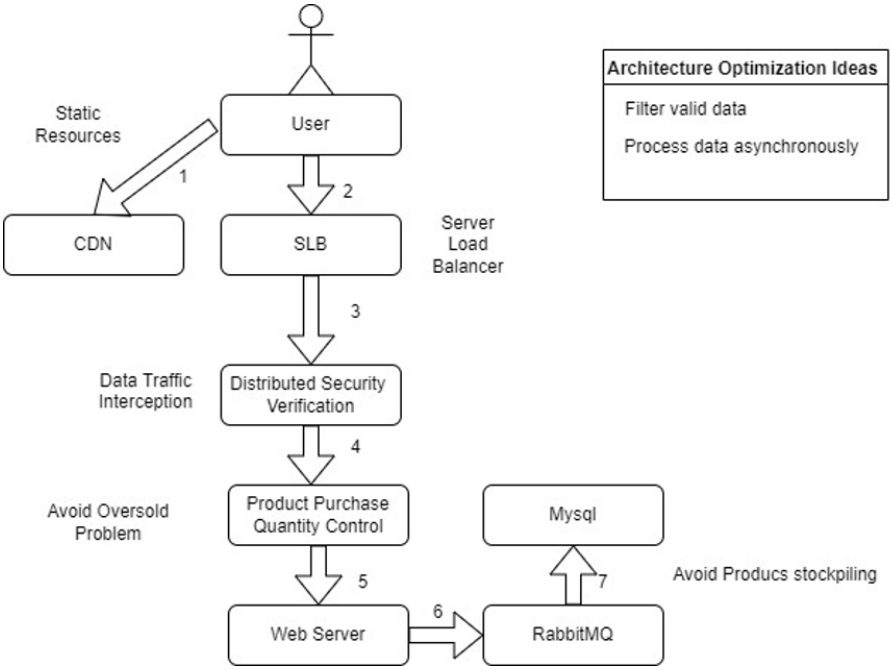


Рисунок 2.26 – система

Дизайн та розробка статичних сторінок

Ми можемо відображати наші сторінки відповідно до запитів після того, як веб-сторінка розроблена статичним методом. Відображення статичних веб-сторінок буде набагато швидшим, ніж динамічних. Динамічний веб-сайт завжди змінюватиметься та відображатиме різні інтерфейси щодо даних у режимі реального часу. Щоразу, коли ми натискаємо на посилання або відвідуємо динамічний веб-сайт, сервер перевірятиме серію динамічних даних та виконуватиме відносні логічні оцінки, щоб перевірити, чи є логіка сервісу законною. Якщо вона законна та відповідає нашим сервісам та логіці, сторінки повинні виконувати рендеринг, щоб реагувати на цю логіку та зрештою відображати відносні сторінки. Для динамічних сторінок, якщо функції запитів та логічні оцінки недостатньо добре розроблені або веб-сторінки отримують

забагато відвідувань, поведінка сервера буде неочікуваною, навіть закриватиметься, автоматично вимикатиметься або аварійно завершуватиме роботу. Однак статична веб-сторінка зменшить такі негативні наслідки та підвищить швидкість і ефективність.[23]

Оскільки динамічним веб-сторінкам потрібно запитувати серію даних, що створюватиме велике навантаження, навіть якщо використання кешу або Redis не дуже допоможе, відображення лише статичної веб-сторінки буде порівняно витрачати трохи ресурсів та зменшувати робоче навантаження на сервер.

У більшості випадків статичні сайти набагато безпечніші, ніж динамічні. Оскільки динамічні сайти, як правило, більше піддаються впливу Інтернету, і кожен запит від сервера повинен взаємодіяти з програмою та базою даних, неочікуваний шкідливий запит може спричинити незліченну кількість помилок або збоїв.

Зазвичай існує кілька способів розробки статичних веб-сайтів, таких як використання генератора статичних сайтів, купівля домену та розгортання. Замість цього ми вирішуємо реалізувати це за допомогою кодування.

По-перше, нам потрібно створити каталоги для зберігання згенерованого HTML-файлу та шаблонів статичних файлів. Потім нам потрібна функція для генерації статичних HTML-файлів. Нам все ще потрібен контролер для генерації HTML після створення функції HTML. Аналогічно, нам потрібно правильно обробити файли та шлях до шаблону та впоратися з помилкою.

Ми використовуємо функцію ParseFiles для генерації шляху до шаблону всередині функції GetGenerateHtml. Крім того, слід також вставити згенерований шлях до HTML-файлу. Після встановлення шляху нам потрібно виконати рендеринг даних шаблону. Останній крок – генерація статичних файлів із призначеними необхідними параметрами.

Контроль безпеки Javascript

У попередніх розділах ми реалізували оптимізацію фронтального веб-

сайту. Тепер у цьому розділі ми розглянемо керування потоком даних фронтального JS.

Багато користувачів одночасно відвідуватимуть наші веб-сайти в режимі реального часу для нашого програмного забезпечення. Якщо ми не встановимо обмеження на потік даних, веб-сервер отримуватиме запит щоразу, коли користувач відвідує наш веб-сайт. Аналогічно, веб-сервер надсилатиме запит до бази даних. Вищезазначена серія впливів призведе до високого навантаження на бази даних та призведе до неочікуваних втрат ресурсів, що вплине на стабільність наших веб-сайтів та додатків. Якщо ми встановимо обмеження потоку даних пошарово, з фронтального веб-сайту на сервер надсилатимуться лише дійсні дані, і відповідно, дійсні запити будуть записані в базу даних. Зрештою, база даних отримуватиме рахунково доступні запити та підтримуватиме стабільну взаємодію із сервером.

Загальний контроль безпеки Javascript можна перерахувати наступним чином:

- JS встановлює обмеження, згідно з яким користувач може надіслати запит лише протягом десяти секунд. Якщо користувач уже надіслав запит на покупку, успішно чи ні, він/вона може чекати лише десять секунд на наступну операцію.
- JS встановив обмеження для користувачів, які не авторизувалися. Тільки наші учасники можуть працювати на наших веб-сайтах, оскільки ми отримуємо такі запити від ідентифікатора користувача, і це обмеження також дозволить уникнути недійсних запитів.

Ми зможемо запустити фронтенд-сторінку після налаштування статичних файлів та каталогів ресурсів. Однак нам слід встановити параметри сторінки за допомогою скрипта у файлі htmlProduct.html. Потім нам потрібно написати код функції rush to buy для користувача з відповідними параметрами та взаємодією з кнопкою.

Коли користувач поспішає купити товар і натискає кнопку, має бути таймер, який обмежуватиме його дії. Користувач зможе виконати наступну

операцію через десять секунд після останньої операції, якщо він успішно придбає товар, код показано на рисунок 2.27.

```
function timeSub(){
    inter = setInterval("timeFunc()",1000)
}
function timeFunc(){
    count--;
    if (count<=0){
        count = interval
        document.getElementById(BuyButtonID).removeAttribute("
            disabled")
        document.getElementById(BuyButtonID).value="Buy now!";
        clearInterval(inter);
    }else{
        document.getElementById(BuyButtonID).value="waiting"+
            count+"seconds to buy";
    }
}
```

Рисунок 2.27 – Код контролю безпеки JS

Нам потрібно надсилати асинхронні запити після виконання функції timeSub(). По-перше, нам потрібно створити об'єкт движка ajax, який підтримуватиме отримання запитів від IE7+, Firefox, Chrome, Opera та Safari. По-друге, ми можемо прив'язати подію listener до об'єкта движка.

2.2.2 Оптимізація Back-end

Узгоджений хеш

Ми розробляємо систему з високою продуктивністю паралельних закупівель для підтримки кількох машин, і, розгортаючи кілька машин, ми можемо розширити продуктивність обробки всієї системи горизонтально. Усі розподілені машини становитимуть кластер розподіленого сховища. Для розподіленого сховища дані різних об'єктів зберігаються на різних машинах, тому ми використовуємо послідовне хешування для встановлення зв'язку між даними та серверами. Алгоритм послідовного хешування гарантує, що при збільшенні або зменшенні кількості машин міграція даних між вузлами

обмежується лише двома вузлами та не викликає глобальних проблем з мережею.

Алгоритм хешування хешує відповідний ключ у простір з кількістю сегментів, що в 2^{32} рази перевищує кількість, та числовим простором від 0 до $2^{32} - 1$. Тепер ми можемо з'єднати ці числа голова до хвоста, утворюючи замкнутий цикл.

Оскільки кількість машин скінченна і не повністю заповнює все хеш-кільце, ми створюємо віртуальні вузли для кожної машини та розподіляємо їх по хеш-кільцю, щоб запобігти перевантаженню машин.

Реалізація узгодженого хешування

Спочатку нам потрібно визначити хеш-структуру, яка включає хеш-кільце, структуру даних для зберігання відсортованих ключів, кількість віртуальних вузлів та блокування читання/запису для керування доступом з кількох машин. Потім створити функцію ініціалізації NewConsistent, яка повертає хеш-структуру типу вказівника, код показано на рисунок 2.28.

```
type Consistent struct {
  circle map[uint32]string
  sortedHashes hashes
  virtualNodeNums int
  sync.RWMutex
}

func NewConsistent() *Consistent{
  return &Consistent{
    circle: make(map[uint32]string),
    virtualNodeNums: 20,
  }
}
```

Рисунок 2.28– Визначення узгодженого хеш

По-друге, ми генеруємо відповідний ключ для кожного віртуального

вузла та визначаємо гарну хеш-функцію, взявши за приклад 127.0.0.1, тоді її ключ для кожного віртуального вузла дорівнює 127.0.0.1-1, символ наступного є номером цього віртуального вузла. Потім ми визначаємо функцію hashKey для хешування ключів, щоб вони були зіставлені з хеш-кільцем, код показано на рисунок 2.29.

```
func (c *Consistent) generateVirtualNodes(s string,index int)
    string{
  return s + "-" + strconv.Itoa(index)
}

func (c *Consistent) hashKey(key string) uint32{
  if len(key) < 64{
    var extendKey [64]byte
    copy(extendKey[:],key)
    return crc32.ChecksumIEEE(extendKey[:len(key)])
  }
  return crc32.ChecksumIEEE([]byte(key))
}
```

Рисунок 2.29 – хеш-функція консистентного хешування

По-третє, налаштуйте функцію додавання та видалення вузлів для хеш-кільця, функція якої полягає в адаптації до потреб додавання та видалення серверів у розподілених сценаріях. Додавання та видалення певного сервера на кільці здійснюється за допомогою виклику функцій додавання та видалення, код показано на рисунок 2.30.

```
func (c *Consistent) add(s string){
  for i := 0; i < c.virtualNodeNums; i++{
    c.circle[c.hashKey(c.generateKey(s,i))] = s
  }
  c.updateSortedHashes()
}
func (c *Consistent) remove(s string){
  for i := 0;i<c.virtualNodeNums;i++{
    delete(c.circle,c.hashKey(c.generateKey(s,i)))
  }
  c.updateSortedHashes()
}
```

Рисунок 2.30 – Додавання та видалення для узгодженого хешування

По-четверте, потрібно реалізувати функцію оновлення, яка використовується для сортування існуючих ключів у хеш-кільці. Запит генеруватиме хеш-значення після хешування та знаходитиме вузол сервера, до якого потрібно отримати доступ через цей відсортований хеш-ключ, код показано на рисунок 2.31.

```
func (c *Consistent) updateSortedHashes(){
  var hashes hashes
  for k := range c.circle{
    hashes = append(hashes,k)
  }
  sort.Sort(hashes)
  c.sortedHashes = hashes
}
```

Рисунок 2.31 – Оновлення для забезпечення узгодженого хешування

По-п'яте, ми реалізуємо функцію пошуку, яка знаходить найближчий вузол сервера за годинниковою стрілкою з sortedHashes після певного хеш-ключа, код показано на рисунок 2.32.

```
func (c *Consistent)search(key uint32) int{
  f := func(x int) bool{
    return c.sortedHashes[x] > key
  }
  i := sort.Search(len(c.sortedHashes),f)
  if i >= len(c.sortedHashes){
    i = 0
  }
  return i
}
```

Рисунок 2.32– пошук узгодженого хешу

Нарешті, ми реалізуємо функцію get, яка обгортає функцію пошуку та повертає інформацію про найближчий сервер для ключа, код показано на рисунок 2.33.

```
func (c *Consistent) Get(name string) (string,error){
  c.RLock()
  defer c.RUnlock()
  if len(c.circle) == 0{
    return "",noDataErrorMsg
  }
  key := c.hashKey(name)
  index := c.search(key)
  return c.circle[c.sortedHashes[index]],nil
}
```

Рисунок 2.33 – отримати для узгодженого хешування

Використання RWlock

Кілька потоків можуть одночасно використовувати ресурс у середовищі з високою конкуренцією. Такі ресурси називаються спільними ресурсами, а багатопотоковий доступ до спільних ресурсів може призвести до невідповідності даних. Тому для захисту спільного ресурсу вводяться блокування, щоб лише один потік міг одночасно отримати доступ до спільного ресурсу, забезпечуючи таким чином узгодженість даних.

Реалізація RWlock

Блокування реалізовано в пакеті sync у мові програмування Go. Спочатку нам потрібно визначити блокування типу sync для блокування спільного ресурсу. Потім Mutex викликає методи Lock та Unlock для завершення доступу та розблокування спільного ресурсу, використовуючи defer для автоматичного розблокування даних в кінці функції. Інші потоки не можуть отримати доступ до спільного ресурсу, доки він не буде розблоковано, щоб забезпечити узгодженість даних, код показано на рисунок 2.34.

```
import "sync"

var mutex sync.Mutex
mutex.Lock()
defer mutex.Unlock()
```

Рисунок 2.34 – отримати для узгодженого хешування

Використання файлів cookie

Ми використовуємо файли cookie як токен ідентифікації користувача для автентифікації клієнта та запобігання доступу неавторизованих користувачів до системи. Файли cookie мають такі переваги:

Файл cookie зберігається на стороні клієнта, і клієнт розміщує файл cookie в запиті для кожного запиту, щоб позначити користувача. Порівняно з сеансами, файли cookie легше впроваджувати.

Файл cookie може зменшити навантаження на сервер, оскільки він зберігається на стороні клієнта, а розмір його сховища обмежений.

Впровадження файлів cookie

Файли cookie можна реалізувати за допомогою пакета контексту в Go. Ми встановлюємо http.cookie для вказаної пари ключ-значення за допомогою функції Setcookie в пакеті контексту.

Потім ми змінюємо функцію входу користувача для автентифікації користувача. Спочатку ми отримуємо номер облікового запису та пароль користувача, а потім отримуємо результати, порівнюючи їх. Якщо користувач проходить автентифікацію, додається файл cookie з назвою auth, щоб ідентифікувати його як легітимного користувача. Потім користувача перенаправляють на сторінку з детальною інформацією про продукт, код показано на рисунок 2.35.

```
func (u *UserController) PostLogin() mvc.Response{
  userName := u.Ctx.FormValue("userName")
  passWord := u.Ctx.FormValue("passWord")
  user, ok := u.Service.IsPwdEqual(userName,passWord)
  if !ok{
    return mvc.Response{
      Path: "/user/login",
    }
  }
  tool.SetGlobalCookie(u.Ctx,"id",strconv.FormatInt(user.ID,10))
  idByte := []byte(strconv.FormatInt(user.ID,10))
  idString,_ := tool.StrEncode(idByte)
  tool.SetGlobalCookie(u.Ctx,"auth",idString)
  return mvc.Response{
    Path: "/product",
  }
}
```

Рисунок 2.35 – Реалізація файлу cookie

2.3 Тестування

Наш додаток буде безпечним та надійним після повного тестування. Мовою програмування Go автоматизоване тестування може бути дуже корисним для пошуку помилок. Наш тест продовжуватиме перевіряти оновлений код після додавання нового коду та видалення старого коду. Ми використовуватимемо інструмент go test для перевірки нашого додатку в нашому додатку. Крім того, ми проведемо простий ручний тест безпеки. Загалом, ми використовуватимемо чотирирівневі тести для демонстрації надійності та безпеки системи закупівель з високим рівнем паралельності наступним чином:

1. Модульні тести - вони перевірять деякі частини додатку незалежно.
2. Тести безпеки - вони перевірять ефективність наших заходів безпеки.
3. Функціональні тести - вони перевірять роботу нашого додатку з точки зору користувача.
4. Тести доступності - вони перевірять наш додаток закупівель з

високим рівнем паралельності, щоб побачити, чи працює він в умовах високого трафіку.

2.3.1 Плани тестування

У тесті безпеки ми спочатку перевірили алгоритм шифрування AES системи, щоб перевірити, чи може зловмисник підробити результат шифрування AES для проходження системної перевірки. По-друге, ми перевірили механізм файлів cookie, який використовується, щоб побачити, чи може користувач, який не ввійшов у систему, безпосередньо отримати продукт.

У функціональних тестах ми моделюємо доступ до URL-адрес за допомогою інструмента curl, щоб перевірити, чи отримано відповідні результати. Оскільки доступ до однієї URL-адреси активує рішення програми, функціональні тести можуть перевірити, чи вся система працює правильно.

У тесті доступності ми проводимо стрес-тест на зовнішньо відкритому порту robo-call нашої системи закупівель з високим рівнем паралельного трафіку через wrk, щоб перевірити зручність використання нашої системи в роботі з високим рівнем паралельного трафіку та довести, що наша система може витримувати тиск високого одночасного та інтенсивного трафіку після використання RabbitMQ та послідовного хешування.

2.3.2 Письмові тести

Модульні тести

Модульний тест в основному перевіряє функції замовлень, користувачів та продуктів у системі. Тут, оскільки ці три функції дуже схожі, ми обрали замовлення як приклад для тестування. Ми пишемо наступні чотири функції для тестування додавання, видалення та перевірки замовлень, код показано на рисунки 2.36, 2.37, 2.38 і 2.39.

```go
func TestOrderService_GetOrderByID(t *testing.T) {
  db,_ := common.NewMysqlConn()
  order := repositories.NewOrderManagerRepository("order",db)
  orderService := NewOrderService(order)
  if orderService == nil{
    t.Errorf("Get null!")
  }
  if _,err := orderService.GetOrderByID(int64(1)); err !=nil{
    t.Errorf("Get null!")
  }
}
```

Рисунок 2.36 – Перевірка на GetOrderByID

```go
func TestOrderService_DeleteOrderByID(t *testing.T) {
  db,_ := common.NewMysqlConn()
  order := repositories.NewOrderManagerRepository("order",db)
  orderService := NewOrderService(order)
  if orderService == nil{
    t.Errorf("Get null!")
  }
  if err := orderService.DeleteOrderByID(int64(1)); err == false{
    t.Errorf("Delete order false!")
  }
}
```

Рисунок 2.37 – Перевірка на DeleteOrderByID

```go
func TestOrderService_UpdateOrder(t *testing.T) {
  db,_ := common.NewMysqlConn()
  order := repositories.NewOrderManagerRepository("order",db)
  orderService := NewOrderService(order)
  if orderService == nil{
    t.Errorf("Get null!")
```

```
  }
  testOrder := &data.Order{int64(1),int64(1),int64(1),0}
  if err := orderService.UpdateOrder(testOrder); err != nil{
    t.Errorf("Update order false!")
  }
}
```

Рисунок 2.38 – Перевірка на UpdateOrder

```
func TestOrderService_InsertOrder(t *testing.T) {
  db,_ := common.NewMysqlConn()
  order := repositories.NewOrderManagerRepository("order",db)
  orderService := NewOrderService(order)
  if orderService == nil{
    t.Errorf("Get null!")
  }
  testOrder := &data.Order{int64(2),int64(1),int64(1),0}
  if _,err := orderService.InsertOrder(testOrder); err != nil{
    t.Errorf("Insert order false!")
  }
}
```

Рисунок 2.39 – Перевірка на наявність InsertOrder

Окрім вищезгаданого модульного тесту, ми також перевірили окрему функцію get IP, код якої показано на рисунок 2.40.

```
func TestGetLocalIp(t *testing.T) {
  if ans,_ := GetLocalIp(); ans != "127.0.0.1" {
    t.Errorf("Get %s wrong", ans)
  }
  if ans,_ := GetLocalIp(); ans == "127.0.0.2" {
    t.Errorf("Get %s wrong", ans)
  }
}
```

Рисунок 2.40: Перевірка IP-адреси

Тести безпеки

Ми перевіримо два аспекти нашої системи з високим рівнем

паралельності. По-перше, ми використовуємо порожній файл cookie, щоб імітувати неавторизованого користувача з прямим доступом.

У нашій системі, якщо система переходить безпосередньо на сторінку входу, це означає, що вона пройшла тест. Більше того, якщо доступ успішний, це означає, що тест не пройшовся. По-друге, ми використовуємо модифікований файл cookie, щоб імітувати доступ зловмисника до системи. За допомогою алгоритму шифрування AES наша система перевіряє значення файлу cookie, модифікованого зловмисником, щоб визначити, що поточний користувач намагається підробити файл cookie, щоб обійти автентифікацію.

Як показано на рисунках 2.41 та 2.42, ми тестуємо це, створюючи вікно безпосередньо в браузері, яке не містить інформації про файли cookie, та вікно, яке містить модифіковану інформацію про файли cookie, та отримуючи до них окремий доступ.

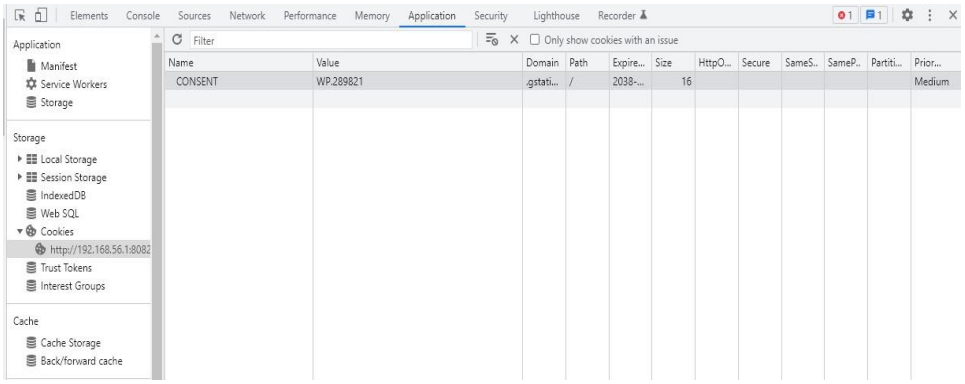


Рисунок 2.41 – Немає файлу cookie автентифікації

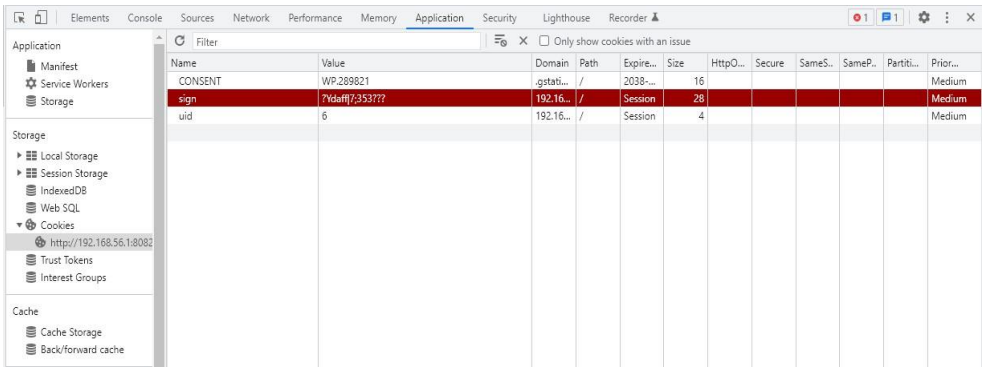


Рисунок 2.42 – Змінений файл cookie автентифікації

Функціональні тести

Ми тестуємо інтерфейс getOne та записи, згенеровані RabbitMQ, у функціональних тестах. Запит інтерфейсу getOne доступний безпосередньо за допомогою інструмента curl, перший – через таку команду.

curl http://127.0.0.1:8084/getOne

Потім ми створимо запит за допомогою curl і перевіримо, чи RabbitMQ прийме запит.

Тести доступності

Ми надали наші тестові випадки за допомогою інструменту wrk наступним чином: параметр t представляє кількість потоків, параметр c представляє кількість встановлених з'єднань, -d представляє тривалість стрес-тесту, а –latency представляє затримку виводу. Ми можемо змінити ці параметри, щоб збільшити навантаження на систему для перевірки її роботи.

wrk -t40 -c300 -d10s –latency "http://127.0.0.1:8084/getOne"

2.3.3 Результати тестування

Модульні тести

Виконайте команду go test -v, щоб ми могли протестувати файл у відповідному тестовому каталозі. Ось результати тестування функції отримання порядку за ідентифікатором та результати тестування отримання локальної IP-адреси, результат показано на рисунок 2.43 та 2.44.

```
[→  services git:(main) × go test -v
=== RUN   TestOrderService_GetOrderByID
--- PASS: TestOrderService_GetOrderByID (0.00s)
PASS
ok      product/services        0.009s
```

Рисунок 2.43 – Перевірка порядку

```
[➜  common git:(main) ✗ go test -v
=== RUN   TestGetLocalIp
--- PASS: TestGetLocalIp (0.00s)
PASS
ok      product/common  0.008s
```

Рисунок 2.44 – Перевірка на getLocalIP

Тести безпеки

Результатом перевірки безпеки є те, що зловмисник автоматично перейде на цільову сторінку, коли відвідає сторінку з порожнім та зміненим файлом cookie, це пов'язано з тим, що наша перевірка файлів cookie працює.

Функціональні тести

Ми провели функціональні тести getOne, протестували RabbitMQ і виявили, що getOne повертає результат "невдалий пошук" без доступу до файлів cookie. Після двох кліків для покупки через браузер, RabbitMQ отримав два запити на додавання замовлення та чекав на його використання, результат показано на рисунок 2.45 і 2.46.

```
[➜  ~ curl http://127.0.0.1:8084/getOne
false%
```

Рисунок 2.45 – Результат curl

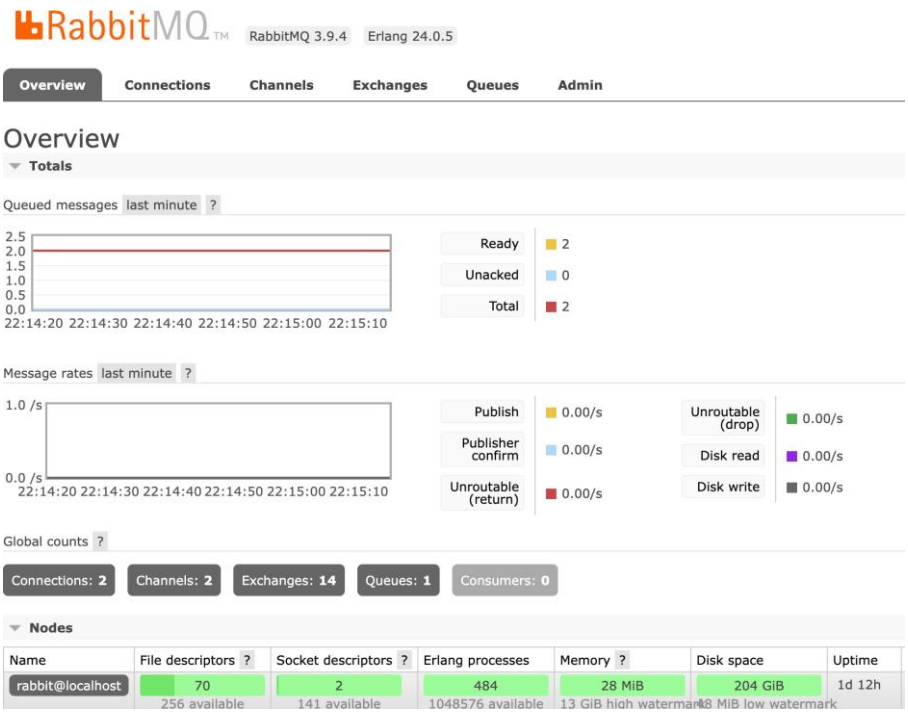


Рисунок 2.46 Функціональний результат RabbitMQ

Тести доступності

Ми провели тест на паралелізм на нашому інтерфейсі getone, як показано на рисунку 2.47, і програмне забезпечення wrk використало 40 потоків для захоплення продукту та 300 підключень за 10 секунд. Ми можемо виявити, що середня затримка інтерфейсу getone становить близько 2,53 мс, 99% затримки розподіляється в межах 5 мс. Обсяг запитів, оброблюваних за секунду, може досягати 110 000, що доводить, що наш інтерфейс snatch може використовуватися в RabbitMQ. Послідовне хешування покращило здатність системи витримувати великі обсяги трафіку та справлятися зі сценаріями з високим рівнем паралельності.

```
Transfer/sec:      2.44MB
[➜  ~ wrk -t40 -c300 -d10s --latency "http://127.0.0.1:8084/getOne"
Running 10s test @ http://127.0.0.1:8084/getOne
  40 threads and 300 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.53ms  615.48us  14.05ms   82.40%
    Req/Sec     2.77k   265.55     3.77k    79.45%
  Latency Distribution
     50%    2.55ms
     75%    2.77ms
     90%    3.00ms
     99%    4.61ms
  1110571 requests in 10.10s, 128.14MB read
  Socket errors: connect 0, read 117, write 0, timeout 0
Requests/sec: 109935.11
Transfer/sec:     12.68MB
```

Рисунок 2.47 – Тест доступності для інтерфейсу покупки

Варто зазначити, що ми створили сервіс лише локально, і якщо його розгорнути на хмарному сервері, він все одно зможе нормально обробляти велику кількість запитів, хоча його затримка збільшиться.

2.3.4 Висновки тестування

Ми можемо зробити висновок, що після вищезазначених чотирьох типів тестів наша система з високою кількістю паралельних покупок має кращу зручність використання, безпеку та відповідає потребам користувачів у покупках за умов високої паралельності.

Ми навели тести інтерфейсу для додавання, видалення, зміни та запиту замовлень як приклад, щоб перевірити, чи всі частини системи працюють належним чином. Ми можемо зробити висновок, що наша система не має проблем з точки зору функціональності.

Ми протестували безпеку системи з точки зору хакера, симулюючи два різні типи атак з порожнім та підробленим файлом cookie, і виявили, що наш механізм cookie може ефективно розрізняти нелегальних користувачів, а механізм шифрування AES ускладнює зловмисникам злом зашифрованих файлів cookie.

Наша система з високою кількістю паралельних покупок повинна витримувати високий трафік. Тому під час тестування зручності використання

ми змоделювали кілька з'єднань та протестували їх протягом певного періоду часу за допомогою автоматизованого інструменту тестування wrk, і виявили, що наша система може обробляти до 100 000 звернень за секунду, з потенціалом витримувати більше, якщо ми використовуватимемо розподілену систему.

Зрештою, ми завершили роботу над програмним забезпеченням, і тепер наш проект добігає кінця. У наступній частині висновків буде підбито підсумки роботи та запропоновано ідеї для майбутньої роботи.

ВИСНОВОК

Ми вже завершили розробку системи закупівель з високим рівнем паралельності для флеш-розпродажів на веб-сайті електронної комерції. Підсумовуючи, ми створили серверну систему управління продуктами та замовленнями для менеджерів, а також систему фронтенд-продажів для клієнтів. Оскільки ми використовували належні технічні інструменти та зручну програмну архітектуру, наша система є стабільною, безпечною та ефективною для користувачів. Клієнти отримають радісний та чесний досвід під час покупок. Менеджери отримають велику вигоду, отримуючи багато замовлень, отримуючи великий прибуток та не турбуючись про проблеми перепроданості завдяки такій стабільній та зручній системі управління.

Оскільки ми використовуємо мову програмування Go та фреймворк Iris з шаблоном MVC, архітектура програмного забезпечення є зрозумілою та легкою для розуміння розробниками для подальшої розробки. Крім того, завдяки розподіленій верифікації та послідовному алгоритму хешування, наше програмне забезпечення зможе масштабуватися, що також буде дуже корисним для майбутніх розробників.

Подальшу роботу над програмним забезпеченням можна завершити наступним чином: 1.Додайте код верифікації на фронтенд-інтерфейс, щоб уникнути реєстрації або входу в систему зі спамом. 2.Ми можемо додати більше шаблонів до моделей продуктів, таких як одиниця зберігання[24] (SKU) або атрибути продуктів, щоб надати користувачам більше вибору та створити зручний досвід покупок. 3.Ми можемо виконувати оптимізацію продуктивності фронтенду на рівні сторінки [25]. Наприклад, зменшити кількість зовнішніх HTTP-запитів, мінімізувати CSS, увімкнути попередню вибірку, збільшити швидкість за допомогою CDN та кешування, використовувати мінімалістичний фреймворк тощо. 4.Ми можемо оптимізувати алгоритм узгодженого хешування, щоб вирішити проблему точного зіставлення між запитами та

серверами кешу та усунути проблеми з серверами кешу.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Avinash Sharma. A Mini-Guide on Go Programming Language. https:/ / appinventiv .com / blog / mini- guide - to- go - programming - language/.

[2] Introduction to TCP/IP. https://w3schools.sinsixx.com/tcpip/tcpip_ intro.asp.htm.

[3] Golang. Concurrency. https://www.golang-book.com/books/intro/10.

[4] go philosophy. https://commandcenter.blogspot.com/2012/06/less-is-exponentially-more.html.

[5] Iris Web Framework. https://www.iris-go.com/.

[6] Jon Calhoun. Using MVC to Structure Go Web Applications. https://www. calhoun.io/using-mvc-to-structure-go-web-applications.

[7] MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/ en/introduction.html. Redis Documentation. https://redis.io/docs/.

[8] RabbitMQ. https://en.wikipedia.org/wiki/RabbitMQ.

[9] RabbitMQ Message Queue. https : / / www . rabbitmq . com / tutorials / tutorial-one-elixir.html.

[10] Jon Calhoun. Using MVC to Structure Go Web Applications. https://www. calhoun.io/using-mvc-to-structure-go-web-applications.

[11] Model–view–controller. https://en.wikipedia.org/wiki/Model%E2%80% 93view%E2%80%93controller.

[12] Jon Calhoun. Using MVC to Structure Go Web Applications. https://www. calhoun.io/using-mvc-to-structure-go-web-applications.

[13] Model–view–controller. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[14] Jon Calhoun. Using MVC to Structure Go Web Applications. https://www. calhoun.io/using-mvc-to-structure-go-web-applications.

[15] Model–view–controller. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[16] MDN Web Docs. HTML: HyperText Markup Language. https://developer. mozilla.org/en-US/docs/Web/HTML.

[17] MDN Web Docs. CSS: Cascading Style Sheets. https://developer.mozilla. org/en-US/docs/Web/CSS.

[18] MDN Web Docs. CSS Introduction. https://www.w3schools.com/css/css_ intro.asp.

[19] lakshita. Introduction to JavaScript. https :/ / www .geeksforgeeks.org / introduction-to-javascript/.

[20] MDN Web Docs. JavaScript. https://developer.mozilla.org/en-US/ docs/Web/JavaScript.

[21] Tomislav Bacinger. What is Bootstrap? A Short Bootstrap Tutorial on the What, Why, and How. https://www.toptal.com/front-end/what-is-bootstrap-a-short-tutorial-on-the-what-why-and-how.

[22] srandby. Dynamic and Static Website Security. https : / / srandby . org / digital-writing/index.html.

[23] Stock Keeping Unit (SKU). https:/ / www .shopify.com / encyclopedia/ stock-keeping-unit-sku.

[24] Arsenault. frontend optimization. https : / / www . keycdn . com / blog / frontend-optimization.

ПРЕЗЕНТАЦІЯ

Магазин високонавантаженої електронної комерції на мові програмування Go

Керівник: доцент
Кравчук Петро – PhD

Студент :
Ван Їсян

Свято шопінгу

- **Акція**
- **Знижка**
- **Чудові пропозиції**
- **Розваги**

- **ПЕРЕПОВНЕНО**

Інтернет-шопінг

- Доступні ціни
- Екологічно чистий
- Більше різноманітності та вибору
- Надсилання подарунків
- Менше нав'язливих покупок
- Більше зручності
- Без тиску з боку продавців
- Без натовпу! ?!

Переповнені онлайн-шопінги?

Клієнти	Магазини
• Перегляд та відвідування • Реєстрація та вхід • Натисніть та купіть	• Управління продуктами • Управління замовленнями

Вебсайт/програмне забезпечення для покупок може бути надзвичайно переповненим, особливо на фестивалях шопінгу!

Воно має бути стабільним постійно та забезпечувати хороший користувацький досвід.

Система закупівель з високою паралельністю

Особливості
• 1. Frontend сайт може витримувати великі потоки даних. • 2. Backend система може витримувати високий курс валют. • 3. Backend система може масштабуватися. • 4. Розробникам слід уникати проблеми перепроданості.

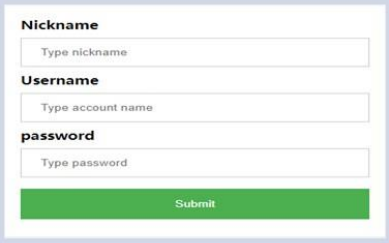
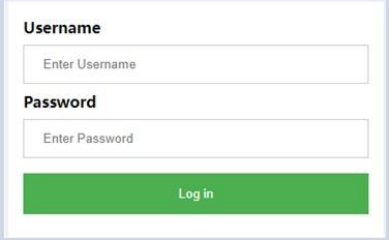
Основні інструменти та методи

Back-end	Front-end
• Go мова програмування • Iris framework • RabbitMQ • Model–view–controller(MVC)	• HTML • CSS • JavaScript • Bootstrap

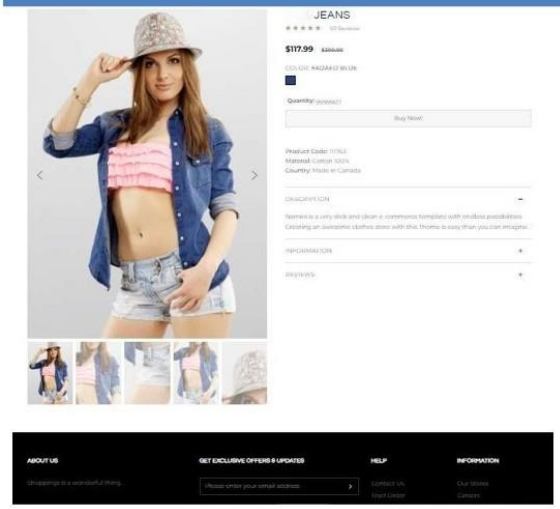
Виконання проєкту

Кроки	
• 1. запустіть backend/main.go //для налаштування backend • 2. запустіть fronted/main.go //для входу користувача • 3. запустіть getOne.go //щоб уникнути проблем перепроданості • 4. запустіть mq_consumer.go //споживачі/користувачі споживають • 5. запустіть validate.go // #Ключ# для послідовного хешування, розподіленої перевірки, тригера покупки	

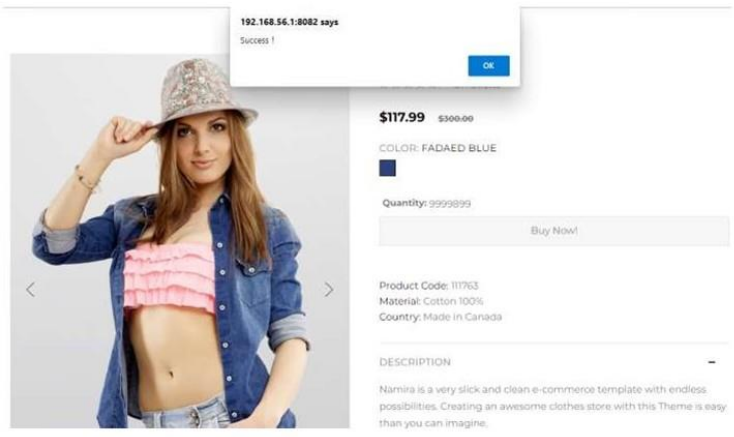
Виконання проєкту

Front-end		
• 	Реєстрація користувача	• Після успішної реєстрації та входу в систему користувач зможе придбати товари онлайн!
	Вхід користувача	

Виконання проєкту

Front-end	
	• Після успішного входу користувач зможе придбати товари онлайн!

Виконання проєкту

Front-end	Успішної покупки!
	• Після успішного входу користувач зможе придбати товари онлайн!

Виконання проєкту

Front-end	Зачекайте 10 секунд, щоб купити!

JEANS

★★★★★ 101 Reviews

$117.99 ~~$300.00~~

COLOR: FADAED BLUE

Quantity: 9999892

Waiting 9 seconds to buy

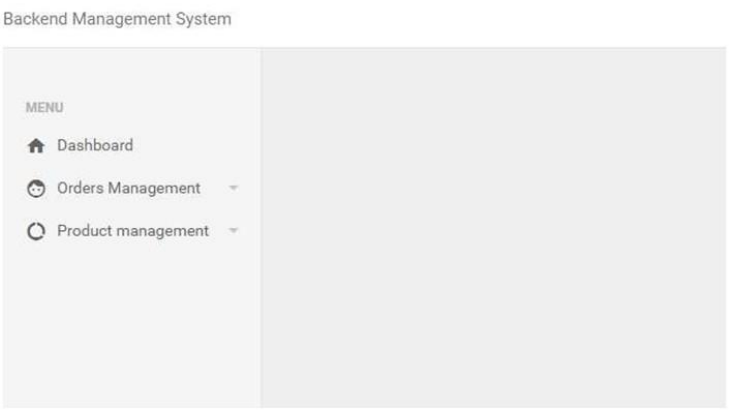
Product Code: 111763
Material: Cotton 100%
Country: Made in Canada

DESCRIPTION -

Namira is a very slick and clean e-commerce template with endless possibilities. Creating an awesome clothes store with this Theme is easy than you can imagine.

Виконання проєкту

Back-end	



Backend Management System

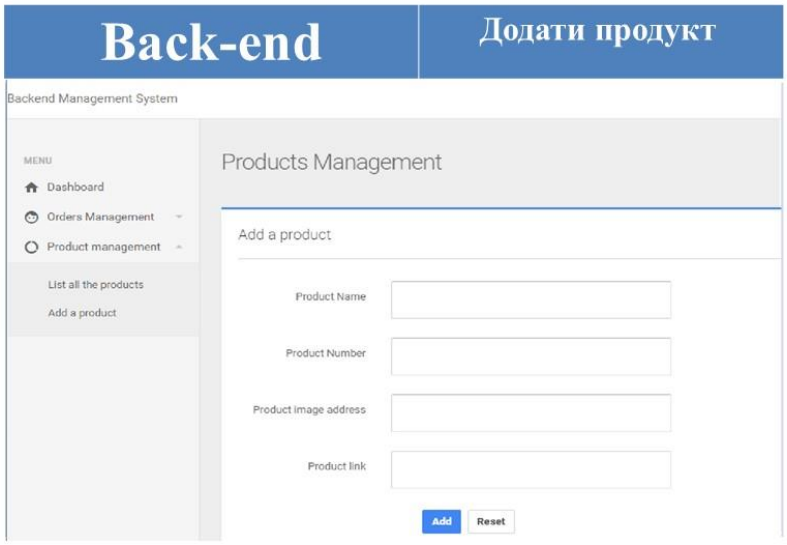
MENU

Dashboard

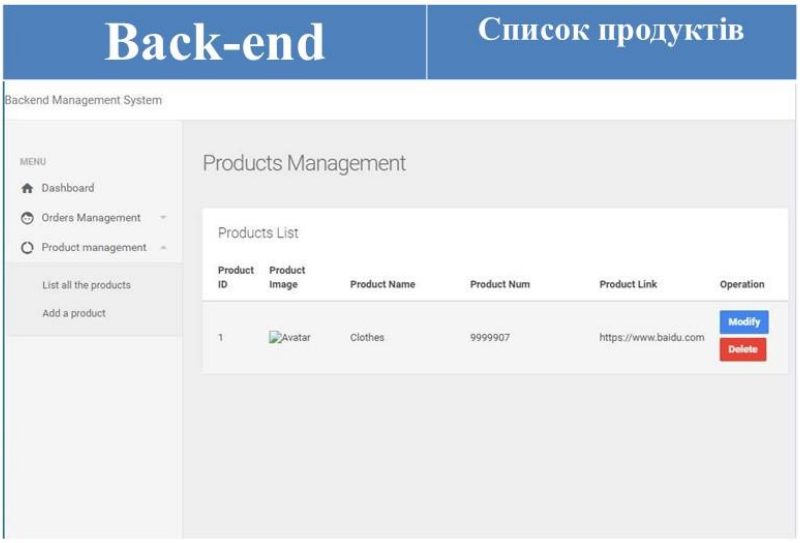
Orders Management

Product management

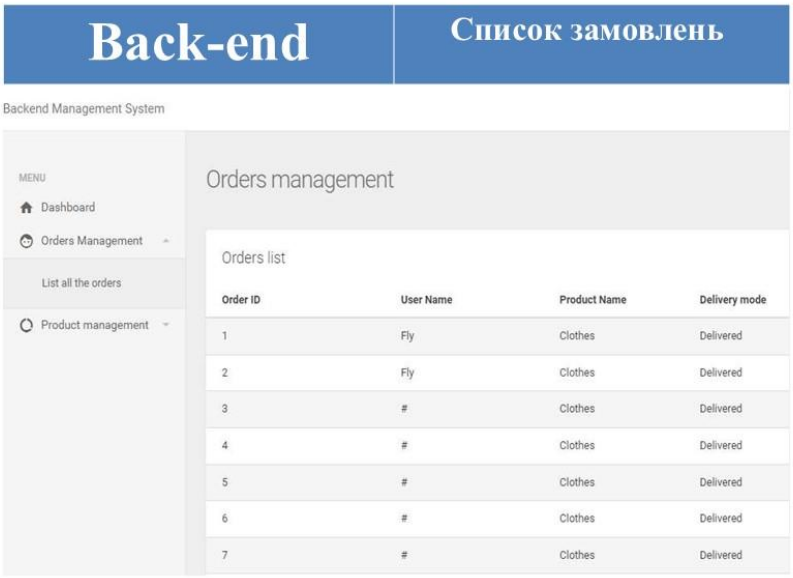
Виконання проєкту



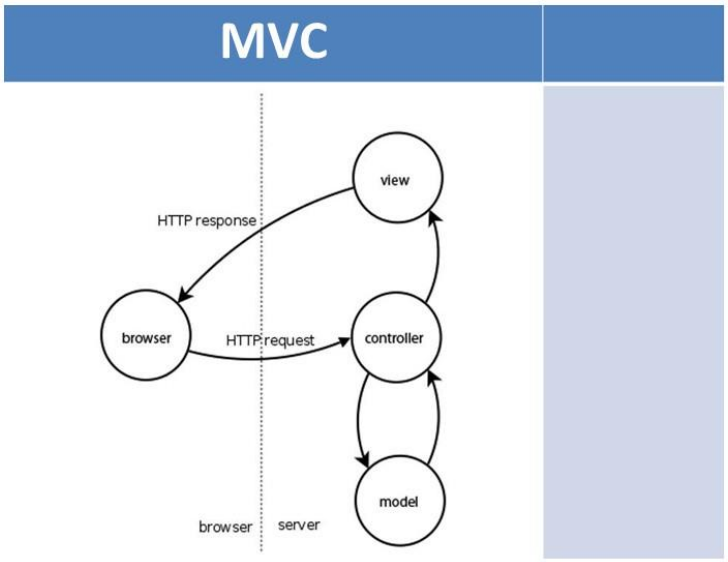
Виконання проєкту



Виконання проєкту



Model–view–controller

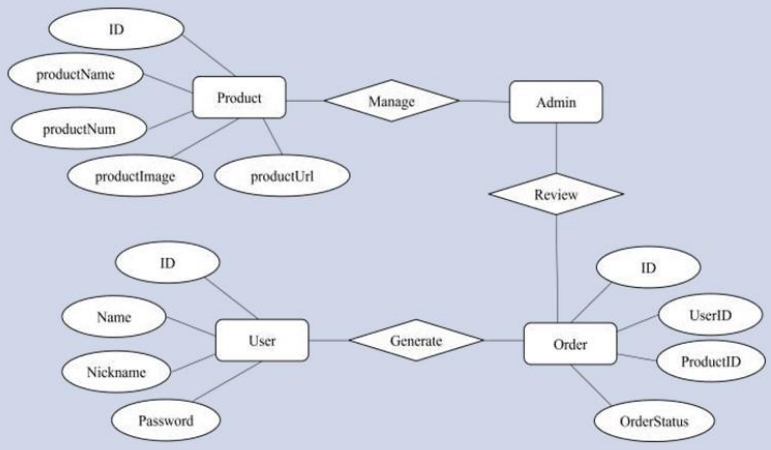


Model–view–controller

Довідник	backend
+-- backend # директорія бекенду | |-- web # веб-директорія | | |-- assets # зберігає статичні документи | | |-- controllers # контролери зберігають документи | | |-- views # директорія шаблонів документів | | | |-- main.go # головний файл | +-- datamodels # директорія моделей +-- repositories # директорія баз даних +-- services # директорія з кодом сервісів	

Об'єкти програмної системи

Діаграма E-R



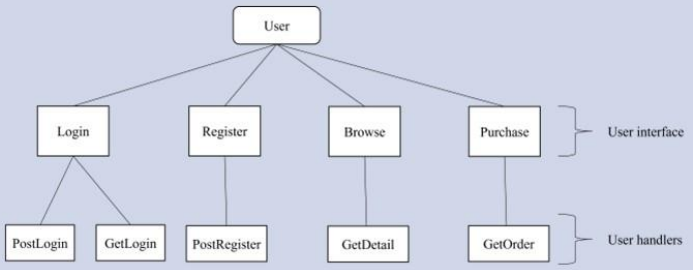
Розробка back-end

Продукт	Замовлення
• 1. Розробка дизайну моделей продуктів. • 2. Розробка функцій продуктів (з чотирма основними функціями: вставка, видалення, зміна та перевірка). • 3. Інтерфейс керування серверною частиною продуктів.	• 1. Розробка дизайну моделей замовлень. • 2. Розробка функцій замовлень (з чотирма основними функціями: вставка, видалення, зміна та перевірка). • 3. Інтерфейс управління замовленнями на внутрішній стороні.

Розробка front-end

Діаграма E-R

- 1. Розробка інтерфейсу входу користувача
- 2. Розробка функцій відображення фронтенд-продуктів
- 3. Розробка системи контролю даних про покупки



Алгоритм AES для входу

Методи	Переваги
• 1. Визначте шаблон доповнення AES • 2. Реалізуйте шифрування, яке складається з 4 частин: ----SubBytes ----ShiftRows ----MixColumns ----Add Round Key • 3. Реалізуйте дешифрування, яке є зворотним процесом до шифрування	• 1. Весь блок даних обробляється як єдина матриця. • 2. Працює за принципом підстановки та перестановки. • 3. Відкритий текст може мати довжину 128, 192 або 256 бітів. • 4. Має великий розмір ключа. • 5. Швидкий та безпечний

Оптимізація програмки

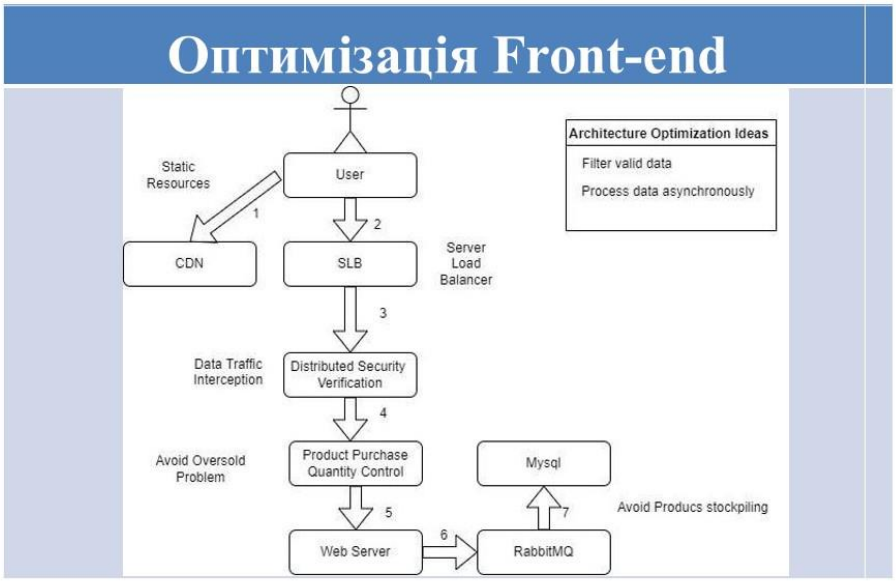
Архітектура Front-end

- 1. Користувач відвідує веб-сервер
- 2. Сервер взаємодіє з базою даних
- 3. База даних отримує інформацію та надсилає її назад на сервер
- 4. Сервер виконує рендеринг сторінки для користувача.

Проблема:

1. Веб-сервер матиме високе навантаження, якщо буде багато відвідувань.
2. База даних матиме високий тиск для отримання даних у режимі реального часу.
3. Важко підтримувати узгодженість даних — проблема перепроданості товарів.

Оптимізація програму



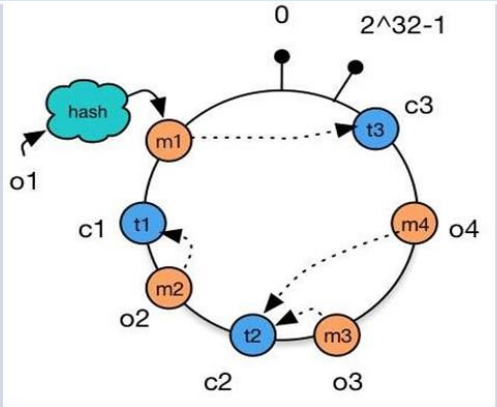
Оптимізація front-end

Дизайн статичної сторінки	Переваги
• 1. Створіть каталоги для зберігання згенерованого HTML-файлу та шаблонів статичних файлів. • 2. Напишіть функцію для генерації статичних HTML-файлів. • 3. Створіть контролер для генерації HTML-коду. • 4. Використайте функцію ParseFiles для генерації шляху шаблону. • 5. Виконайте рендеринг даних шаблону. • 6. Згенеруйте статичні файли з призначеними параметрами.	• 1. Він буде швидшим за динамічні. • 2. Витрачатиме менше ресурсів. • 3. Зменшить навантаження на сервер. • 4. Швидкий та безпечний

Оптимізація front-end

Контроль JavaScript	Переваги
• 1. Користувач може подати запит лише протягом десяти секунд. • 2. Тільки наші учасники (логін користувача) можуть працювати на наших вебсайтах.	• 1. Зменшити навантаження на бази даних. • 2. Підтримувати стабільну роботу веб-сайтів та серверів. • 3. Зменшити непередбачувані витрати ресурсів.

Оптимізація back-end

Алгоритм послідовного хешування	Функції
 Синій: Машина Помаранчевий: Об'єкт	• Пошук ресурсів • Балансування навантаження (машини) • Розподілена перевірка + Узгоджений алгоритм хешування ↓ Масштабування

Оптимізація back-end

Cookies	Переваги
• 1. Використайте контекстний пакет, щоб встановити http.cookie для вказаної пари ключ-значення. • 2. Змініть функцію входу користувача для автентифікації користувача ---- Отримайте номер облікового запису користувача та пароль. ---- Отримайте результати, порівнявши їх. ---- Якщо користувач пройшов автентифікацію, ми отримуємо легітимного користувача. В іншому випадку, перенаправте на сторінку інформації про продукт.	• Використовуйте файли cookie як токен ідентифікації користувача для автентифікації. • Легше впровадити порівняно із сеансами. • Зменште навантаження на сховище.

Тестування програму

Модульний тест	Тест безпеки
• В основному перевіряє функції замовлень, користувачів та продуктів у системі. • Самостійно тестує програмне забезпечення.	• Перевірте ефективність наших заходів безпеки. • Перевірте алгоритм шифрування AES та файли cookie системи.

Функціональний тест	Тест доступності
• Перевірте роботу нашої програми з точки зору користувача. • Змоделюйте доступ до URL-адрес за допомогою інструмента curl, щоб перевірити, чи отримано відповідні результати.	• Перевірте нашу високу паралельність, щоб перевірити, чи працює вона в ситуаціях з високим струмом. • Використайте wrk для проведення стрес-тесту.

Тестування програму

Тест доступності

```
~ wrk -t40 -c300 -d10s --latency "http://127.0.0.1:8084/getOne"
Running 10s test @ http://127.0.0.1:8084/getOne
  40 threads and 300 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.53ms  615.48us  14.05ms   82.40%
    Req/Sec     2.77k   265.55     3.77k    79.45%
  Latency Distribution
     50%    2.55ms
     75%    2.77ms
     90%    3.00ms
     99%    4.61ms
  1110571 requests in 10.10s, 128.14MB read
  Socket errors: connect 0, read 117, write 0, timeout 0
Requests/sec: 109935.11
Transfer/sec:     12.68MB
```

обробити 1 мільйон запитів за 10 секунд

Система з високим рівнем паралельності

- wrk можна запускати лише в системі linux/unix
- t - потік
- c - з'єднання
- d - тривалість

Висновок

Система онлайн-покупок

- Високий рівень паралельності
- Масштабування
- Відсутність проблеми перепроданості