МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ Є. О. Давиденко

«___»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

COMPOSER-ПАКЕТ LARAVEL ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.21810914

Студент _____ Є. А. Калашников

«___»_____2022 р.

Керівник д-р техн. наук, доцент _____ А. В. Швед

«___»_____2022 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

«__»_____2022 р.

Миколаїв 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
КРБ	–	кваліфікаційна робота бакалавра
ПЗ	–	програмне забезпечення
API	–	Application programming interface
MVC	–	Model-View-Controller
UML	–	Unified Modeling Language
PSR	–	PHP Standards Recommendations
JSON	–	JavaScript Object Notation
ORM	–	Object-Relational Mapping

ВСТУП

Актуальність теми КРБ зумовлена поширенням розробки сайтів та вебзастосунків з великою користувацькою базою використовуючи php-фреймворк Laravel. З кожним роком фреймворк отримує оновлення та новіші функції, які допомагають прискорити та полегшити процес розробки, оптимізувати процеси з базою даних. На даний момент налічуються більше ніж 1,519,305[1] сайтів розроблених на данному php-фреймворку. Більш ніж 30% складають eCommerce сайти, а саме 459,818[2] та з кожною хвилиною показник все збільшується через відкриття все більшої кількості малих та середніх бізнесів товарного напряму. З 2015 року ВВП малого та середнього бізнесу виріс на 50% відсотків[3], що стрімко вплинуло на розвиток сайтів з напрямом eCommerce. Звідси попит на eCommerce сайти виріс, тому швидкість розробки таких сайтів має велике значення.

Найбільш витратним в розробці eCommerce сайтів є:

- розробка власної системи кошику з кешуванням різних типів;
- власної системи адміністрування з обчислювальними методами;
- власної системи сповіщення користувачів з використанням форми отриманих товарів, тобто invoice`у;
- створення облікової системи розпізнавання даних користувача.
- обробка даних замовлень з системою дискаунтів, промокодів та вибору способу обробки замовлення за заданими параметрами.

Саме в ситуаціях, де потрібно знову прописувати дублюючий код, який вже використовується в інших проєктах допомагають composer-пакети. Мета Composer-пакетів прискорити розробку та розширити і додати нові функції в існуючий проєкт не вдаючись в логіку роботи пакету використовуючи функції з файлу документації, яка прикріплена до composer-пакету або вказана на сайті розробника.

Мета: спрощення процесу розробки eCommerce сайту за рахунок створення composer-пакету для php-фреймворку Laravel, що містить набір предметно-орієнтованих рішень та розширеного функціоналу у відповідності до створюваного проєкту.

Об’єкт дослідження: процес налаштування функцій розробника та конфігурації Laravel проектів орієнтованих на e-commerce.

Предмет роботи: інформаційні технології та інструментальні засоби розробки програмного забезпечення composer-пакету.

Для досягнення мети треба вирішити відповідні **завдання**:

- проаналізувати вже існуючі рішення, а саме: сценарії роботи системи, версії php-фреймворку які підтримує рішення, засоби апаратної та програмної реалізації;
- спроєктувати систему у вигляді діаграми використання, діаграми класів, діаграми станів та діаграми діяльності;
- розробити back-end частини з використанням php-фреймворку Laravel;
- завантажити composer-пакету на ресурси.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ АВТОМАТИЗАЦІЇ ПРОЦЕСУ РОЗРОБКИ COMPOSER-ПАКЕТІВ

1.1 Опис предметної сфери розробки composer-пакетів

Composer – це менеджер, який слугує для підключення та керування сторонніми бібліотеками у PHP-проєкті. Також composer ще має назву **пакетний менеджер**. Головна відповідальність пакетного менеджеру є відслідковування корректних зв’язків між встановленими бібліотеками. Зазвичай менеджер використовує файл, де знаходиться вся конфігурація та всі сторонні бібліотеки, які повинні бути присутні у проєкті. Composer використовує файл «**composer.json**», в якому вказані всі сторонні бібліотеки в JSON-форматі. При наразі помилки бібліотеки або структури проєкту пакет одразу повідомляє у консоль, сервер або log-файл. На даний час composer використовують такі популярні фреймворки як Symfony, Yii та Laravel.

Composer-пакет – стороння бібліотека, яка має в собі набір функцій. Даний набір функцій допомагає розширити або модифікувати вже існуючий проєкт. Така бібліотека є репозиторієм, яка зберігається в окремому місці з усіма існуючими бібліотеками composer. Щоб встановити пакет, треба використати команду *composer* в консолі з ім’ям пакету у якості параметру. Пакет відразу з’явиться у вигляді залежності до файлу конфігурації проєкту. Кожен пакет має певну структуру файлів, завдяки якій пакетний менеджер аналізує та обробляє пакет згідно PSR та файл «**composer.json**», який повідомляє про основні дані(такі як ім’я, автор, тощо) та залежності щодо інших бібліотек. Якщо файли пакету не відповідає стандартам PSR-4, то composer не зможе корректно використовувати даний пакет та виведе повідомлення про помилку. Базовими та обов'язковими полями у файлі «**composer.json**» є:

- name;
- description;

- license;
- require;
- authors.

Необов'язкові поля можна подивитися на сайті з офіційною документацією.

1.2 Керування репозиторіями на основі composer-пакетів

Репозиторій – окреме місце, де зберігаються певні дані у вигляді файлів з можливістю огляду кожної версії оновлення файлу. Найвідомішим репозиторієм на наш час є репозиторій **Git**. Зазвичай репозиторії використовують консоль як інструмент для роботи.

Composer-пакети також використовують систему контролю версій та власний репозиторій. Таким репозиторій є окремий сайт, де кожен розробник може безкоштовно завантажити на ресурс свій пакет та безкоштовно використовувати пакети інших розробників. Кожен розробник може обрати окрему версію пакету та встановити її у свій проєкт. Щоб встановити пакет обраної версії треба використати команду *composer<ім'я пакету>@<версія пакету>*. Після чого composer автоматично почне встановлення пакету та додасть його до файлу конфігурації залежностей. Якщо такої версії у репозиторії нема, то composer відразу покаже повідомлення та причину помилки.

Контролювати саме які версії всіх composer-пакетів повинні бути в проєкті можна через конфігураційний файл. У файлі вказується зареєстроване ім'я та версія, яка повинна бути. Також можна вказати окремі параметри, які будуть свідчити про те, які самі версії можуть бути встановлені в проєкт, тобто буде використано обмеження точної залежності. У файлі можна задавати діапазон версій за допомогою операторів (рис. 1.1): **>**, **>=**, **<**, **<=** та **!=**. Кожен оператор свідчить про наявність версії та вирішенні задачі при відсутності версії пакету в репозиторії. Для більш точного налаштування можна

використовувати оператори такі як: **кома** або **whitespace**, **подвійна вертикальна лінія**. Також для діапазону можна використати звичайний **дефіс**. Один з найбільш розповсюджених операторів для конфігураційного файлу є оператор **тільда**, який обирає діапазон серед дочірніх версій. Наступним розповсюдженим оператором є оператор ^. Він схожий на тільду, але обирає найближчий діапазон серед головних версій. Щоб не піклуватися про стабільність версії пакету в конфігураційному файлі присутні такі параметри як **-dev** та **-stable**. Кожен з них присутній в файлі неявно, але якщо потрібно, можна вказувати явно, щоб composer встановив потрібну версію.

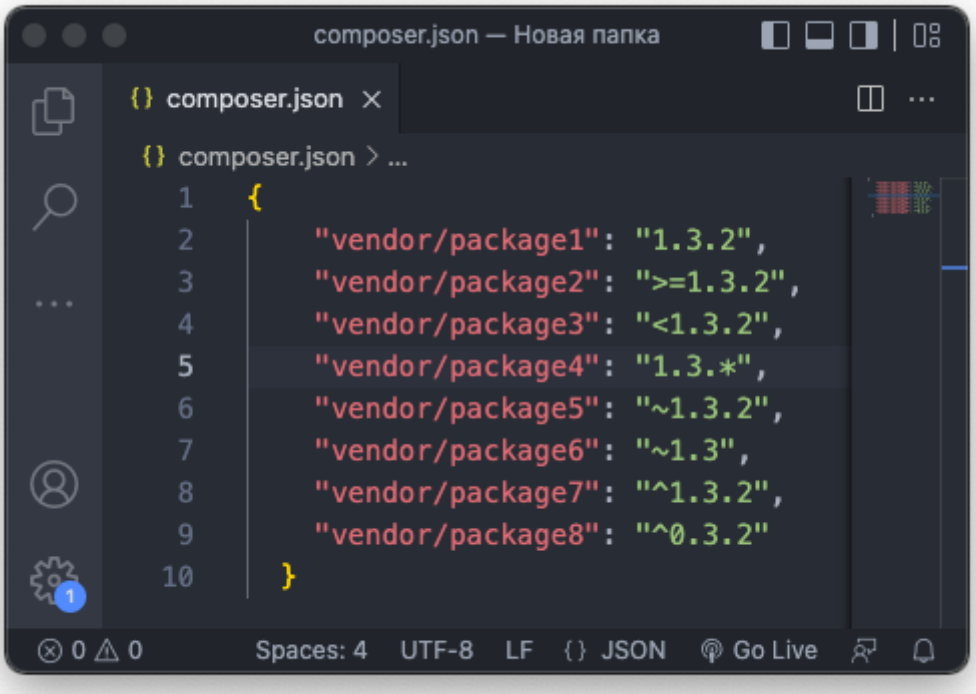

Рисунок 1.1 – Приклад використання операторів в файлі конфігурації

1.3 Аналіз та порівняльна характеристика сучасних PHP-фреймворків, що підтримують пакетний менеджер composer

В наш час існує три найпопулярніші PHP-фреймворки, які використовують composer: **Laravel** (табл. 1.1), **Symfony** (табл. 1.2) та **Yii** (табл 1.3). Вони всі мають архітектуру MVC. Усі порівняння були взяті з сайту[4],

який проводить опитування серед людей, які користуються даними фреймворками.

Таблиця 1.1 – PHP-фреймворк Laravel

Назва	Laravel
Версія PHP	1. Laravel 4.0, 4.1 ≥ PHP 5.3.0; 2. Laravel 4.2, 5.0 ≥ PHP 5.4.0; 3. Laravel 5.1 LTS, 5.2 ≥ PHP 5.5.9; 4. Laravel 5.3, 5.4 ≥ PHP 5.6.4; 5. Laravel 5.5 LTS ≥ PHP 7.0.0; 6. Laravel 5.6, 5.7, 5.8 ≥ PHP 7.1.3; 7. Laravel 6 LTS, 7 ≥ PHP 7.2 та ≤ PHP 8.0; 8. Laravel 8 ≥ PHP 7.3 та ≤ PHP 8.1; 9. Laravel 9 PHP ≥ 8.0 та ≤ PHP 8.1; 10. Laravel 10 (анонс) ≥ PHP 8.1.
Версія MySQL	MySQL ≥ 5.1
Переваги	1. зрозуміла архітектура; 2. зростаюча спільнота; 3. дружня з composer; 4. відкритий код; 5. фреймворк стрімко розвивається; 6. dependency injection; 7. MVC; 8. ORM (Eloquent); 9. маршрути, які підтримують RESTful; 10. міграції бази даних та заповнення за допомогою seed; 11. artisan scaffolding; 12. зручна документація; 13. шаблонізатор blade.

Кінець таблиці 1.1

Недоліки	1. доволі повільна; 2. занадто багато composer залежностей; 3. багато статичних методів.
Hacker News Points	516
Reddit Points	5.3K
Stack Overflow Questions	191.9K
GitHub Stars	69.7K
Репозиторій	https://github.com/laravel/laravel
Офіційна документація	https://laravel.com/docs

Таблиця 1.2 – PHP-фреймворк Symfony

Назва	Symfony
Версія PHP	1. Symfony 2.0 ≥ PHP 5.3.2; 2. Symfony 2.1 – 4.3 ≥ PHP 5.3.3; 3. Symfony 4.4 ≥ PHP 7.1.3; 4. Symfony 5.0 – 5.4 ≥ PHP 7.2.5; 5. Symfony 6.0 ≥ PHP 8.0.2; 6. Symfony 6.1 (анонс) ≥ PHP 8.1.0;
Версія MySQL	MySQL ≥ 5.1
Переваги	1. відкритий код; 2. dependency injection; 3. організована; 4. модульна архітектура; 5. присутні принципи SOLID; 6. інструкція з передової практики; 7. сервісний контейнер; 8. YAML; 9. MVC; 10. зрозуміла документація.

Кінець таблиці 1.2

Недоліки	1. дуже багато конфігураційних файлів; 2. велика кількість сторонніх бібліотек;
Hacker News Points	304
Reddit Points	11.8K
Stack Overflow Questions	70.9K
GitHub Stars	26.9K
Репозиторій	https://github.com/symfony/symfony
Офіційна документація	https://symfony.com/doc

Таблиця 1.3 – PHP-фреймворк Yii

Назва	Yii
Версія PHP	1. Yii 1.1 ≥ PHP 5.1.0; 2. Yii 2.0 ≥ PHP 5.4.0; 3. Yii 3.x (анонс) ≥ PHP 7.4.0;
Версія MySQL	MySQL ≥ 5.1
Переваги	1. відкритий код; 2. генератор коду; 3. MVC; 4. стабільні версії; 5. довгострокова підтримка; 6. модульна архітектура; 7. дуже швидкий фреймворк;
Недоліки	1. реалізує код-спагетті; 2. неприродня любов до масивів; 3. пропагує погану практику коду;
Hacker News Points	307
Reddit Points	1.3K
Stack Overflow Questions	16.8K

Кінець таблиці 1.3

GitHub Stars	4.8K
Репозиторій	https://github.com/yiisoft/yii2
Офіційна документація	https://www.yiiframework.com/doc

1.4 Способи створення composer-пакету в Laravel

Існує два типи composer-пакетів. Перший тип цілковито автономний, наприклад **Carbon** або **PHPUnit**, а інший співіснує лише з Laravel, тобто конфігурація, роути, контролери, шаблони, тощо працюють лише з проєктом, який створений за допомогою фреймворку Laravel.

Laravel дізнається про існування composer-пакету за допомогою конфігураційного файлу «**config/app.php**», в якому прописуються всі постачальники у полі **providers** (рис. 1.2).

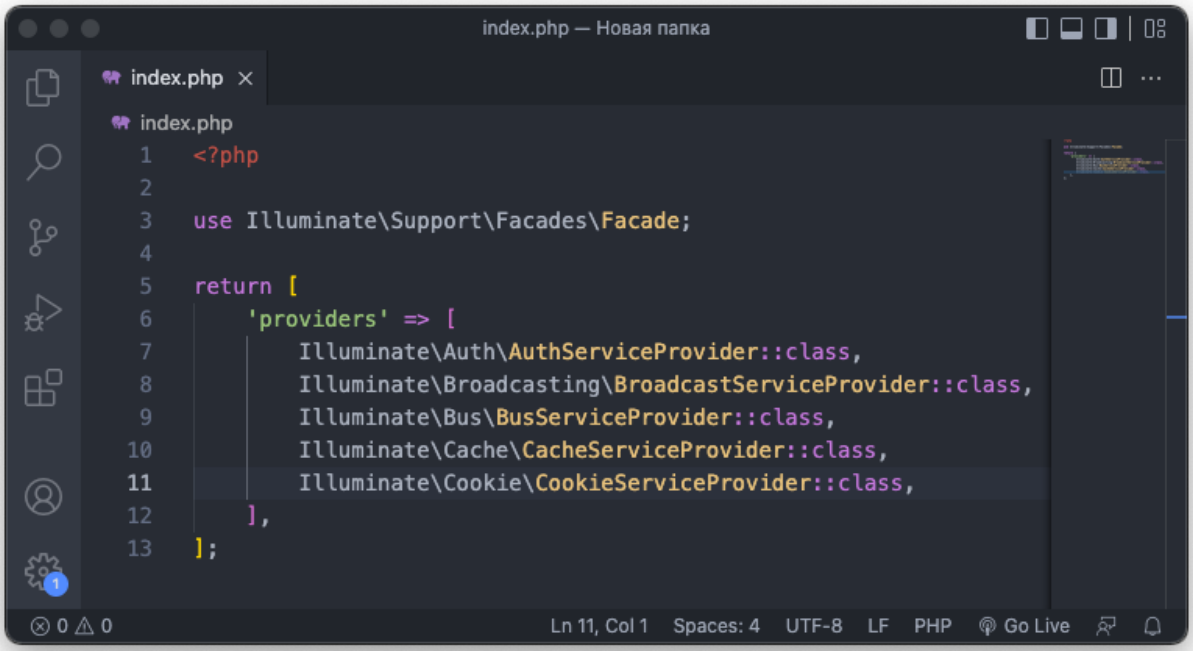

Рисунок 1.2 – Приклад підв’язування пакетів

Також головним аспектов в створенні пакету є постачальники служб, а саме клас **ServiceProvider**, який має два методи: **boot** та **register**. Дані методи слугують базовими для кожного постачальника, який буде приєднано до проєкту Laravel.

У методі **boot** прописується що саме потрібно робити під час запуску. Зазвичай в цьому методі прописуються методи публікацій:

- конфігурації;
- роутів;
- міграцій;
- перекладів;
- файлів;
- шаблонів;
- команд artisan;
- публічних ресурсів.

Далі розробка пакету проходить як звичайний процес розробки Laravel проєкту.

1.5 Огляд аналогів створюваного composer-пакету

Першим прикладом існуючого ПЗ буде composer Laravel пакет **Laravel Basket**, розробником якого є користувач GitHubfaustbrian. Даний пакет є доповненням архітектури вебзастосунку на базі php-фреймворку Laravel. Мовою реалізації даного пакету є PHP.

Пакет має такий перелік функцій та характеристик:

- Додавання товару до кошику з можливістю редагування, видалення та фільтрації;
- Використання PHP версією не нижче 7.2;
- Має файл з додатковими функціями, які завантажуються в процесі ініціалізації проєкта;
- Використовується з базою MySql;
- Код відповідає стандартам PSR-0, PSR-1, PSR-2, PSR-4;
- Має широкий асортимент функцій з різною сигнатурою;
- Для встановлення пакету лише треба запустити консольну команду та дотримуватись документації.

Переваги:

- Підтримує старі версії фреймворку;
- Має зручну документацію інсталяції та використання.

Недоліки:

- Підтримує лише старі версії Laravel та PHP;
- Немає гнучкого налаштування системи.

Наступним аналогом ПЗ, який розробляється є composer пакет **Laravel Extendable Basket**, розробником якого є користувач GitHub DivineOmega. Даний пакет є розширенням системи на основі фреймворку Laravel.

Основні функції та характеристики:

- Додавання товару до кошику з можливістю редагування, видалення та фільтрації;
- Має методи автоматичного підрахунку суми всіх товарів, які знаходяться у кошику;
- Має файл з додатковими функціями, які завантажуються в процесі ініціалізації проєкта;
- Код відповідає стандартам PSR-4;
- Фільтрація товарів за певними характеристиками.

Переваги:

- Підтримує старі версії фреймворку;
- Працює з актуальною для розробки версією PHP;
- Працює з старими версіями MySql;
- Присутня вбудована фільтрація та пошук;
- Має зручну документацію інсталяції та використання.

Недоліки:

- Підтримує лише стару версію MySql;
- Немає гнучкого налаштування системи;
- Фільтрація обмежена;
- Підтримує лише старі версії Laravel та PHP;

- Код не відповідає стандартам написання PSR-0, PSR-1, PSR-2.

Останнім аналогом існуючого пакету є composer пакет **LaravelBasket**. Розробником даного пакету є користувач GitHub ChrisWillerton. Мовою реалізації даного пакета є PHP. Даний пакет допомагає швидко налагодити роботу кошика в інтернет-магазині на базі фреймворку Laravel.

Основні функції та характеристики:

- Додавання товару до кошику з можливістю редагування та видалення;
- Має методи автоматичного підрахунку суми всіх товарів, які знаходяться у кошику;
- Має файл з додатковими функціями, які завантажуються в процесі ініціалізації проєкта;

Переваги:

- легка інсталяція.

Недоліки:

- підтримка лише стару версію MySql;
- відсутність гнучкого налаштування системи;
- відсутність можливості фільтрації контенту;
- підтримка лише старі версії Laravel та PHP;
- код не відповідає стандартам написання PSR-0, PSR-1, PSR-2;
- відсутність гнучкого налаштування системи;
- відсутність документації.

1.6 Специфікація вимог до програмного забезпечення composer-пакету для розробки Інтернет-магазину

Системою, що розробляється є composer пакет до php-фреймворку Laravel. Даний пакет є збіркою вже існуючого функціоналу інтернет магазину та доповненням до вже існуючого проєкту з додатковим функціоналом.

Основні функції:

- Додавання товару до кошику з можливістю редагування або видалення;
- Робота в режимі коли система налаштована в стані сесії;
- Робота в режимі коли система налаштована працювати з клієнтською частиною через cookie;
- Фільтрація товарів за певною характеристикою;
- Автоматичне підрахування замовлення;
- Використання промокодів до замовлення;
- Створення PDF-файлу з створеним замовленням (Invoice);
- Налаштування вигляду Invoice`у в окремому файлі *.blade;
- Експорт даних у вигляді JSON, XML та інших форматів;
- Врахування особливостей цінової політики (за місцем знаходження та іншими параметрами);
- Налаштування особливостей цінової політики в окремому файлі;
- Редагування даних замовлення адміністратором;
- Робота системи може використовуватись не за прямим призначенням;
- Перетворення вигляду класу з реляційної бази даних у постреляційну;
- Адаптивність системи у разі переключення з режиму сесії у режим cookies;
- Адаптивність системи у разі переключення з режиму cookies у режим сесії;
- Розширення можливостей вже існуючих класів у системи;
- Використання поліморфного зв’язку класів з базою даних;
- Врахування ціни доставки. Користувач системи сам встановлює ціну. Ціна може бути встановлена як числовим форматом так і у вигляді відсотків;
- Має вбудовані artisan команди для генерації файлів єкспорту.

Висновки до розділу 1

Проведено опис предметної сфери розробки composer-пакетів та пакети як спосіб керування репозиторіями. Аналіз розробки проєкту на базі php-фреймворку з використанням composer. Описано використання composer, як систему контролю версіями залежностей бібліотек у створюваному проєкті шляхом конфігурування файлу «**composer.json**» та змінення параметрів шляхом використання операторів.

Проведено аналіз та порівняльну характеристику серед опитаних користувачів існуючих php-фреймворків, які підтримують composer, а саме Laravel, Symfony та Yii різних версій.

Проаналізовано та зроблено порівняльну характеристику вже існуючих аналогів сторюванного ПЗ composer-пакету. Виявлено переваги та недоліки кожного аналогу створюванного ПЗ composer-пакету.

Розглянуто та описано способи створення composer-пакету для середовища з розробкою проєкту на базі php-фреймворку Laravel. Сформульовано специфікаційні вимоги щодо майбутнього ПЗ composer-пакету, а саме встановлення функціональних вимог до ПЗ.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ COMPOSER-ПАКЕТУ

2.1 Діаграма прецедентів (Use case)

Діаграма прецедентів – діаграма, яка моделює відносини між акторами та прецедентами, описуючи систему на концептуальному рівні. Така діаграма містить головні компоненти у вигляді **актора**, **прецедента** та **рамки системи**.

Прецедент – абстрактна функція системи, яка призводить до певного результату.

Актор – людина або система, яка причинна до прецеденту.

Рамки системи – прямокутна межа, в якій знаходяться прецеденти та зв’язок між ними. У верхній частині має назву системи, яка розробляється.

В діаграмах прецедентів зустрічаються три базові типи відношень (рис. 2.1), які свідчать про різні відношення між прецедентами.

Перший базовий тип відношення є **асоціативний**. Такий тип відношень є фундаментальним серед діаграм UML та вказує на тісну залежність між системою або актором з прецедентом. Позначається стрілкою з напрямом на прецедент.

Другий тип – **включення**. Такий тип відношення сигналізує про те що зміна одного прецедента призводить до зміни іншого. Перший прецедент є базовим та саме він регулює стан. Включення на діаграмі позначається ключовим словом *include* та стрілкою з штризовою лінією.

Третій тип – **розширення**. Даний тип відношення використовується у випадку, коли від рішення базового прецеденту залежить чи буде використовуватись наступний прецедент. Такий тип на діаграмі позначається ключовим словом *extend* та стрілкою з штриховою лінією.

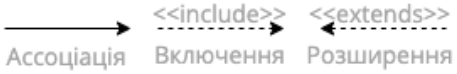

Рисунок 2.1 – Базові типи відношень

Також поділяють ще три типи не базових відношень (рис. 2.2), а саме: тип, який **вимагає** виконання попереднього прецедента; тип, який є **схожим** та **рівнозначний** тип.

Тип, який **вимагає** – такий тип наказує попередньому прецеденту бути виконаним. На діаграмі позначається так само як і **включення**, але з ключовим словом *require*.

Тип, який є **схожим** – подібний прецедент, але має іншу реалізацію функціоналу. На діаграмі позначається так само як і **включення**, але з ключовим словом *resembles*.

Рівнозначний тип – прецедент, який має подібну реалізації функціональність як і у базового прецедента, але користувач сприймає по різному. На діаграмі позначається так само як і **включення**, але з ключовим словом *equivalent*.

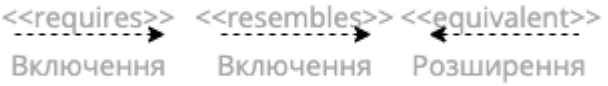

Рисунок 2.2 – Типи відношень

Виділяють ще один тип відношення з пустою стрілкою, який є подібним до відношення базового та дочірнього класу.

У діаграмі до розроблюванного ПЗ (рис. 2.3) присутні чотири актори:

- адміністратор;
- клієнт системи;
- користувач системи;
- composer-пакет для розробки Інтернет-магазину.

Composer-пакет виступає в ролі посередника між розробником та сайтом, який розробляється. В даному випадку в діаграмі прецедентів видно, що розробник лише розробляє роути використання власної системи, в той час як composer-пакет виконує функції, які вже є прописаними та не потребують у

додатковій реалізації (у випадку якщо функціоналу недостатньо, розробник може написати додатковий функціонал).

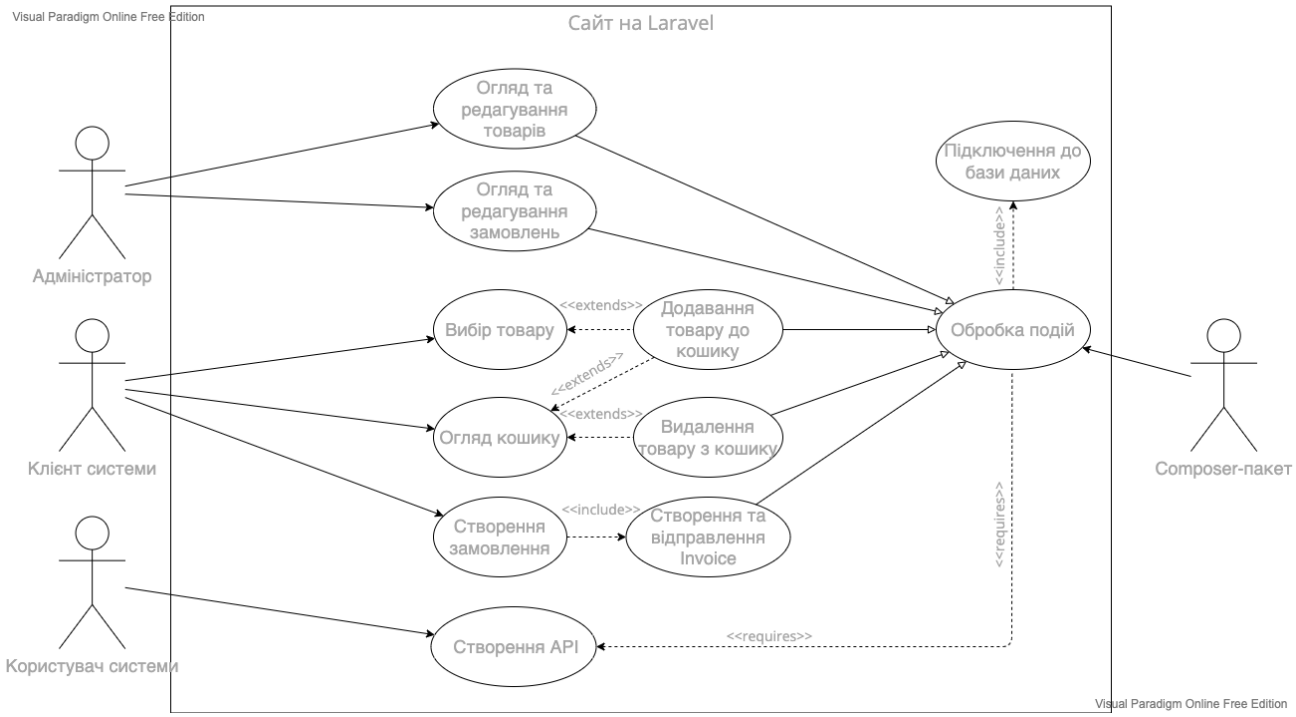

Рисунок 2.3 – Діаграма прецедентів

У даній діаграмі огляд та редагування товарів і замовлень виконується за допомогою вбудованих функцій, які присутні у composer-пакеті.

Під ключовим словом *Обробник подій* передбачається базовий клас з статичними методами, які роблять певні вичислення та дії з моделями реляційної бази даних. Задля роботи даних функцій потрібно розробнику (користувачу системи) прописати API, який буде викликати дані методи з певними параметрами. Як саме буде працювати API вирішує розробник, але робота контролерів потрібна відображати логіку, яка передбачена документацією даного пакету.*Огляд та редагування товарів, Огляд та редагування замовлень, Додавання товару до кошику, Видалення товару з кошику* та *Створення та відправлення Invoice* входить до базового функціоналу пакету та у діаграмі відображається як прецедент, який відноситься до прецеденту *Обробник подій*. Всі інші прецеденти на діаграмі виконуються

згідно уподобань розробника (користувача системи) та їх реалізація виконана користувачем системи.

2.2 Діаграма активності (Activity diagram)

Діаграма активності – візуальне представлення дій у вигляді блок-схем або потоку даних.

Діаграма активності має такі умовні позначення:

- **початок** – позначається чорним або кольоровим колом з якого виходить стрілка;
- **активність** – прямокутник, який відображає абстрактне виконання певної дії;
- **стрілка** – позначає перехід з однієї дії до іншої;
- **рішення та розгалуження** – позначається параллелограммом та має дві вихідні стрілки;
- **час події** – позначається значком, який схожий на пісочний таймер.
- **злиття подій** – позначається у вигляді параллелограмма, який приймає в себе необмежену кількість стрілок до однієї вершини та з виходом однієї стрілки на нішному кінці.
- **синхронізація** – позначається горизонтальною прямою до якої надходять стрілки;
- **кінець** – позначається колом з іншим колом у центрі, яке зафарбовано кольором.

Composer-пакет виступає у ролі абстракції у діаграмі активності передбачуваного сайту (рис. 2.4), яка виконує певні функції. В даному випадку composer-пакет описується як посередником між роутами розробника та адміністратором сайту, але composer-пакет не є окремою оболонкою, а лише доповненням до певних дій розробника.

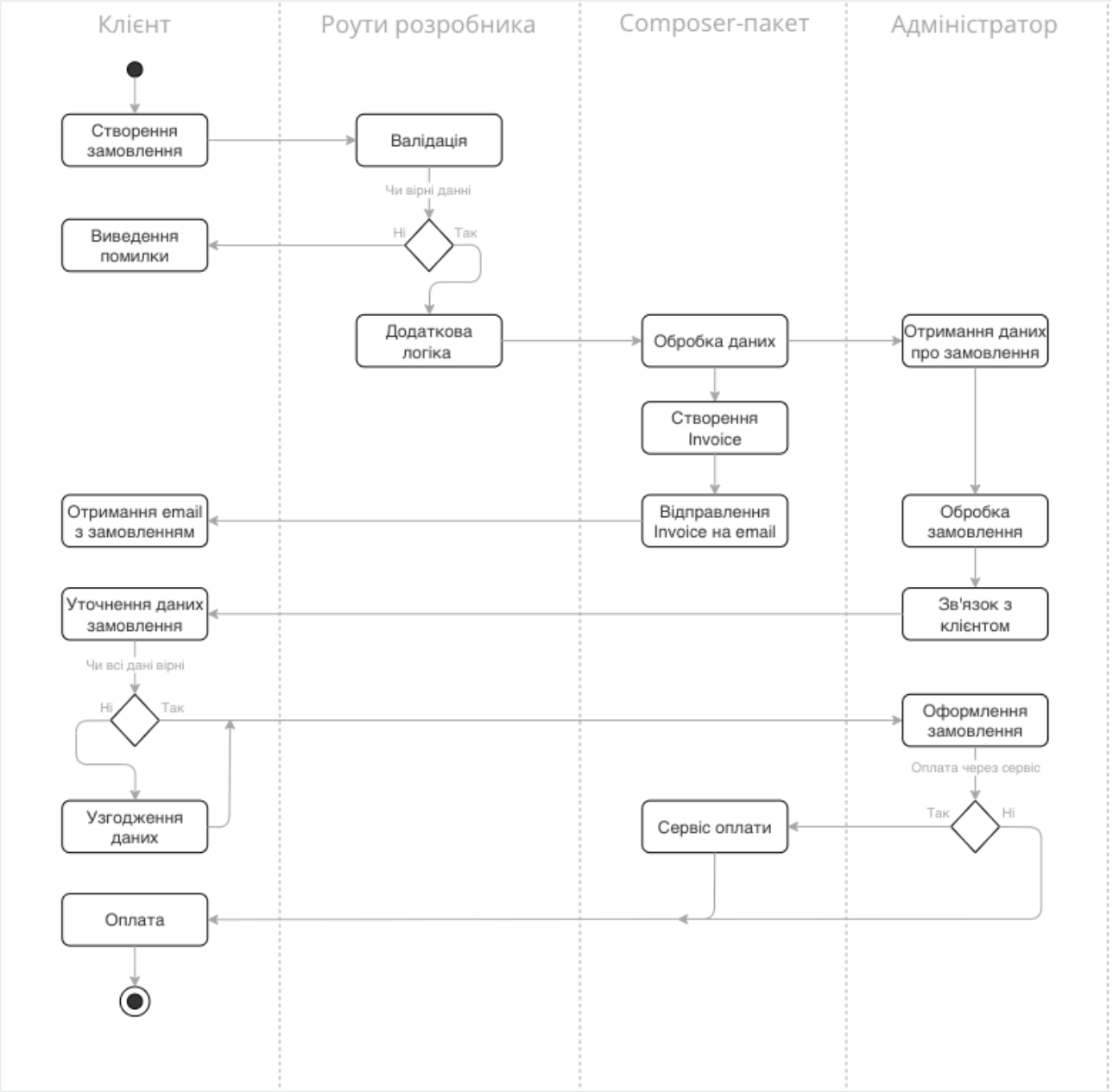

Рисунок 2.4 – Діаграма активності Інтернет-магазину

На діаграмі зображено використання composer-пакету задля досягнення певних операцій, а саме:

- обробка даних шаблону замовлення;
- створення Invoice`у та відправлення його;
- використання сервісу оплати.

Обробка даних шаблону замовлення проходить автоматично, розробнику не потрібно вдаватися в логіку роботи методу, а лише ввести в параметри

потрібні дані. Саме на цьому етапі буде використовуватись сесія користувача та під'єднання сесії до бази даних за для створення замовлення. Сесія буде створена не в залежності від того чи авторизований користувач. Система в якості сесії бере IP-адресу, операційну версію та її версію, браузер та його версію.

Створення Invoice`у та відправлення проходить автоматично за допомогою вже встановлених івентів в Laravel, які допомагають реалізувати дії паралельно з реалізацією базового коду. В коді присутній певний шаблон, який можна налагодити з вподобанням користувача системи. Шаблон буде відправлений користувачу з даними та стилями, які налаштував користувач системи.

Використання сервісу оплати також проходить через вбудовані івенти Laravel. Розробнику лише потрібно обрати сервіс оплати, який хоче використовувати та налаштувати його у середовищу Laravel з усіма API ключами системи оплати. Використання сервісу composer-пакетом слугує як callback фуккція, яка повертає результат виконання сервісом. Розробник може сам регулювати тип, що повертається.

2.3 Діаграма діяльності (Sequence diagram)

Діаграма діяльності – схожа на діаграму активності, але має більш точне відображення роботи системи за рахунок розширених можливостей у вигляді типізованих елементів та систем. Діаграма такого типу відображає обмін даними між сутностями або об'єктами, події, які ініційовані між сутностями та обмеження, які накладені між цими сутностями.

У діаграмах такого типу присутні основні елементи (рис. 2.5):

- **актор** – виконує роль учасника процесу;
- **клас-розмежувач** – відображає сутність, яка відділяє систему від зовнішнього представлення, наприклад інтерфейс користувача;

- **клас-контролер** – виконує функцію активного елементу, який обробляє дії, наприклад сервер;
- **клас-сутність** – бізнес-об’єкт БД або системи.

Кожен з цих елементів відображається зверху діаграми, але такі елементи як клас-сутність мають змогу утворюватись динамічно та відображатись там, де потрібно об’єкту бути створеним.

За допомогою вертикальних ліній, які відходять від перерахованих елементів, зображена паралельна дія кожної сутності та їх життя протягом процесу роботи системи.

Рисунок 2.5 – Позначення основних компонентів діаграми

Одним за найважливішим компонентом діаграми послідовності є **повідомлення** (рис. 2.6), яке відображається *стрілкою* та *лінією*, яка відходить від одного процесу та закінчується на початковій точці іншого процесу. Стрілка може бути як заповнена так і не заповнена. В першому випадку така стрілка буде вказувати на *синхронну (synchCall)* дію, як в той же час не заповнена стрілка буде повідомляти що дія є *асинхронною (asynchCall)*, тобто виконуватись в той же час що й інші дії потоку. Також повідомлення може бути таких типів: *повідомлення з відповіддю (reply)*, *втрачене повідомлення (lost)* та *знайдене повідомлення (found)*. Кожна стрілка має свій напис, який свідчить про певне виконання певної дії. *Повідомлення з відповіддю* – позначається на діаграмі як стрілка з пунктирною лінією. Таке повідомлення означає, що такий тип має змістовну частину, яке сигналізує про відповідь певного методу чи дії. *Втрачене повідомлення* – тип повідомлення, яке не має приймача. Для такого повідомлення не існує подія прийому, але присутня подія передачі. *Знайдене*

повідомлення – тип повідомлення, який схожий на *втрачене повідомлення*, але навпаки: присутня подія прийому та відсутня подія передачі.

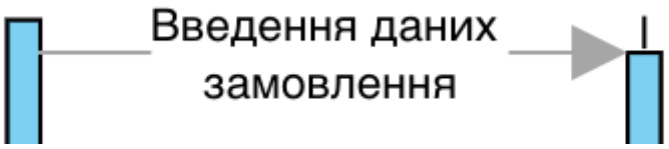

Рисунок 2.6 – Приклад повідомлення

Кожне повідомлення підв’язане до прямокутників, які знаходяться впродовж вертикальної лінії, яка відходить від базових елементів діаграми послідовності. Таким чином за допомогою таких прямокутників на діаграмі відображається взаємодія між елементами діаграми та процесами.

В розробці діаграми повідомлення допомагають **операнди**, які розширюють розуміння та сприйняття діаграми. За допомогою операндів можна гнучко описати діаграму. Операнди відображаються на діаграмі у вигляді прямокутника з написом у лівому верхньому кутку в іншому прямокутнику. Деякі операнди можут мати всередині фрагменти, які поділяються пунктирну лінією. Приклад операнда зображено на рисунку 2.7.

Бувають такі типи операндів:

- **Alt**;
- **Opt**;
- **Par**;
- **loop**;
- **region**;
- **Neg**;
- **ref**;
- **Sd**.

Операнд **Alt** – операнд, який приймає в себе фрагменти, які в свою чергу є альтернативними. Виконується той фрагмент у якого умова є істинною. Такий операнд може мати декілька альтернативних фрагментів. Умова вказується в квадратних дужках та в рамках фрагменту до якого ця умова має дію.

Операнд **Opt** – те ж саме що й **Alt**, але має лише один фрагмент опції та виконується тільки тоді, коли умова є істинною. Істина вказується так само як і в орпенаді **Alt**.

Par – використовується тоді, коли потрібне паралельне виконання всіх фрагментів. Таких фрагментів може бути безліч.

loop – операнд циклу, який виконується певну кількість разів або поки умова цикла не буде істинною. Кількість ітерацій вказується на діаграмі поруч з ім’ям операнда, а умова так само як і в опернаді **Alt**.

region – операнд, який повідомляє, що фрагмент може бути виконан лише один раз. Ніяких додаткових позначень не потребує.

Neg – операнд, який повідомляє, що така взаємодія не є вірною. Так само як і **region** додаткових позначень або параметрів не потребує.

ref – операнд посилання, який посилається на іншу діаграму або інший фрейм. В якості параметру у квадратних дужках вказується діаграма або фрейм.

Sd – операнд, який використовується для опису всієї діаграми послідовності. Потребується, якщо в іншій діаграмі присутнє посилання на дану діаграму.

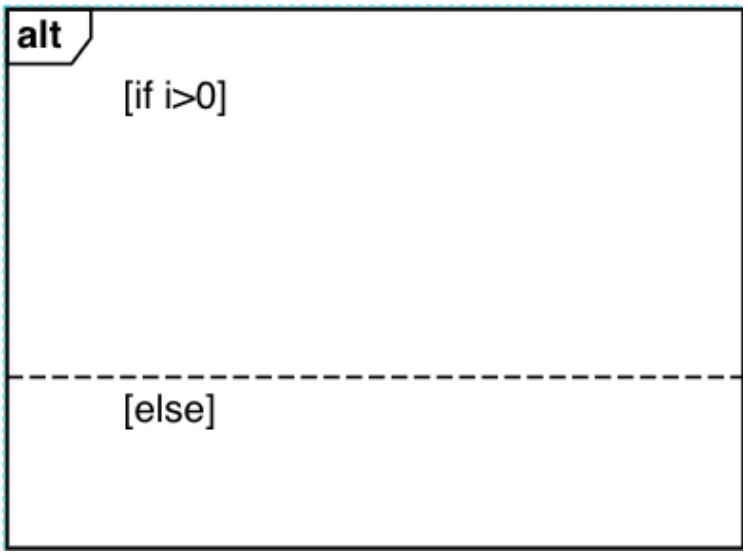

Рисунок 2.7 – Приклад операнду

В діаграмі послідовності до розроблюваного ПЗ composer-пакету Laravel для розробки Інтнернет-магазину (рис. 2.8) composer-пакет виступає частиною

діаграми послідовності Інтернет-магазину, а саме додатковою оболонкою до серверної частини сайту. У діаграмі присутні декілька важливих елементів для опису системи сайту: **клієнт системи**, **форма замовлення**, **сервер**, **composer-пакет**, **замовлення** та **адміністратор**. Кожен з цих елементів виконує абстрактну модель поведінки моделей на сайті.

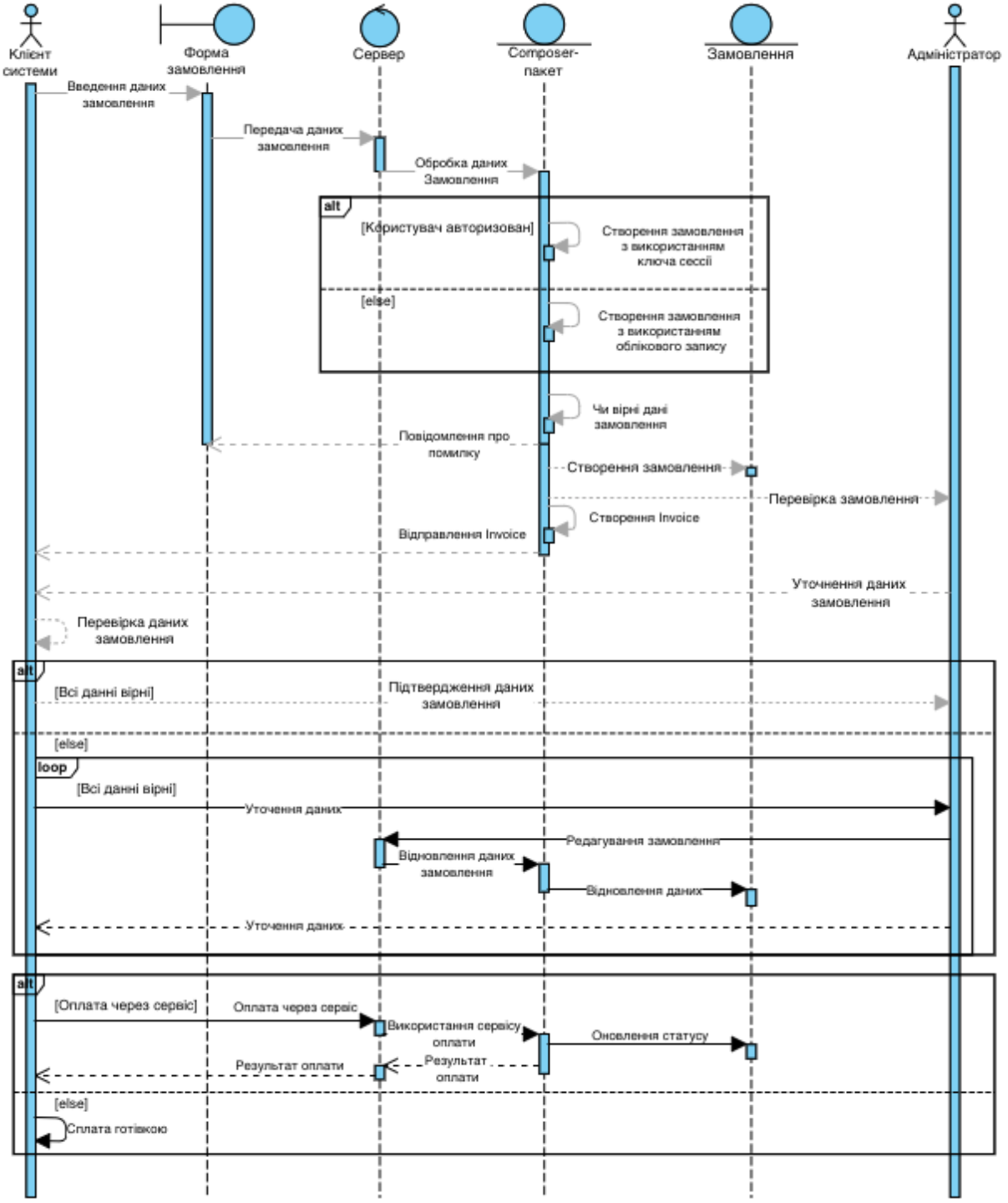

Рисунок 2.8 – Діаграма послідовності

Клієнт системи та ***адміністратор*** виконують роль акторів. В даному випадку актори є реальними людьми, які є частиною даної системи.

Форма замовлення – виконує роль **класу-розмежувача**, який відділяє зовнішню оболонку між ***клієнтом системи*** та ***сервером***.

Сервер – **клас-контролер**, який відображає сутність серверної частини системи.

Composer-пакет – описує **клас-сутність** серверної частини системи.

Замовлення – **клас-сутність** системи.

В даній діаграмі використовуються такі операнди як **Alt** та **loop**. В першому операнді **Alt** описується робота ***composer-пакету*** з даними користувача та що потрібно роботи у різних ситуаціях. Якщо користувач вже має прив’язаний обліковий запис, то пакет буде використовувати саме обліковий записа, в іншому випадку буде використовувати ключ сесії. Все це можна налаштувати через конфігураційний файл пакету. У другому операнді **Alt** присутній операнд циклу **loop**, який виконується у разі, якщо клієнт системи не згоден з даними замовлення. Ітерація операнду **loop** відбувається до того моменту поки клієнт системи не буде задоволений деталями замовлення.

Головну роль даної системи виконує composer-пакет. Саме він виконує основну логіку системи та піклується про те, щоб користувач та адміністратор отримали саме ті дані, які повинні бути присутні в замовленні. Composer-пакет обробляє форму замовлення після чого створює новий об’єкт замовлення, який підв’язаний до ключа сесії або облікового запису користувача (рис. 2.9). Далі composer-пакет повідомляє клієнта про те, що замовлення на сайті проведено успішно через створення та відправлення Invoice`у на електронну адресу клієнта. В той же час адміністратор отримує відповідну форму на електронну адресу та звіряє чи всі дані введені корректно (рис. 2.10). Якщо у адміністратора є сумніви з приводу даних замовлення, то він зв’язується з клієнтом та редагує замовлення. Редагування замовлення теж проходить через пакет composer. Після

того як користувач погоджується з правильністю введених даних, він обирає спосіб оплати (рис. 2.11). У випадку, якщо користувач обирає спосіб оплати через обраний сервіс, то composer-пакет виконує процес оплати сервісом з результатом виконання. Після процесу оплати composer-пакет оновлює статус замовлення. Якщо оплата була не вдала, то замовлення отримує відповідний статус та попередить про це адміністратора.

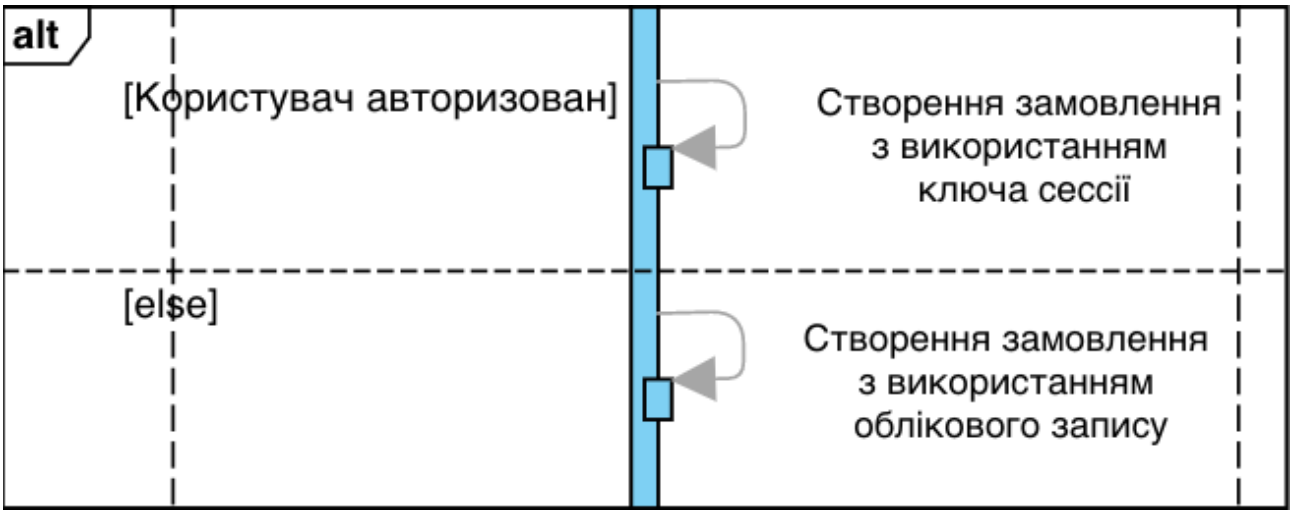

Рисунок 2.9 – Операнд **Alt**

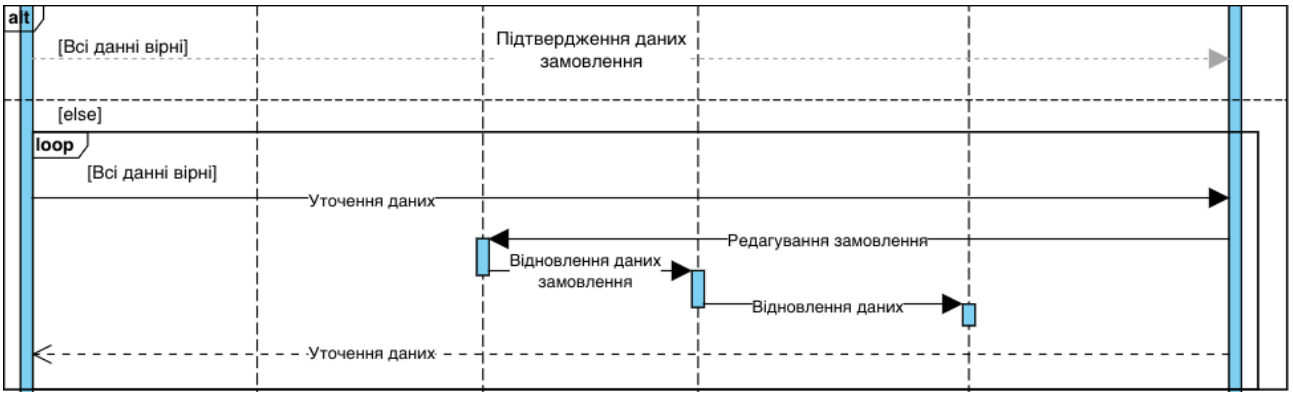

Рисунок 2.10 – Операнд **Alt** з використанням операнду **loop**

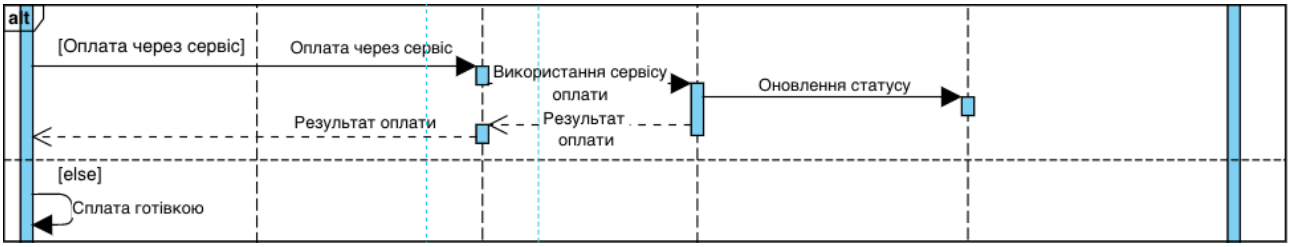

Рисунок 2.11 – Операнд **Alt** з вибором клієнт

Висновки до розділу 2

Створено діаграму прецедентів за допомогою якої було проведено аналіз прецедентів та акторів системи під яку розробляється composer-пакет Laravel для розробки Інтернет-магазину. Розглянуто composer-пакет як частину передбачуваної системи. Розглянуто роль кожного актора в передбачуваній системі з використанням composer-пакету.

Описано composer-пакет як додаткову оболонку до системи, яка буде використовуватись іншими розробниками та створено діаграму активності в якій відображено сумісну роботу передбачуваної системи та composer-пакету для розробки Інтернет-магазину.

Виявлено основні тригери системи composer-пакету та саме яке місце повинен займати пакет та створено діаграму діяльності в якій виявлено кругообіг дій в якому присутній composer-пакет Laravel для розробки Інтернет-магазину.

3 ВИБІР ЗАСОБІВ РОЗРОБКИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ COMPOSER-ПАКЕТУ

3.1 Вибір технологій та засобів розробки Composer-пакету для розробки Інтернет-магазину

Для розробки ПЗ composer-пакету для розробки Інтернет магазину були задіяні такі технології:

- PHP;
- Composer;
- PhpStorm;
- Laravel;
- шаблонізатор blade;
- JavaScript;
- MVC;
- Session;
- Cookies;
- Postman.

Також для зручної та ефектиктивної розробки було застосовано два composer-пакети для Laravel: **laravel-debugbar** та **laravel-ide-helper**.

PHP – мова програмування, яка найчастіше використовується у вебпрограмуванні. У жаргоні розшифровується як Personal Home Page. Дана мова програмування є скриптовою та інтерпертатором мови є вебсервер. PHP відноситься до типу об'єктно-орієнтованих та імперативних мов програмування. Для розробки обрано саме PHP, бо за статистикою[10] 77.5% з усіх сайтів в світі складають сайти написані саме на цій мові програмування. Також було обрано версію PHP 7.4, бо саме з усіх сайтів дана версія складає 71.6%.

PhpStorm – IDE для розробки на мові програмування PHP. Дане програмне середовище має тріал-версію на 30 днів або ж безкоштовне використання для учнів університетів та інших навчальних закладів. Всі інші

тарифні плани присутні на офіційному сайті. Для практичної частини КРБ було придбано версію для освітніх закладів з статусом студента. Саме цю IDE було рекомендовано освітнім закладом та керівником КРБ для більш ефективної розробки ПЗ composer-пакету для розробки сайтів типу ecommerce.

JavaScript – мова програмування, яка використовується переважно у вебпрограмуванні. Мова програмування відноситься до об'єктно-орієнтованої, але лише з частини прототипування. Саме дана мова програмування підходить для динамічної зміни сторінки. JavaScript буде використовуватись у ПЗ composer-пакету лише для динамічних методів, які визначають регіон користувача.

Postman – інструмент, який допомагає гнучко робити операції з API. Даний інструмент дає розширений функціонал для відлагодження будь-яких API з будь-якими даними та будь-якими типами запиту (GET, PATCH/PUT, POST, DELETE, ACTION, тощо). Даний інструмент було обрано за рекомендацію популярного сайту-порталу Katalon[11]. Даний сайт виділяє чотири найголовніші переваги Postman:

- легке у використанні;
- спрощений API;
- надійні можливості тестування;
- інтеграція та спільна робота.

Шаблонізатор blade – шаблонізатор, який використовується за замовчуванням у проєктах Laravel. Саме за допомогою даного шаблонізатору Laravel може гнучко та ефективно передавати дані у шаблони з вбудованими функціями, які у процесі конвертуються у стандартний PHP. Також за допомогою даного шаблонізатору генеруються листи для користувачів. У симбіозі з розроблюваним ПЗ composer-пакету надається можливість легко створювати Invoice`и замовлень та генерувати без зайвих проблем. Якщо розробнику не вистачає функціоналу Laravel, то у composer-пакету є

запропоновані методи роботи з шаблонізатором. Всі методи описані в документації.

Session(сесія) – технологія, яка допомагає відстежити запити від конкретного браузера та кешувати дані на певний період часу. Таким чином всі дані між сторінками зберігаються. Дані зберігаються на сервірній частині застосунку. Laravel використовує окремий автоматично згенерований ключ(знаходиться у файлі «**.env**») застосунку, який допомагає генерувати особистий ключ(token) до кожної сесії, який підв'язується до IP-адреси, версії браузеру та версії операційної системи.

Cookies – схоже з **session**, але всі дані зберігаються з боку клієнта, тобто у браузері.

MVC – паттерн проєктування, який тісно пов'язаний з розробкою вебзастосунків та використовується в проєктах Laravel. Даний патерн поділяє три парадигми: модель(Model), представлення(View) та контроллер(Controller). У composer-пакет переважно використовується тільки модель та контролери, але це не означає що не можна використовувати представлення.

laravel-debugbar – composer-пакет для Laravel, який використовується для зручного debug-процесу системи. Для роботи пакету треба проєкт змінити на режим DEBUG. Для цього у відповідному файлі «**.env**» змінити ключ APP_DEBUG на значення **true**. При запущені серверу сайту можна побачити відображення панелі (рис. 3.1) в якій присутні вкладки з відповідними даними.

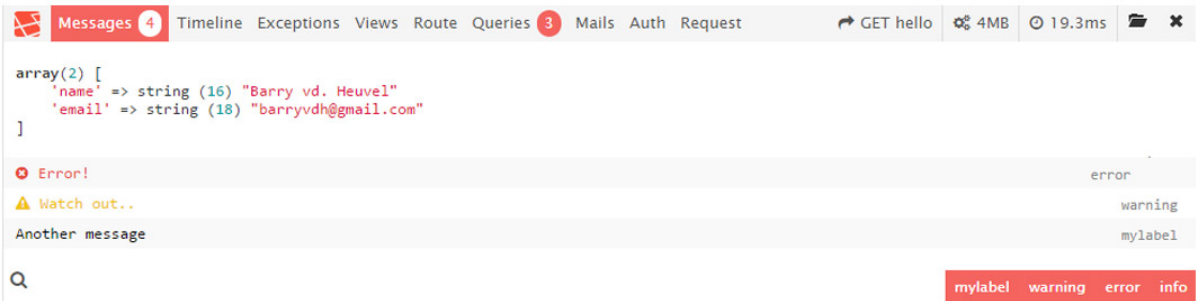

Рисунок 3.1 – Панель debugbar

laravel-ide-helper – composer-пакет для Laravel, який автоматично генерує коментарі до кожного файлу, який має розширення *.php* та написаний згідно

стандартів PSR-4. Даний пакет допомагає IDE зрозуміти чи присутні певні поля та методи з певною сигнатурою в окремих класах. Також допомагає позбутися помилки у IDE, які пов’язані з відсутністю певних методів у класі.

3.2 Опис функціоналу Composer-пакету

Найголовнішими та базовими функціями composer-пакету для розробки Інтернет-магазину є *маніпулювання кошиком авторизованого та не авторизованого користувача*, *обробка даних замовлення як авторизованого та і не авторизованого користувача* та *відправлення Invoice`у клієнту та адміністратору системи.*

Маніпулювання кошиком авторизованого та не авторизованого користувача відбувається автоматично за допомогою вбудованих функцій в composer-пакеті. Обробка всіх даних проходить через **session** або **cookies**. Яка система збереження даних у застосунку буде використовуватись вирішує розробник системи. Саме як буде вести себе composer-пакет в різних режимах визначається в конфігураційному файлі, який генерується в процесі встановлення пакету згідно документації.

Якщо користувач ***авторизован***, то обробка даних буде відбуватись згідно облікового запису, який під'єднаний до **session**, яка зберігається у базі даних. Кожен раз коли користувач потрапляє на сторінку відбувається запит на сервер з перевіркою наявністю кошика користувача. У випадку якщо немає кошику система створює новий об’єкт, який під’єднується до облікового запису.

Якщо користувач ***не авторизован,*** тоді системи буде використовувати **session**, який під’єднано до певного браузера. Всі інші операції аналогічні з сценарієм ***авторизованого*** користувача.

В обох варіантах передбачена ситуація коли дані користувача зберігаються через **cookies**. В цьому випадку буде використовуватись *token* користувача, який генерує автоматично Laravel.

Логіка кошика передбачає розширені методи, які допомагають з легкістю фільтрувати, редагувати та робити інші будь-які операції. Кошик передбачає можливі події клієнта системи та покриває всі можливі варіанти використання системи. При *додаванні товару до кошику* аналізуються наступні параметри: чи є в наявності товар такого типу з заданими параметрами, чи наявний у кошику товар такого типу з відповідними параметрами. У першому випадку перевіряється у базі даних наявність товару, та у випадку якщо його не має composer-пакет робить виключення, яке користувач системи повинен обробити у вигляді помилки на сторінці або іншим шляхом. Якщо товар є в наявності, то composer-пакет додає до кошику товар. У другому випадку composer-пакет перевіряє присутність однотипного товару з параметрами, які повинні збігатись. Якщо такого товару не знайдено, то системи додає новий товар до кошику. Налагодити однотипність товарів за певними товарами можна у класі, який наслідується від базового класу елементу кошику. Видалення кошику як з боку клієнта так і адміністратора відбувається у форматі *soft delete*, тобто всі дані зберігаються в базі даних, але не відображаються у користувача та адміністратора. Таким чином присутня можливість повернути дані. Кошик має методи підрахунку з використанням умов підрахування як *знижка* та *доставка.*

Обробка даних замовлення як авторизованого та і не авторизованого користувача відбувається аналогічно з кошиком. Всі дані підв’язуються з вже створеного кошика та прив’язуються до клієнта системи. Система замовлень має окрему *систему дискаунтів* та *систему обчислення ціни доставки*. *Система дискаунтів* та промокодів повністю налаштовується розробником системи. Composer-пакет лише додає до бази відповідні ключі за якими буде проходити знижка. Знижку можна вказувати як у відсотках так і у числах. Система обчислення ціни доставки теж перепадає на розробника, але composer-пакет має методи, які допоможуть в розробці системи.

Відправлення Invoice`у клієнту та адміністратору системи відбувається автоматично завдяки прописаних методів, які побудовані на

івентах Laravel. За допомогою шаблонізатору **blade** розробник може стилізувати та налагодити відображення Invoice. Шаблон blade автоматично генерується при встановленні composer-пакету згідно документації. Шаблон відправляється на поштову скриньку як клієнта так і адміністратора системи. У разі якщо присутні помилки, то адміністратор може надіслати редагований(остаточний) варіант списку замовлення.

3.3 Діаграма класів (Class diagram)

Діаграма класів (Class diagram) – UML-діаграма, яка відображає взаємозв'язок класів, їх ієрархію та поля з методами, які присутні в цих класах.

Всі класи, які присутні на діаграмі, створені згідно екосистеми Laravel. Найголовнішим класом у даній діаграмі є клас **PackageFacade** (рис. 3.2). Даний клас є похідним від вбудованого класу в середу Laravel **Facade**. Клас Facade є постачальником всіх провайдерів, які у свою чергу постачають Laravel`у класи, конфігураційні файли, звичайні файли, тощо. Даний клас має лише масив провайдерів, які автоматично будуть під'єднані до проєкту через базові методи.

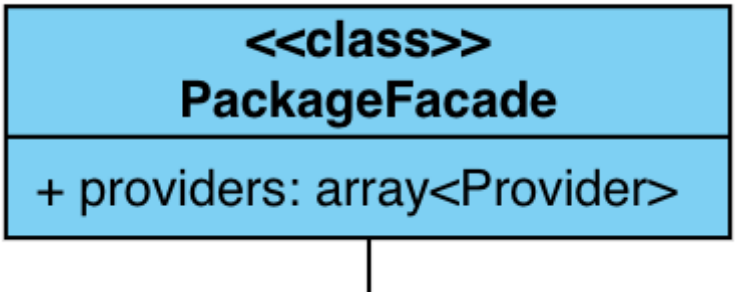

Рисунок 3.2 – Клас **PackageFacade**

Діаграма класів має декілька провайдерів:

- BusketProvider;
- OrderProvider;
- InvoiceProvider;

Всі ці класи є похідними від класу **Provider** (рис. 3.3). Даний клас має два поля, які є композицією класів **Authenticator** та **Settings**.

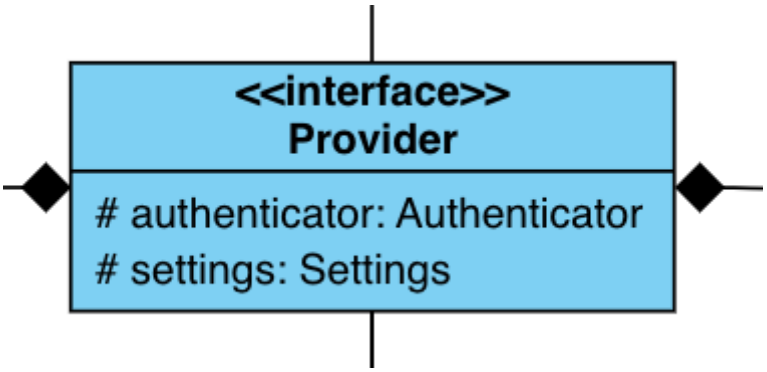

Рисунок 3.3 – Клас **Provider**

BusketProvider (рис. 3.4) – похідний клас від вбудованого класу **Provider**, який реалізує всі методи та всю логіку роботи системи. Даний клас має поле **buskets**, яке має тип класу **Basket** та метод **getBuskets**, який повертає як масив класів так і певний екземпляр класу за певним id. В логіку знаходження потрібного кошика входить використання **Authenticator**, який відслідковує входження кожного кошика за певним користувачем.

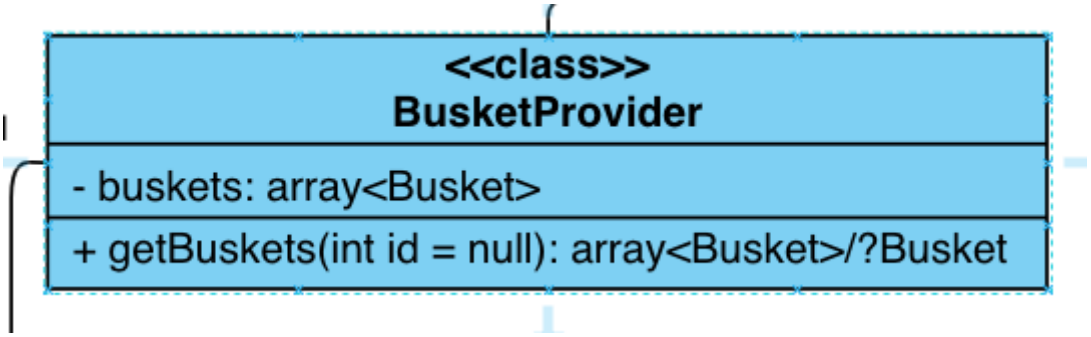

Рисунок 3.4 – Клас ***BusketProvider***

OrderProvider (рис. 3.5) – похідний клас від вбудованого класу **Provider**, який має поле **orders,** яке є масивом класів типу **Order**, та групу методів, які повністю відповідають сигнатурі Rest API контролера. Даний провайдер використовує клас **Authenticator** для логіки знаходження замовлень.

InvoiceProvider (рис. 3.6) – похідний клас від вбудованого класу **Provider**, який має поле **invoices,** яке є масивом класів типу **Invoice**, та групу методів, які повністю відповідають сигнатурі Rest API контролера. Даний клас для логіки знаходження інвойсів використовує клас **Authenticator**.

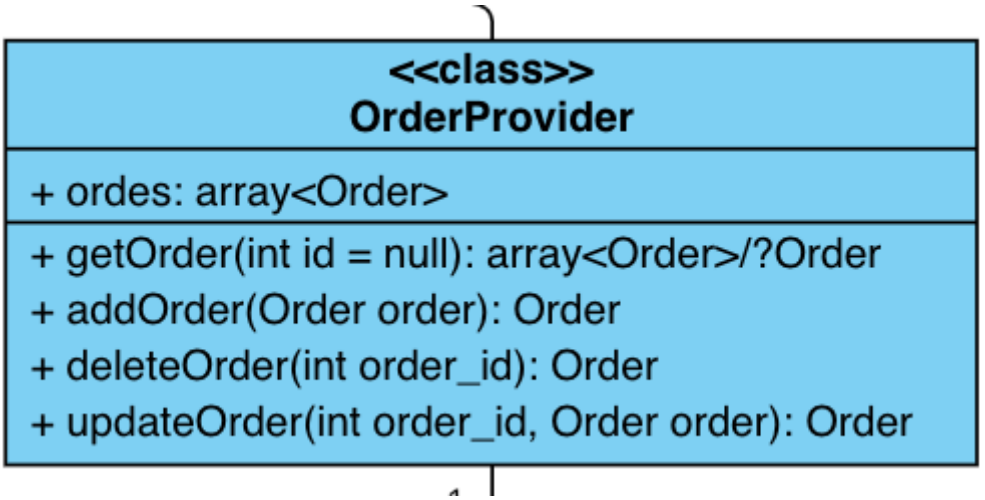

Рисунок 3.5 – Клас ***OrderProvider***

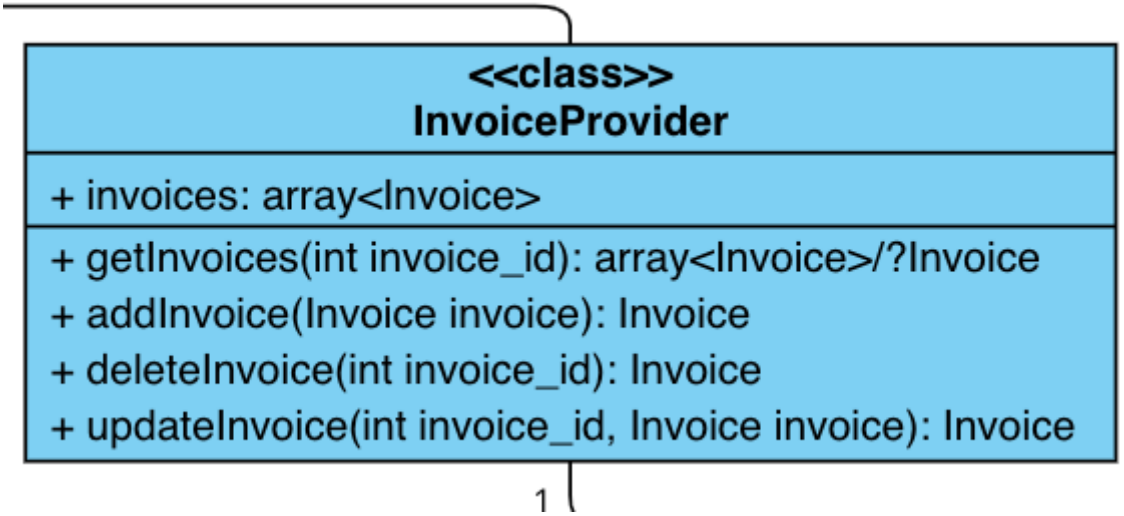

Рисунок 3.6 – Клас ***InvoiceProvider***

Одним за найважливіших класів є клас ***Settings*** (рис. 3.7). Даний клас виконує роль гетера конфігураційного файлу системи composer-пакету. Для того щоб отримати певний параметр конфігураційного файлу, треба скористатись методом **getSettings**. В якості параметру даний метод приймає строкове значення ключа, який потрібно отримати з конфігураційного файлу системи. Якщо такого ключа не знайдено, то система за допомогою виключення повідомляє розробнику про відсутність ключа.

На діаграмі також представлені плоскі моделі, які мають лише поля та не мають власних методів обробки даних. Дані класи є прямим відображенням того як саме зберігаються дані у базі даних. Всі методи, які присутні у цих

класах є методами, які присутні у базовому класі **Eloquent**, який присутній в базовому пакеті Laravel та має всі перевизначенні методи для роботи з базою даних типу MySQL.

До плоских моделей діаграми відносяться такі класи: **Product** (рис. 3.7), **ProductAttribute** (рис. 3.8), **Discount** (рис. 3.9), **ShippingType** (рис. 3.10), **OrderAttribute** (рис. 3.12), **InvoiceAttribute** (рис. 3.13), **User** та **Session**.

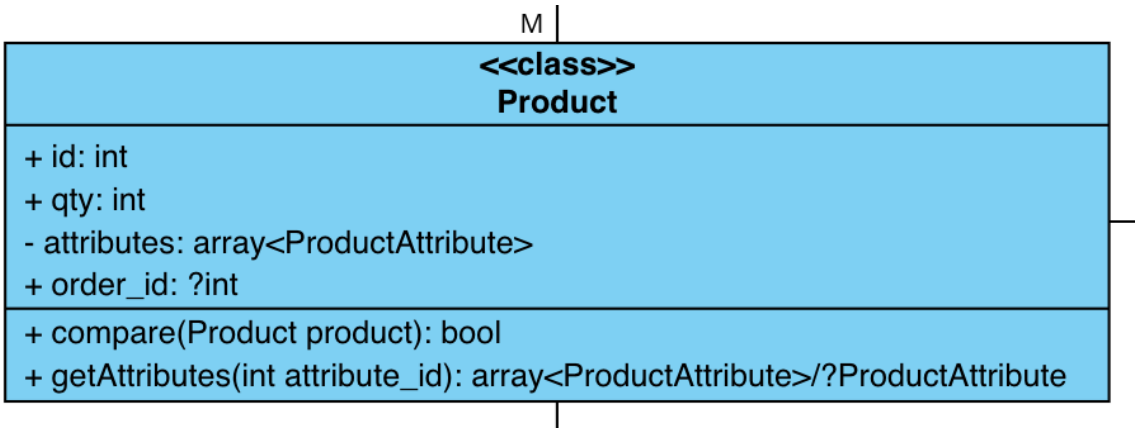

Рисунок 3.7 – Клас **Product**

Product – клас, який наслідується від стандартного класу Laravel проєкту **Eloquent**. Даний клас являє собоюплоску модель відображення даних продукту в базі даних. Клас має окремий масив атрибутів товару, які мають тип класу ***ProductAttribute***.

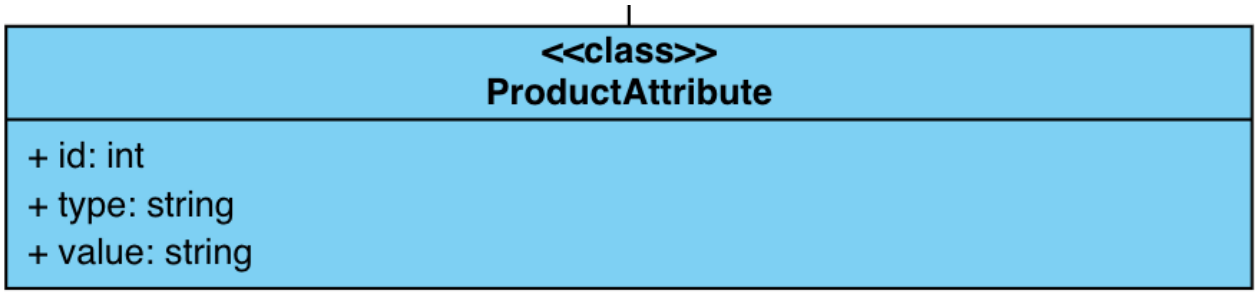

Рисунок 3.8 – Клас **ProductAttribute**

ProductAttribute – клас, який наслідується від стандартного класу Laravel проєкту **Eloquent**. Клас відображає плоску модель таблиці бази даних та використовується як значення полів класу ***Product***. За допомогою ключового слова *Attribute* даний клас можна використовувати з іншими пакетами, які

працюють з локалью проєкту, тобто можна використовувати власні перекладити кожного поля. Наприклад, можна використовувати composer-пакет **spatie/laravel-translatable**.

Рисунок 3.9 – Клас **Discount**

Discount – клас, який наслідується від стандартного класу Laravel проєкту **Eloquent**. Клас застосовується за для відображення плоскої моделі бази даних та виступає у ролі купонів, промокодів, тощо. Використовується як окреме поле для класу **Order**.

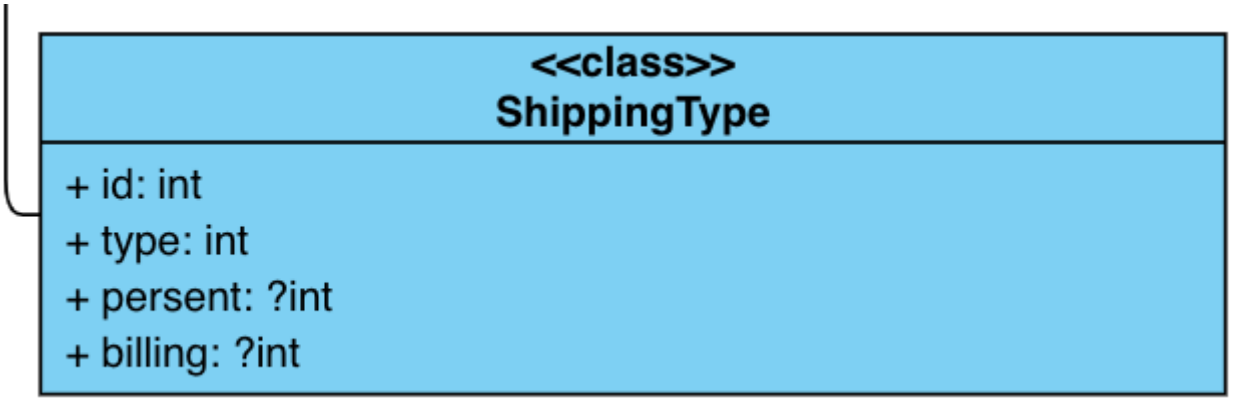

Рисунок 3.10 – Клас **ShippingType**

ShippingType – клас, який як і інші плоскі моделі діаграми, наслідується від класу **Eloquent**, який є базовим для всіх проєктів Laravel. Слугує даний клас для визначення типу доставки та його вартості. Вартість доставки може вказуватись як у вигляді відсотка так і у вигляді значення. Користувач системи сам вирішує яким чином повинно проходити обчислення кожного типу доставки. Система автоматично аналізує дані для підрахунку.

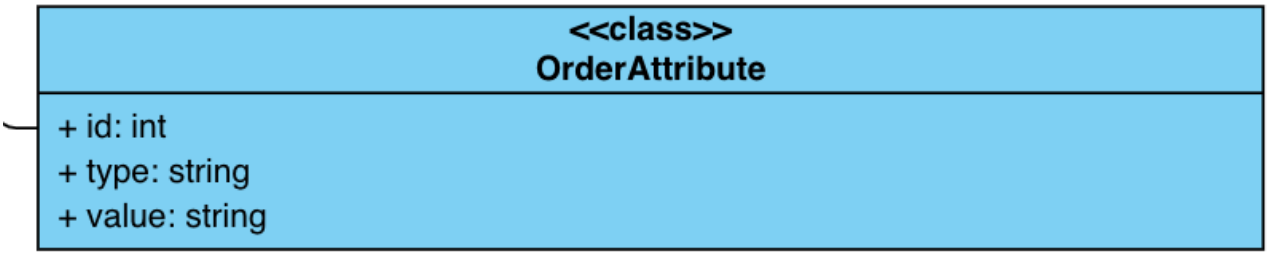

Рисунок 3.11 – Клас **OrderAttribute**

OrderAttribute – клас, який переважно має всі характеристики класу ***ProductAttribute***, але використовується як поле для класу **Order**.

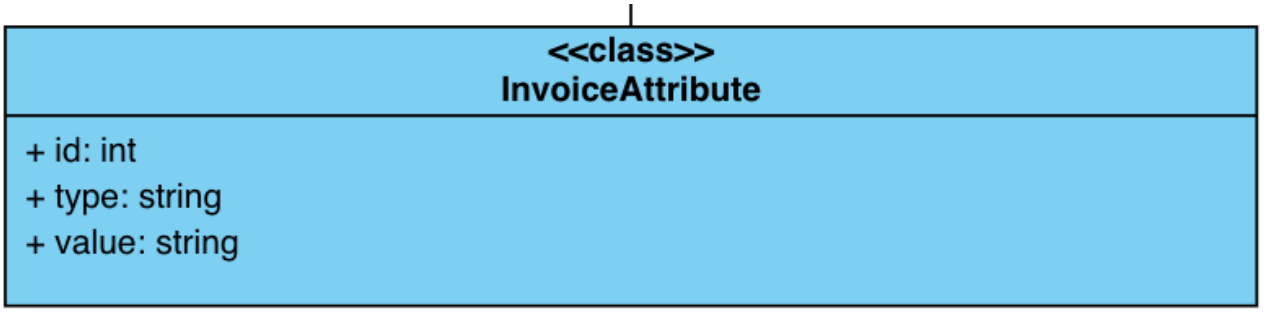

Рисунок 3.12 – Клас **InvoiceAttribute**

InvoiceAttribute – клас, який має всі характеристики як ***OrderAttribute*** та ***ProductAttribute***, але використовується для поля у якості масиву у класі **Invoice**.

Усі інші класи, які представлені як плоскі моделі є базовими для кожного проєкту Laravel, який використовує базу даних як місце для зберігання даних сесії.

Інші класи у діаграмі класів мають власну реалізацію з перевизначиними методами. Набір методів зумовлений стандартному ресурсному (RESTful) контролеру, тобто усі методи, які відповідають типовому CRUD.

Такі класи як **Authenticator** та **Settings** слугують для забезпечення певних даних системі. Клас **Authenticator** відповідає за постачання даних користувача та типу авторизації у системі. Клас **Settings** постачає усі дані з конфігураційних файлів системи. Якщо певні ключі відсутні або є не зміними, то передбачається автоматичний вибір певного ключа.

На діаграмі класів (рис. 3.13) присутні всі класи, які відповідають системі, що розробляється, у сесійному режимі. В іншому випадку використання класу **Session** не є обов'язковим.

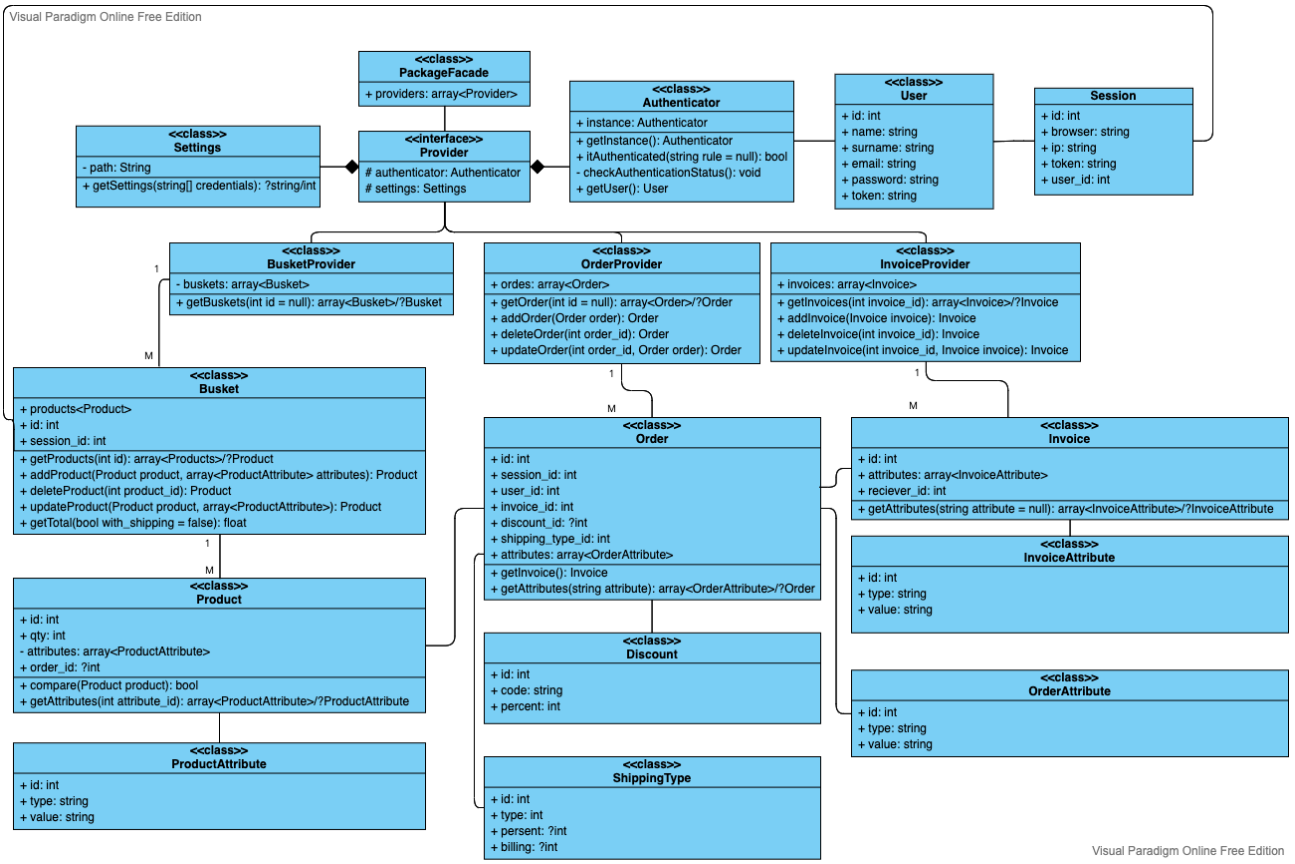

Рисунок 3.7 – Діаграма класів

У системі використовується більша кількість класів ніж присутня у діаграмі класів. Це зумовлено тим, що велика кількість моделей у системі є плоскими та майже не мають методів. Такі класи слугують лише для додаткової інформації, яка не є обов'язковою задля роботи системи. Також у системі присутні класи, які слугують для логування системи та всі дані виводяться у базовий файл «**laravel.log**» для логування у проєкті Laravel. У випадку коли логування не потрібне, користувач системи може вимкнути логування у конфігураційному файлі системи. У системі присутні власні класи *виключення*, які використовується при порушенні логіки системи або параметрів входу у системі. Дані виключення є дочірніми класами стандартного класу базового проєкту Laravel **Exception**. За допомогою перевизначених методів Laravel

автоматично відслідковує виключення. Якщо проєкт знаходиться в стані відлагодження, тоді Laravel автоматично виведе повідомлення виключення.

Висновки до розділу 3

Проведено аналіз та обґрунтовано вибір кожної технології для створення програмного забезпечення composer-пакету для створення сайтів типу ecommerce. Описано яким чином кожна технологія прискорює розробку та чому саме конкретна технологія присутня у composer-пакеті.

Докладно описано функціонал майбутнього програмного забезпечення composer-пакету для створення інтернет-магазинів. Описано яким чином прискорюється процес розробки за рахунок створеного функціоналу та вбудованих методів. Розглянуто крок дій системи у різноманітних випадках роботи системи, яка розробляється користувачем інтернет-магазином на Laravel разом з розроблюваним програмним забезпеченням.

Створено діаграму класів з поясненням ролі кожного класу у системі. Детально описано кожен клас програмного забезпечення composer-пакету для розробки інтернет-магазину. Описано ієрархію класів, зв’язок між класами та можливості розширення класів з додаванням нового функціоналу. Виявлено яким чином працює система з проєктом Laravel на базі класів, які є дочірніми базових класів стандартного проєкту Laravel.

4 ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ COMPOSER-ПАКЕТУ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ

4.1 Структура composer-пакету Laravel для розробки Інтернет-магазину

Для створення власного composer-пакету було використано документацію **GitLab**[12]. В документації описано кожен крок та перелік файлів, які повинні бути присутні для створення власного пакету та зареєструвати його у реєстрі пакетів.

Для того щоб створити власний пакет треба мати такий перелік файлів:

- **composer.json**;
- **.gitlab-ci.yml**.

Файл «**composer.json**» (рис. 4.1) є обов'язковим файлом для того, щоб створити власний пакет, на відміну від файлу «**.gitlab-ci.yml**» (рис. 4.2). Останній файл використовується лише тоді, коли реєстрація пакету буде проходити через сервіс **GitLab** з використанням **CI/CD**.

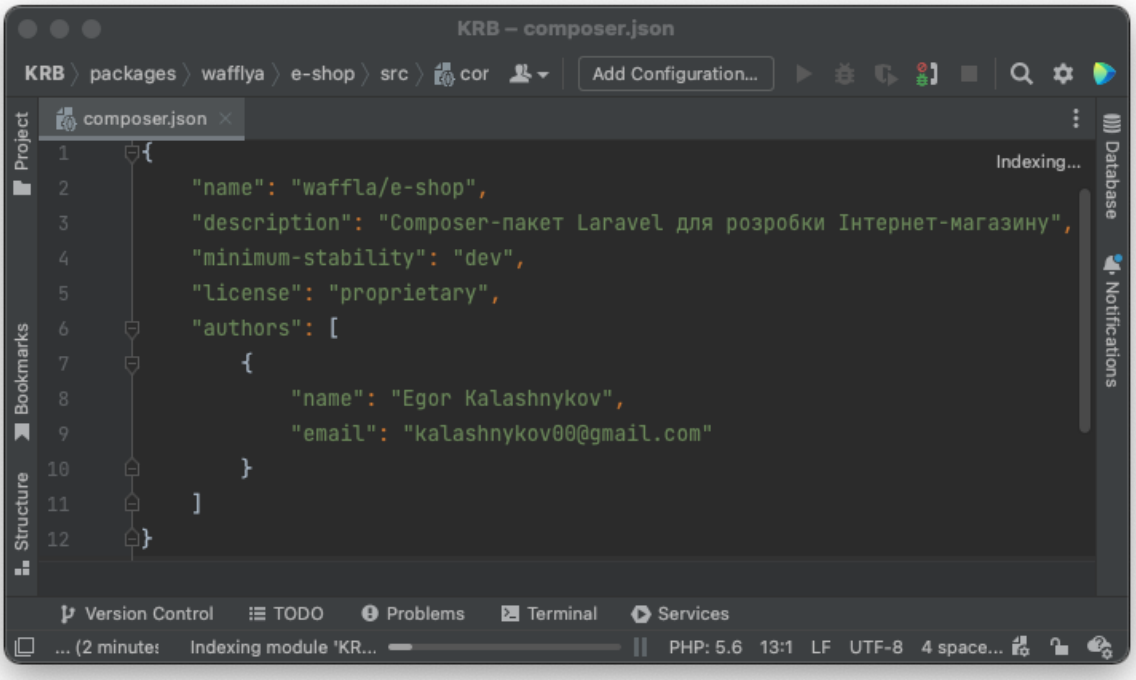

Рисунок 4.1 – Файлу **composer.json**

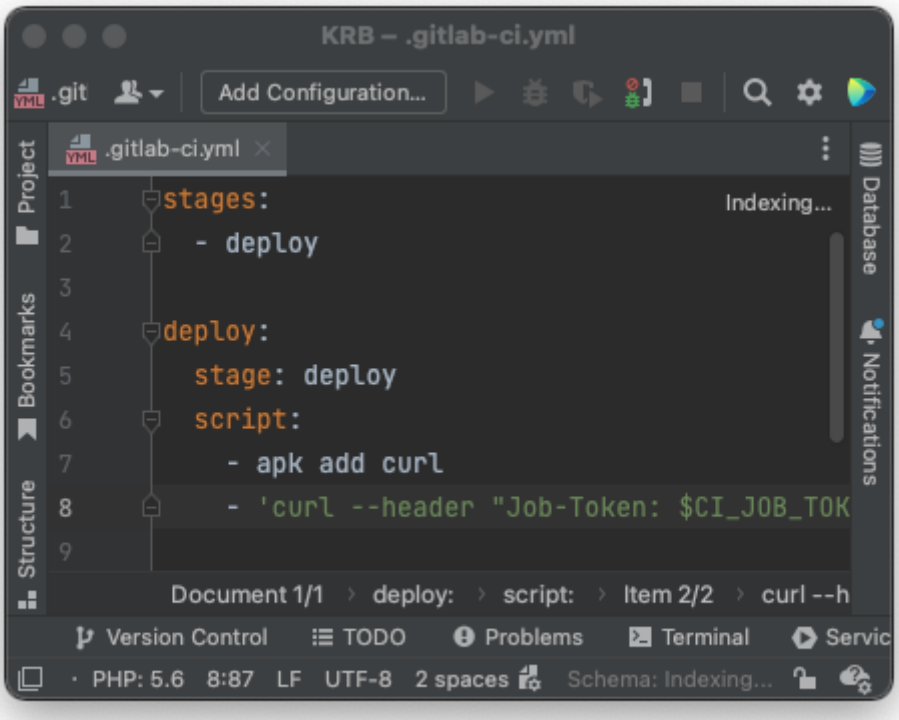

Рисунок 4.2 – Файл **.gitlab-ci.yml**

Laravel дозволяє легко створювати пакети власними засобами. Для цього треба створити папку **packages** у папці проєкту. Далі створити папку з ім’ям автора пакету та в цій папці створити папку з ім’ям пакету. Папка с ім’ям пакету потрібна містити папку **src**. В даній папці відбувається розробка пакету з усіма необхідними папками та конфігураційними файлами.

Для створюваного програмного забезпечення composer-пакету було створено файлову структуру (рис. 4.3). У даній файловій структурі присутні такі директорії:

- **packages;**
- **wafflya;**
- **e-shop;**
- **src;**
- **config;**
- **migrations;**

- **Models;**
- **views;**
- **Exceptions.**

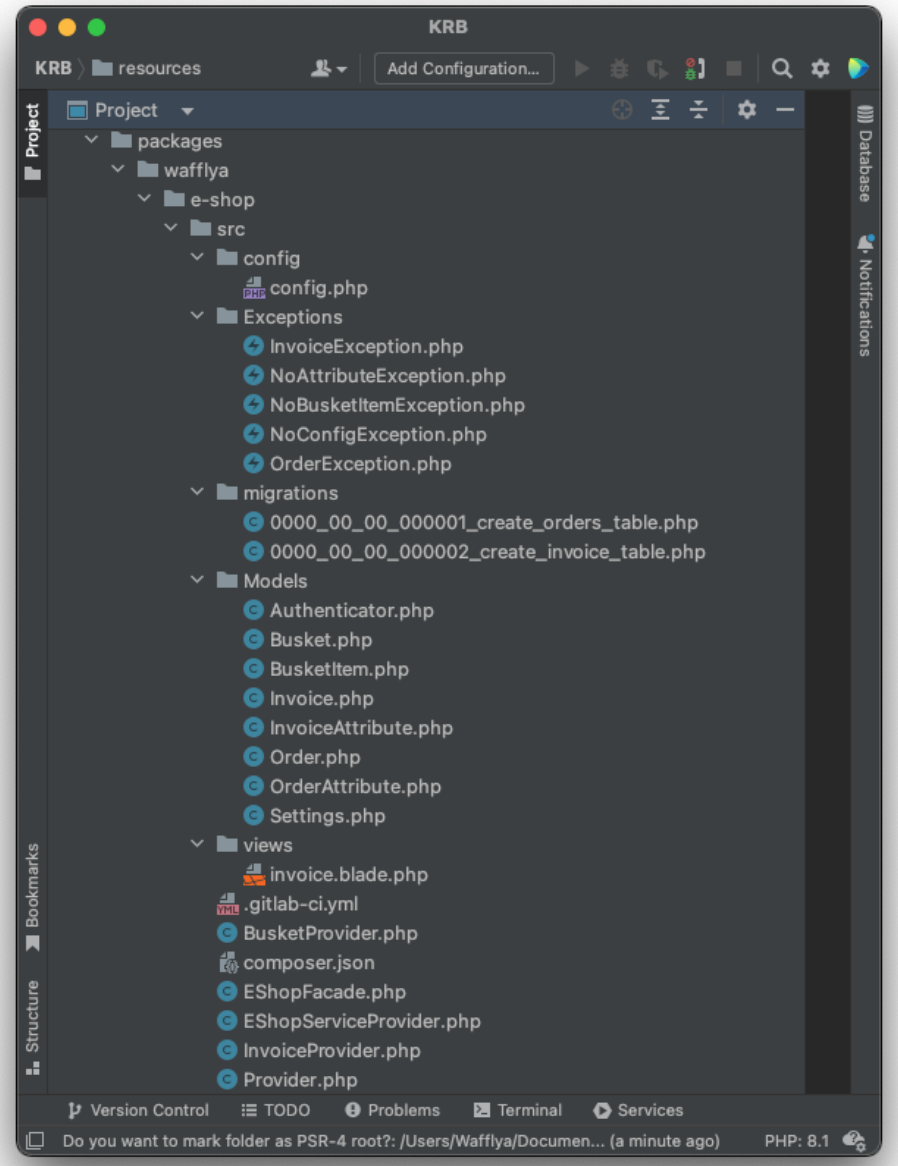

Рисунок 4.3 – Файлова структура пакету

Директорія ***packages*** – стандартна директорія, яка використовується у базових проєктах Laravel для створення власних пакетів. Дочірньою директорією є директорія ***wafflya***.

Директорія ***wafflya*** – директорія, яка має ім’я розробника пакету та зберігає всі дочірні каталоги як composer-пакети. Ім’я директорії використовується як параметр конфігураційного файлу кожного composer-пакету, тому для публікації дана директорія. Дочірньою директорією є дирекотрія ***e-shop***.

Директорія ***e-shop*** – директорія пакету, який буде публікуватись. Ім’я використовується в якості параметру конфігураційного файлу «**composer.json**». Дочірньою директорією є директорія ***src***.

Директорія ***src*** – директорія, яка використовується за замовчуванням для роботи composer-пакету. В даній директорії присутні усі файли та папки для роботи composer-пакету. Дочірніми директоріями є директорії: ***config***, ***Exceptions, migrations, Models, views***. У директорії присутні файли налагодження пакету для публікації «**composer.json**» та «**.gitlab-ci.yml**». Також у директроії присутні файли, які наслідуються від базових класів стандартного пакету Laravel та слугують для підключення та роботи з Laravel проєктом. Всі ці файли є php-файли з класами, які описані у діаграмі класів з 3 розділу КРБ.

Директорія ***config*** – директорія, яка використовується для публікації у Laravel проєкт усіх супутніх файлів конфігурації для роботи пакету з системою, яка присутня у Laravel проєкті.

Директорія ***Exceptions*** – директорія в якій присутні всі файли для роботи з виключеннями. Дані файли мають розширення ****.php*** та мають у собі класи, які є дочірніми класу **Exception**, який присутній у стандартному пакеті Laravel та відслідковуються системою відлагодження середовищем Laravel. Всі файли підтягуються до Laravel за рахунок composer та стандарту оформлення коду PSR-4.

Директорія ***migrations*** – директорія в якій присутні всі міграційні файли за стандартом Laravel. Дані файли автоматично не підтягуються як файли з директорії, тому файли підтягуються за допомогою artisan-команд.

Директорія ***Models*** – директорія, яка містить всі моделі пакету. За допомогою composer та стандарту написання коду PSR-4 файли автоматично підтягуються до проєкту. Моделі присутні у даній директорії є класами або плоскої моделі відображення даних у базі даних або класами, які задіяні у архітектурі системи.

Директорія ***views*** – директорія, яка зберігає всі представлення пакету, які будуть у свою чергу задіяні у проєкту Laravel. Представлення зберігається у вигляді файлу з розширенням ****.blade***. Також як і міграційні файли, файли представлення недоступні з пакету та мають бути опубліковані у проєкті Laravel за допомогою artisan-команд.

4.2 Інструкції користувача з установки та налаштування composer-пакету

Для того щоб встановити composer-пакет, треба скористатись командою, яка присутня у composer *composer require <ім'я_розробника>/<назва_пакету>*. У випадку composer-пакету Laravel для розробки Інтернет-магазину composer-команда виглядає як *composer require wafflya/e-shop*. Дану команду треба записувати у кореневій директорії проєкту Laravel. Після запуску команди почнеться встановлення пакету (рис. 4.4). Під час встановлення будуть одночасно встановленні інші пакети для роботи системи:

- barryvdh/laravel-ide-helper;
- barryvdh/reflection-docblock;
- composer/pcre;
- doctrine/cache;
- doctrine/dbal;
- doctrine/deprecations;
- doctrine/event-manager;
- psr/cache.

Кожен пакет має певний набір функцій та власні версії підтримки Laravel. Composer-пакет працює стабільно в системі проєкту Laravel версії не нижче ніж 7 версія.

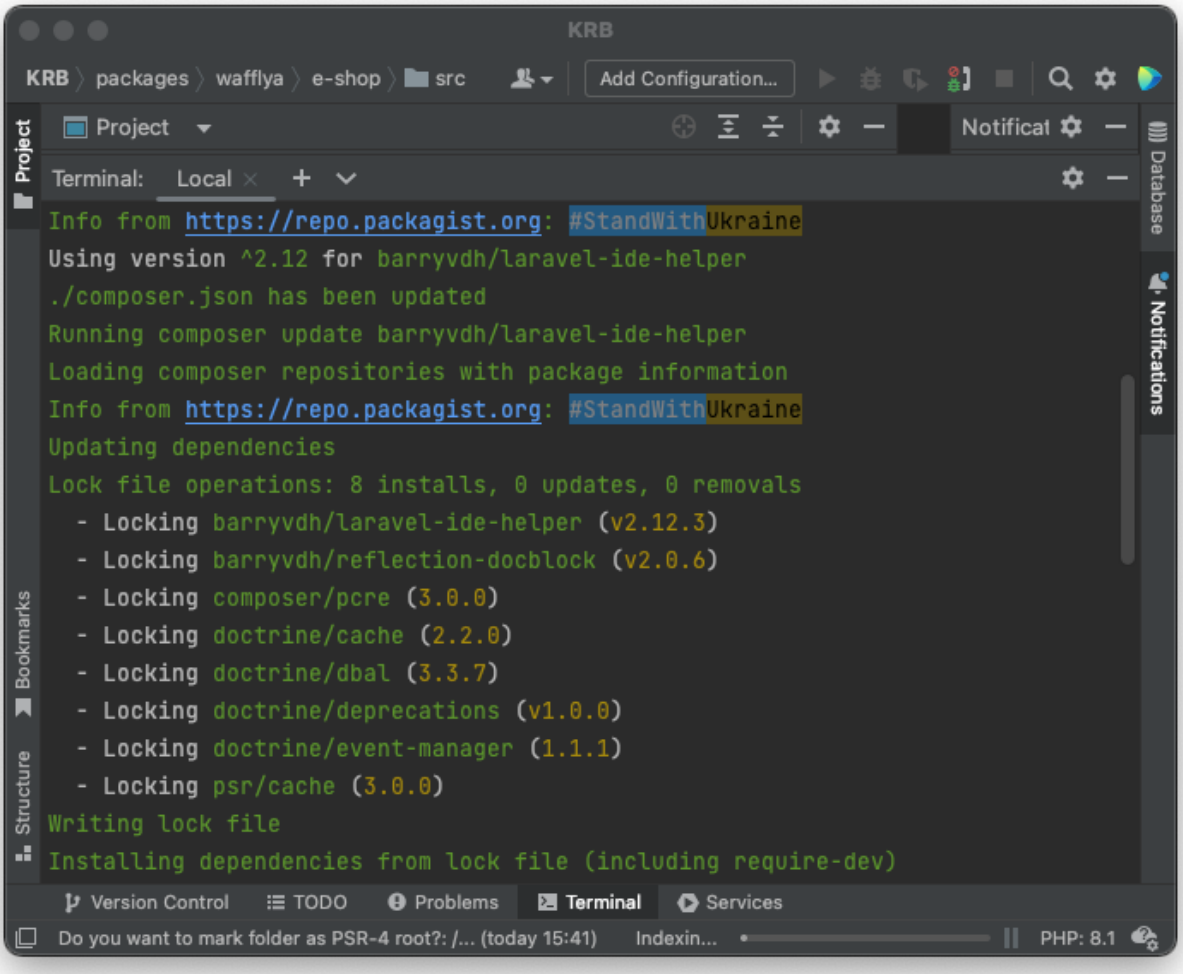

Рисунок 4.4 – Процес встановлення пакету

Для того щоб опублікувати всі файли до існуючого проєкту Laravel, треба використати команду *php artisan vendor:publish* з ключами, які потрібні для публікації необхідних файлів. Після введення команди в консолі з'явиться повідомлення, що процес публікації файлів завершено.

Після треба визначити клас, який буде використовуватись для зберігання у кошику. На прикладі буде взятий клас **Product** (рис. 4.5), який наслідується від класу **BusketItem** та використовує trait **Busket**. Дані класи розширюють клас **Product** та допомагають системі зрозуміти з якими класами треба працювати.

Також до класу треба додати додаткові поля у вигляді масиву **$busketAttributes**. Кожен елемент масиву є строковим. У строковому елементі вказується назва поля та через вертикальну риску тип даних, які повинні зберігатись у базі.

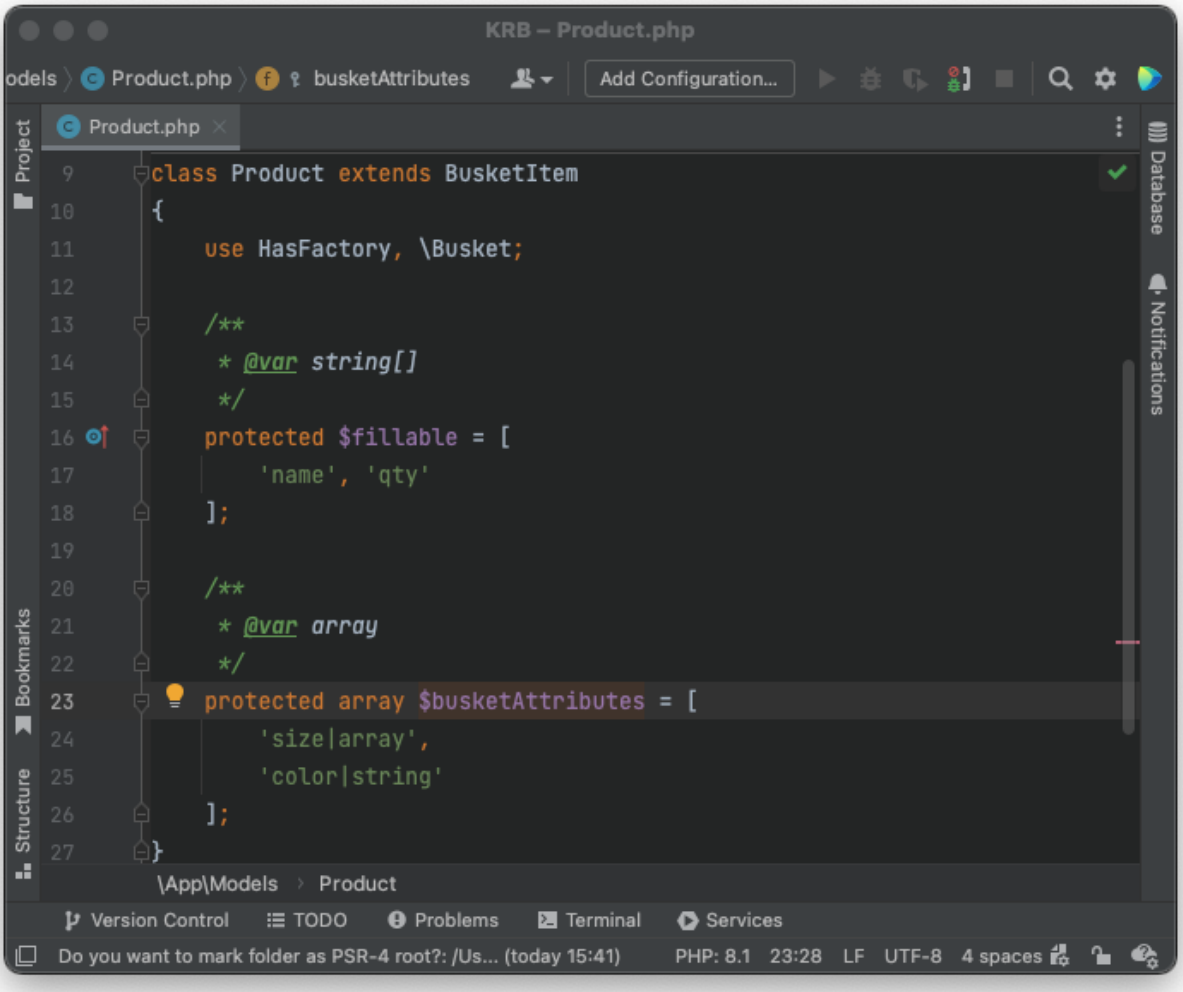

Рисунок 4.5 – Клас Product

Після публікації всіх необхідних конфігураційних файлів треба налагодити проєкт Laravel для роботи, а саме згідно офіційної документації Laravel[13] змінити драйвер сесії з файлового режиму у режим з використанням бази даних. Таким чином сесія буде використовуватись як дані у реляційній базі даних.

Наступним параметром, який треба змінити – ***middleware*** авторизації, який буде використовуватись як параметр за для отримання аутентифікованого

користувача. Якщо middlware відсутній, то системи буде автоматично використовувати базовий метод для отримання авторизованого користувача у системи з використанням його облікового запису. Якщо у проєкті присутні декілька проміжних програмних забезпечень(middleware), то у цьому випадку в якості параметру можна використати масив значень, у якому перелічені всі проміжні програми.

Для того щоб припинити логування системи, треба поставити ключ FALSE у відповідному полі **log** у конфігураційному файлі пакету.

4.3 Інструкція користувача з використання composer-пакету для розробки Інтернет-магазину у проєкті Laravel

Для того щоб використовувати composer-пакет для розробки Інтернет-магазину потрібно використовувати класи, які відповідають за логіку роботи системи, а саме: **Invoice**, **Order**, **Cart**. Кожен з цих класів має статичні методи, які допомагають легко і швидко робити обчислення та отримувати необхідні дані.

Клас ***Cart*** містить певний набір методів (рис. 4.6), які відповідають за кошик та всі операції з обраними товарами користувачем.

Першим статичним методом є метод **get()**. За допомогою даного методу можна отримати всі товари, які присутні у кошику. Якщо у якості параметру ввести значення числового типу, тоді метод поверне елемент кошику з ID, який дорівнює числовому параметру методу.

Наступним методом є метод додавання товару до кошику **add()**. В якості параметру приймається продукт та значення, які повинні використовуватись при додаванні товару до кошику. Якщо такий товар вже є, то метод аналізує чи є товар з таким ж самими характеристиками. Система аналізує чи є товар подібний тому що додається. Метод повертає продукт, який вдалося додати до бази даних.

Метод **update()** допомагає відновити дані певного продукту у кошику. Першим параметром методу є продукт, який треба відновити, а другим – поля з відновленими даними. Якщо об’єкт вдалося відновити, то метод повертає оновлений продукт. У іншому випадку метод поверне значення **false**.

Метод **remove()** використовується у разі, якщо потрібно видалити певний елемент з кошику. В якості параметру метод приймає об’єкт продукту, який треба видалити. Метод повертає видалений об’єкт класу.

Для порівняння двох об’єктів використовується метод **compare()**. В якості параметру приймається два об’єкти класу. Метод повертає бульове значення.

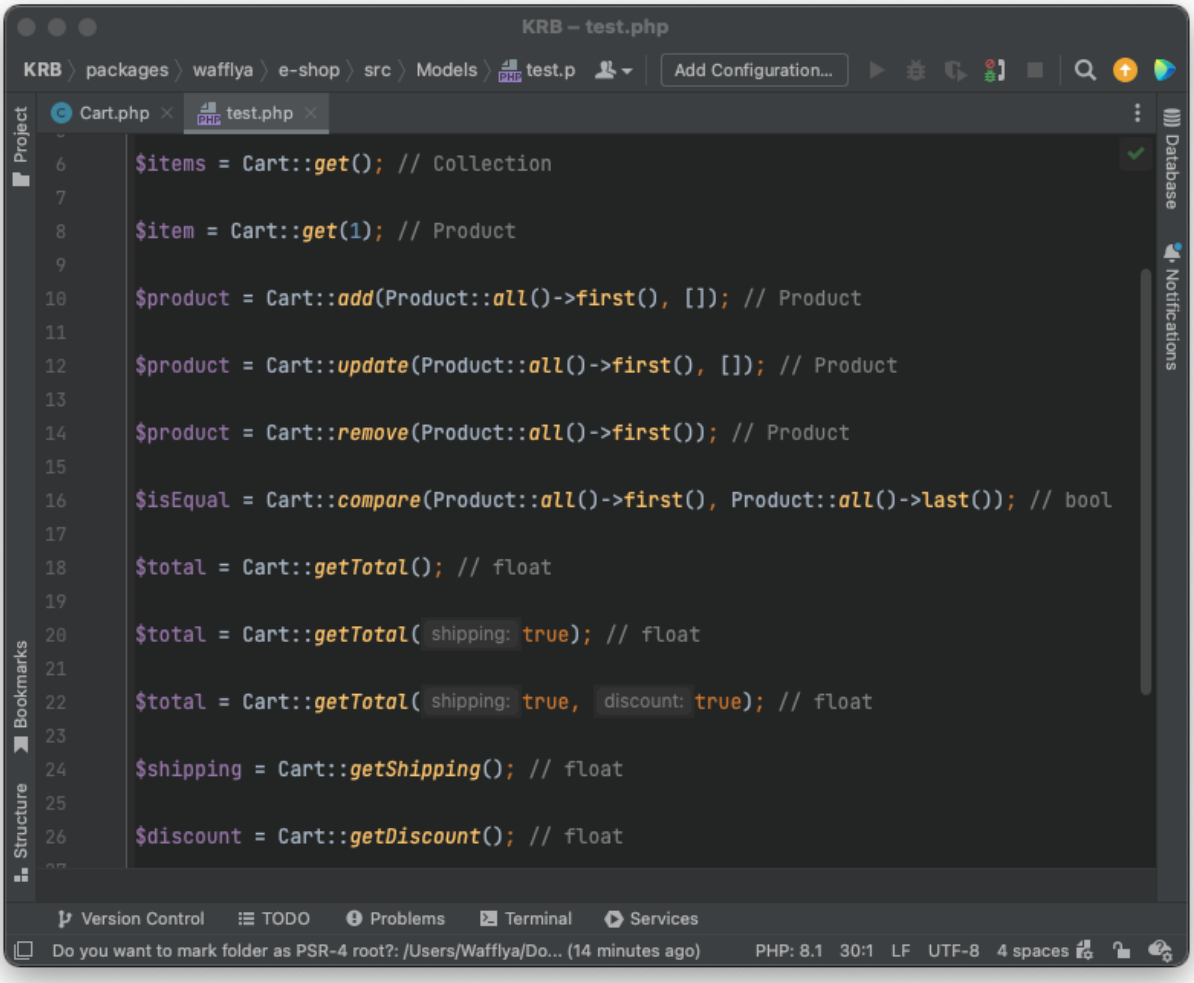

Рисунок 4.6 – Список статичних методів класу **Cart**

Метод **getTotal()** повертає суму цін усіх товарів, які присутні у кошику. Метод має два параметра, які допомагають обчислити ціну з доставкою та з

знижкою. Для того щоб окремо отримати знижку можна викликати метод **getShipping()**, а для отримання знижки – **getDiscount()**.

Клас ***Invoice*** регулює повідомлення користувачам на електронну скриньку. За допомогою статичних методів (рис. 4.7) можна легко маніпулювати з повідомленнями.

Метод **create()** використовується для створення повідомлення для користувача. В якості параметру метод приймає дані, які використовуються для створення повідомлення. Як саме виглядає повідомлення та саме які використовуються дані для відображення визначається у файлі ***invoice.blade.php***.

Метод **send()** приймає в якості параметру об'єкт повідомлення, відправника та отримувача. Метод використовує стандартні засоби Laravel для відправки повідомлення на електронну скриньку користувача. Якщо відправка була вдалою, то метод повертає **true**. У іншому випадку – **false**.

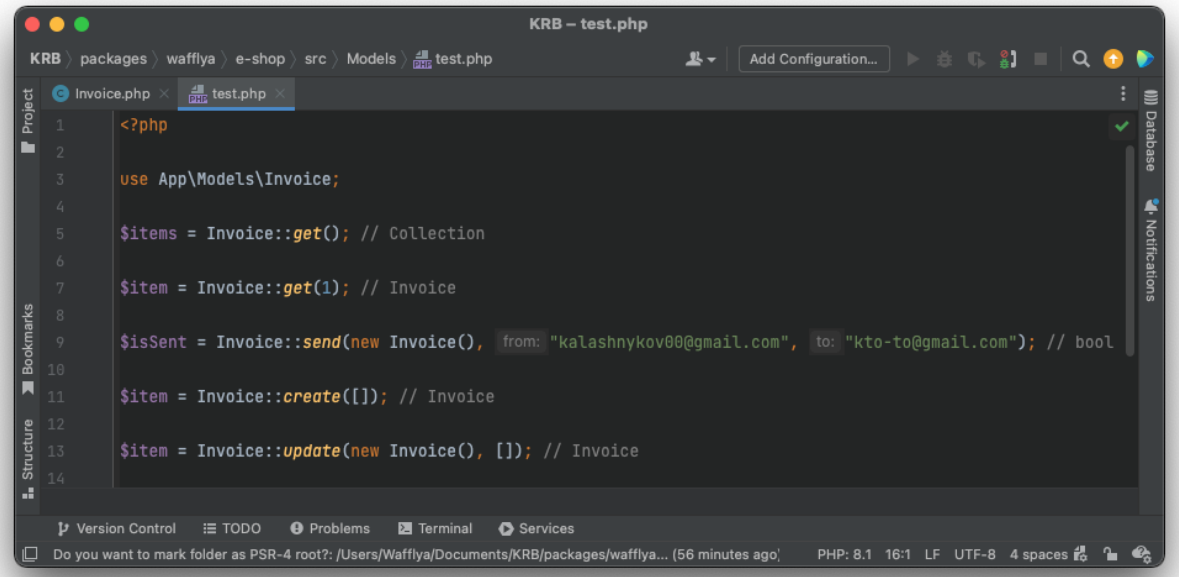

Рисунок 4.7 – Статичні методи класу **Invoice**

Метод **get()** еквівалентний статичному методу **get()** у **Cart**, але повертає об'єкти Invoice. Так само і методи **create()** та **update()**, але метод створення

повідомлення приймає лише параметри, які будуть використовуватись в представленні повідомлення поштової скриньки користувача.

Методи класу ***Order*** (рис. 4.8) схожі на методи класу ***Cart*** та мають схожу сігнатуру.

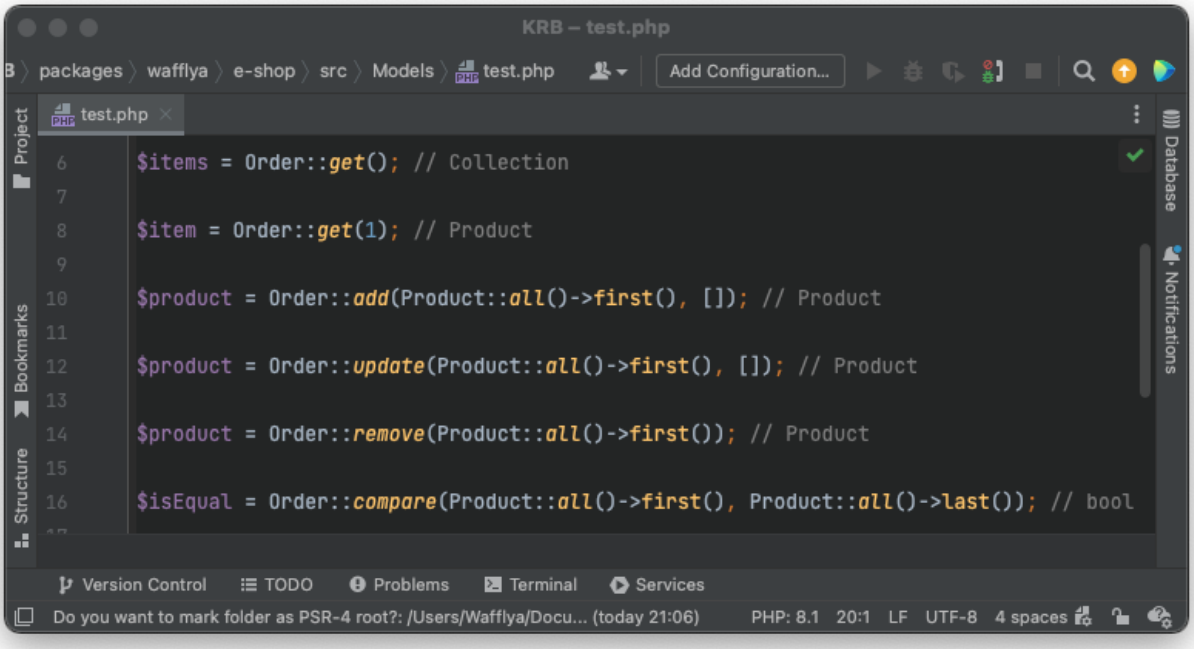

Рисунок 4.8 – Методи класу **Order**

Висновки до розділу 4

Зроблено детальний опис файлової структури composer-пакету для розробки Інтернет-магазину. Описано призначення кожної директорії та файлу у симбіозі з Laravel проєктом.

Зроблено докладну інструкцію з детальним поясненням кожного кроку інсталяції та налагодження composer-пакету у проєкті Laravel.

Створено детальну інструкцію використання composer-пакету для створення Інтернет-магазину.

ВИСНОВКИ

Результатом кваліфікаційної роботи є повноцінна система для розробки сайту на базі composer у середовищі Laravel, яка оптимізує та прискорює розробку Інтернет-магазину за рахунок розробки методів, які присутні у великій кількості сайтів.

Розробка кошика, система обліку кругообігу товарів та замовлень займає значну частину розробки інтернет-магазину та займає майже 30%[14](відсоток може відрізнятися від складності проєкту) від розробки повноцінного стандартного інтернет-магазину. Якнайменше для розробки корпоративного сайту потрібно 3 місяці часу програмістів середнього рівня досвідченності. Таким чином за допомогою пакету компанія може зменшити бюджет на кодування сайту майже на 30% за рахунок вже прописаного функціоналу, який присутній в багатьох сайтах типу ecommerce.

Для досягнення визначеної мети виконані певні задачі:

- проаналізувано вже існуючі рішення, а саме: сценарії роботи системи, версії php-фреймворку які підтримує рішення, засоби апаратної та програмної реалізації;
- спроєктовано систему у вигляді діаграми використання, діаграми класів, діаграми станів та діаграми діяльності;
- розроблено back-end частини з використанням php-фреймворку Laravel;
- завантажено composer-пакету на ресурси.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Laravel Usage Statistics. URL: https://trends.builtwith.com/framework/Laravel (дата звернення 20:42 25.04.2022).
2. Is Laravel Reliable for eCommerce Website Development? URL: https://www.avyatech.com/why-choose-laravel-for-ecommerce-development (дата звернення 20:53 25.04.22).
3. Международный сравнительный анализ роли малых и средних предприятий в национальной экономике. URL: https://www.researchgate.net/publication/334061178_Mezdunarodnyj_sravnitelnyj_analiz_roli_malyh_i_srednih_predpriatij_v_nacionalnoj_ekonomike_statisticeskoe_issledovanie (дата звернення 16:56 19.05.22).
4. Laravel vs Symfony vs Yii. URL: https://stackshare.io/stackups/laravel-vs-symfony-vs-yii#pros (дата звернення 16:49 18.05.22).
5. Bauer S. Laravel secrets: first edition. Germany : Self Published, 2020. 128 с.
6. Davis A. L. Design patterns. Modern programming made easy. Berkeley, CA, 2020. P. 59–69. URL: https://doi.org/10.1007/978-1-4842-5569-8_9 (дата звернення: 10.05.2022).
7. Duckett J. PHP and MySQL. Wiley & Sons, Incorporated, John, 2019. 672 с.
8. Laravel 5 Essential. Martin Bean, Published by Packt Publishing Ltd. 2015. 105 с.
9. MySQL. A hands-on introduction to data science. 2020. С. 187–206. URL: https://doi.org/10.1017/9781108560412.008 (дата звернення: 10.05.2022).
10. Usage statistics of PHP for websites. W3techs. URL: https://w3techs.com/technologies/details/pl-php (дата звернення: 17:07 14.06.2022).

11.3 Best Practices to Refine API Testing | Postman. URL: https://katalon.com/resources-center/blog/postman-alternatives-api-testing (дата звернення: 17:41 14.06.2022).
12. Composer packages in the Package Registry. URL: https://docs.gitlab.com/ee/user/packages/composer_repository/ (дата звернення: 18:59 20.06.2022).
13. HTTP Session. URL: https://laravel.com/docs/9.x/session#main-content (дата звернення: 20:07 21.06.2022).
14. Using Laravel for E-Commerce: Tutorial & Live Demo. URL: https://snipcart.com/blog/laravel-ecommerce-website-tutorial (дата звернення: 22:07 21.06.2022).
15. Composer. In: Magento 2 DIY. Khliupko, V. (2017). 267 с.
16. Web Application Development with Yii 2 and PHP. Mark Safronov, Jeffrey Winesett. 305 с.
17. Yii 2 For Beginners. Bill Keck. 2015. 290 с.
18. Mastering Yii. Charles R. Portwood II. 2016 Packt Publishing. 256 с.
19. Yii2 Application Development Cookbook. Andrew Bogdanov, Dmitry Eliseev. 2016 Packt Publishing. 187 с.
20. Symphony: A Framework for Accurate and Holistic WSN Simulation. Albert M. K. Cheng. 205 с.
21. SYMPHONY: A Parallel Framework. T. K. Ralphs, L. Ladanyi. 195 с.
22. Bringing climate change into ecosystem-based management of the sea: Data and methods for the Symphony framework. Iréne Wåhlström , Jonas Pålsson, Oscar Törnqvist, Per Jonsson, Matthias Gröger, Elin Almroth-Rosell1. 367 с.
23. PHP and MySQL Web Development. Luke Welling, Laura Thomson. 189 с.
24. Programming PHP. Rasmus Lerdorf, Kevin Tatroe, Bob Kaehms, Ric McGredy. 257 с.
25. Web Database Applications with PHP and MySQL: Building Effective Database. Hugh E. Williams, David Lane. 407 с.

26. Learning PHP, MySQL & JavaScript: With JQuery, CSS & HTML5. Robin Nixon. 456 с.
27. Beginning PHP and MySQL: From Novice to Professional 2010. W Jason Gilmore. 316 с.
28. Programming PHP: Creating Dynamic Web Pages. Kevin Tatroe, Peter MacIntyre. 353 с.
29. PHP and MySQL for Dynamic Web Sites, Fourth Edition: Visual QuickPro Guide. Larry Ullman. 575 с.
30. PHP 5 Power Programming. Gutmans, Andi Bakken, Stig Sather Rethans, Derick. 235 с.