Київський національний торговельно-економічний університет

Кафедра інформаційних технологій у міжнародній торгівлі

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ (РОБОТА)

на тему:

«Розробка web-додатку для електронної комерції»

(за матеріалами ПАТ "Українська фінансова група" *м. Київ)*

Студента 2м курсу, 7 групи, факультету міжнародної торгівлі та права, денної форми навчання спеціальності 122 «Комп'ютерні науки»	_____ *(підпис студента)*	Тарасенка Євгенія Вадимовича
Кандидат фізико-математичних наук, доцент	_____ *(підпис наукового керівника)*	Нагірна Алла Миколаївна
Гарант освітньої програми Доктор технічних наук, професор	_____ *(підпис гаранта освітньої програми)*	Краскевич Валерій Євгенійович

Київ 2018

Зміст

Аннотація

Тарасенко Є.В.

“Розробка web-додатку для електронної комерції”

Сьогодні у створенні веб-додатків задіяна велика кількість веб-майстрів і для покращення продуктивності їх роботи розроблену низьку інструментів які допомагають уникати типової роботи. Одним із таких інструментів є фреймворк для розробки. У роботі розглядається сучасний, популярний і потужний фреймворк Yii2.

Yii2 має модульну структуру, надає розробнику можливість генерувати типовий код й побудований із застосуванням багатьох перевірених сучасних шаблонів проектування тому багато веб-майстрів використовують його для створення своїх веб-додатків.

Ключові слова: YII2, PHP FRAMEWORK, WEB DEVELOPMENT, REST API, YII2 GII.

					КНТЕУ-122-2018	
Зм.	Аркуш	№ документу	Підпис	Дата		
Зав. кафедрою		Краскевич В.Є.			Використання фреймворку побудови веб-додатку	Сторінка 3 / Сторінок 102
Керівник		Нагірна А.М.				
Гарант		Краскевич В.Є.				Факультет обліку, аудиту та інформаційних систем, 7м група
Розробив		Тарасенко Є.В.			Розробка web-додатку для електронної комерції	
Перевірив		Нагірна А.М.				

Annotation

Tarasenko E.V.

"E-commerce web application development"

Today, a large number of webmasters are involved in the creation of web applications and developed low-level tools to improve their productivity, helping to avoid typical work. One of these tools is the development framework. The modern, popular and powerful frameworks Yii2 are considered in the work.

Yii2 has a modular structure, giving the developer the ability to generate a typical code and built with the use of many proven modern design templates so many webmasters use it to create their own web applications.

Keywords: YII2, PHP FRAMEWORK, WEB DEVELOPMENT, REST API, YII2 GII.

Вступ

Актуальність теми полягає в тому, що Yii2 - це універсальний фреймворк і може бути задіяний у всіх типах веб-додатків. Завдяки його компонентної структурі і відмінній підтримці кешування, фреймворк особливо підходить для розробки таких великих проектів, як портали, форуми, CMS, магазини або RESTful-додатки.

Метою дослідження є аналіз фреймворку Yii2 і написання додатку із його використанням.

Перед автором ставилось наступне **завдання**:

- Оцінка структури фреймворка.
- Порівняння із конкурентами.
- Аналіз функціоналу фреймворку.
- Налаштування веб-серверу для розташування веб додатку.
- Аналіз існуючих баз данних і підбір тієї, що задовольнить потреби додатку.
- Створення додатку, з обов'язковим доступом у WEB
- Тестування додатку.
- Зробити додаток доступним у WEB.

Yii2 - це високопродуктивний компонентний PHP фреймворк, призначений для швидкої розробки сучасних веб-додатків. Слово Yii (вимовляється як Йі [ji:]) в китайській мові означає «простий і еволюціонуючий».

Об'єктом дослідження є розробка веб додатку для електронної комерції.

Предметом дослідження є фреймворк Yii2.

Перше на що звертають увагу в Yii.

- Як і у багатьох інших PHP фреймворків, для організації коду Yii використовує архітектурний патерн MVC (Model-View-Controller).
- Yii дотримується філософії простого і елегантного коду, не намагаючись ускладнювати дизайн шаблонами проектування без яких можна обійтися.
- Yii є full-stack фреймворком і включає в себе перевірені і добре зарекомендовні можливості, такі як ActiveRecord для реляційних і NoSQL баз даних, підтримку REST API, багаторівневе кешування і інші.
- Yii відмінно розширюваний. Фреймворк надає можливість налаштувати або замінити практично будь-яку частину основного коду. Використовуючи архітектуру розширень, легко ділитися кодом або використовувати код спільноти.
- Одна з головних цілей Yii - продуктивність.

Yii - не проект однієї людини. Він підтримується і розвивається сильною командою і великим співтовариством розробників, які їй допомагають. Автори фреймворка стежать за тенденціями веб-розробки і розвитком інших проектів. Найбільш підходящі можливості і кращі практики регулярно впроваджуються в фреймворк у вигляді простих і елегантних інтерфейсів.

На даний момент існує дві основні гілки Yii: 1.1 і 2.0. Гілка 1.1 є попереднім поколінням і знаходиться в стані підтримки. Версія 2.0 - це повністю переписаний Yii, що використовує останні технології і протоколи, такі як Composer, PSR, простору імен, трейти і багато іншого. 2.0 - поточне покоління фреймворка. На цій версії будуть зосереджені основні зусилля кілька наступних років. Далі у роботі буде розлядатися тільки друге покоління фреймворку.

Yii 2.0 вимагає PHP 5.4.0 і вище і найкращим чином працює на актуальній версії PHP 7. Щоб дізнатися вимоги для окремих можливостей ви можете

запустити скрипт перевірки вимог, який поставляється з кожним релізом фреймворка.

Для розробки на Yii буде потрібно загальне розуміння ООП, так як фреймворк повністю слідує цій парадигмі. Також варто вивчити такі сучасні можливості PHP як простір імен і трейти.

Розділ 1. Використання фреймворку Yii2 для побудови веб-додатку

1.1 Поняття електронної комерції

Електронна комерція - це сфера економіки, яка включає в себе всі фінансові і торговельні транзакції, що здійснюються за допомогою комп'ютерних мереж, і бізнес-процеси, пов'язані з проведенням таких транзакцій.

До електронної комерції відносять:

- електронний обмін інформацією (Electronic Data Interchange, EDI),
- електронне рух капіталу (Electronic Funds Transfer, EFT),
- електронну торгівлю (англ. e-trade),
- електронні гроші (e-cash),
- електронний маркетинг (e-marketing),
- електронний банкінг (e-banking),
- електронні страхові послуги (e-insurance).

Існує кілька загальновизнаних категорій, на які поділяється електронна комерція. Як правило, таке розмежування проводиться по цільовій групі споживачів.

Схема B2B або бізнес-бізнес.

Принцип здійснення подібної взаємодії дуже простий: підприємство торгує з іншим підприємством. Інтернет-платформи дають можливість значно спростити проведення операцій на всіх етапах, зробити торгівлю більш оперативної та прозорою. Часто, в таких випадках представник сторони замовника має можливість інтерактивного контролю процесу виконання замовлення шляхом роботи з базами даних продавця.

Інформація про товари може бути представлена як на сайтах, доступних для всіх користувачів в інтернеті, так і на веб-ресурсах, доступних тільки для авторизованих користувачів. Прикладом B2B угоди може бути продаж шаблонів для сайту компаніям для подальшого використання в якості основи дизайну власного веб-ресурсу компанії. Безумовно, сюди відносяться будь-які взаємодії, що включають в себе оптові поставки товару або аналогічне виконання замовлень. Прикладом такої взаємодії може бути оформлення замовлення онлайн дилером в особистому кабінеті, розміщеному на сайті дистриб'ютора [1].

Схема B2C або бізнес-споживач. У цьому випадку підприємство торгує вже безпосередньо з клієнтом (не юридичним, а фізичною особою). Як правило, тут мова йде про роздрібну реалізацію товарів. Клієнту такий спосіб здійснення комерційної операції дає можливість спростити та прискорити процедуру покупки. Йому не доводиться йти в магазин, щоб вибрати потрібний товар: досить переглянути характеристики на сайті постачальника, вибрати потрібну конфігурацію і замовити продукт з доставкою. Комерсанту ж можливості Інтернету дозволяють оперативніше відстежувати попит (крім економії на приміщенні і кадрах). Приклади цього виду торгівлі - традиційні інтернет-магазини, спрямовані на цільову групу безпосередніх споживачів товарів. З 2010 року розпочала розвиток так звана соціальна комерція, або сфера продажів товарів і послуг в соціальних мережах.

Схема C2C або споживач-споживач. Такий спосіб здійснення електронної комерції передбачає укладення угод між двома споживачами, жоден з яких не є підприємцем в юридичному сенсі слова. Інтернет-майданчики для подібної торгівлі є чимось середнім між ринком і колонкою оголошень в газеті. Як правило, комерція за схемою C2C здійснюється на сайтах Інтернет-аукціонів, що набувають все більшої популярності в наш час, тому даний вид електронної комерції вважається одним з динамічно розвиваючихся останнім

часом. Для клієнтів таких систем основна зручність полягає в більш низькій ціні товару, у порівнянні з його вартістю в магазинах.

Крім описаних вище найбільш поширених схем електронної комерції, існує і кілька інших. Вони не настільки популярні, але, все ж, застосовуються в деяких специфічних випадках. Йдеться про взаємодію як підприємців, так і споживачів з державними структурами. Останнім часом багато операцій щодо справляння податків, заповнення анкет, форм для замовлення поставок, робота з митницею стали проводитися за допомогою Інтернет-технологій. Це дозволяє значно полегшити роботу державних службовців з одного боку і дати можливість платникам позбутися деякої частки бюрократії - з іншого [6].

Електронна комерція стала невід'ємною частиною сучасної економіки. Все більше споживачів купують товари за допомогою мережі Інтернет, а комерційні організації так чи інакше використовують можливості даної мережі при здійсненні підприємницької діяльності. Загальний світовий обсяг продажів в одному тільки споживчому сегменті електронної комерції перевищив позначку в 1 трлн дол. Ще в 2012 р і характеризується стійким зростанням. Ринок електронної комерції в Європі досяг 312 млрд є в 2012 р.

За прогнозами до 2017 року 10,3% ($ 370 млрд в грошовому вираженні) загальних витрат населення США на товари повсякденного попиту припадатиме на електронну торгівлю і більше 60% всіх продажів будуть як-небудь пов'язані з інтернетом.

Протягом останніх 5 років український ринок електронної комерції показував щорічне зростання на рівні 50%-60%[2] незалежно від перманентних економічних коливань. При цьому ринок володіє серйозним потенціалом. Наприклад, в Німеччині ємкість ринку електронної комерції становить близько $36 млрд. В Україні аналогічний показник становить не більше $400-$500 млн.

Основні фактори, які гальмують розвиток українського ринку електронної комерції:

- відсутність законодавчої бази, яка б регулювала процес купівлі/продажу онлайн, здійснення електронних платежів за оплачені товари/послуги та яка б встановлювала прозорі правила гри на ринку (як для продавців, так і для покупців);
- слабо розвинена національна система електронних платежів;
- низька ефективність більшості існуючих українських торговельних майданчиків;
- низький рівень проникнення інтернету в регіонах [5].

1.2 Структура фреймворку Yii2

Yii2 додатки організовані згідно шаблону проектування модель-уявлення-контролер (MVC). Моделі представляють собою дані, бізнес логіку і бізнес правила; уявлення відповідають за відображення інформації, в тому числі і на основі даних, отриманих з моделей; контролери приймають вхідні дані від користувача і перетворюють їх в зрозумілий для моделей формат і команди, а також відповідають за відображення потрібного уявлення.

Крім MVC, Yii2 додатки також мають наступні сутності:

Додатки. Це глобально доступні об'єкти, які здійснюють коректну роботу різних компонентів програми та їх координацію для обробки запиту.

Існує два види додатків: веб додатки та консольні додатки. Перший тип в основному займається обробкою веб запитів, в той час як останній - консольних команд.

Компоненти програми — це об'єкти, зареєстровані в додатку, що надають різні можливості для обробки поточного запиту.

Додатки є сервіс локаторами. Вони зберігають безліч так званих компонентів додатка, які надають різні засоби для обробки запитів. Наприклад, компонент urlManager відповідальний за маршрутизацію веб запитів до потрібного контролера; компонент db надає функціонал для роботи з базою даних; і т.д.

Кожен компонент додатка має свій унікальний ID, який дозволяє ідентифікувати його серед інших компонентів в одному і тому ж додатку. Отримати доступ до компоненту наступним чином:

\Yii::$app->componentID

Наприклад, для отримання з'єднання з БД використовують наступний код \Yii::$app->db, і \Yii::$app->cache для отримання доступу до основного компоненту кешу, зареєстрованому в додатку.

Компонент додатка буде створено при першому зверненні до нього через вищевказаний вираз. Будь-які подальші звернення повертатимуть екземпляр того самого компонента.

Компонентами програми можуть бути будь-які об'єкти. Вони реєструються за допомогою властивості yii\base\Application::components в конфігурації програми.

Модулі. Це самодостатні пакети, які включають в себе повністю всі необхдні для MVC файли. Додаток може бути організовано за допомогою декількох модулів.

Модуль базується в директорії, яка називається базовим шляхом модуля. Так само як і в директорії додатку, в цій директорії існують піддиректорії controllers, models, views та інші, в яких розміщуються контролери, моделі, подання та інші елементи. У наступному прикладі (Таблиця 1.2.1) показано приблизний вміст модуля.

Таблиця 1.2.1 Приблизний вміст модуля

module/	Корінь модуля
Module.php	Файл класу модуля
controllers/	Директорія із контроллерами
DefaultController.php	Контроллер за замовченням
models/	Директорія із моделями
views/	Директорія із відображеннями
layouts/	Відображення шаблонів
default/	Відображення контроллерів
index.php	Відображення базового контроллеру

Поведінки (behaviors) - це екземпляри класу yii\base\Behavior або класу, успадкованого від нього. Поведінки, також відомі як домішки, дозволяють

розширювати функціональність існуючих компонентів без необхідності зміни дерева успадкування. Після прикріплення поведінки до компоненту, його методи і властивості "впроваджуються" в компонент, і стають доступними так само, як якщо б вони були оголошені в самому класі компонента. Крім того, поведінка може реагувати на події, що створюються компонентом, що дозволяє точно керувати або модифікувати звичайне виконання коду компонента.

Фільтри - це об'єкти, які можуть запускатися як перед так і після дій контролера. Наприклад, фільтр управління доступом може запускатися перед діями щоб упевнитися, що користувачеві дозволено доступ до ресурсу; фільтр стиснення вмісту може запускатися після дій для стиснення вмісту відповіді перед відправкою його кінцевому користувачеві.

Фільтр може складатися з пре-фільтра (фільтруюча логіка застосовується перед діями) і / або пост-фільтра (логіка, застосовувана після дій).

Фільтри є особливим видом поведінок. Їх використання нічим не відрізняється від використання поведінок. Ви можете оголошувати фільтри в класі контролера шляхом перекриття методу behaviors (), наприклад:

```
public function behaviors()
{
    return [
        [
            'class' => 'yii\filters\HttpCache',
            'only' => ['index', 'view'],
            'lastModified' => function ($action, $params) {
                $q = new \yii\db\Query();
                return $q->from('user')->max('updated_at');
            },
        ],
    ];
}
```

Рисунок 1.2.1 Приклад поведінок

За замовчуванням фільтри, оголошені в класі контролера, будуть застосовуватися до всіх його дій. Тим не менш, ви можете явно вказати і конкретні дії задавши властивість only. В наведеному вище прикладі фільтр HttpCache застосовується тільки до дій index і view. Можливо налаштувати властивість except щоб вказати дії, до яких фільтр застосовуватися не повинен.

Крім контролерів, можна оголошувати фільтри в модулі або в додатку. У цьому випадку вони застосовуються до всіх дій контролерів, які перебувають в цьому модулі або додатку якщо не задані властивості only і except як було описано вище.

Коли кілька фільтрів вказуються для одного виду діяльності, вони застосовуються відповідно до таких правил:

Пре-фільтрація

- Застосовуються фільтри, оголошені в додатку в тому порядку, в якому вони перераховані в behaviors ().
- Застосовуються фільтри, оголошені в модулі в тому порядку, в якому вони перераховані в behaviors ().
- Застосовуються фільтри, оголошені в контролері в тому порядку, в якому вони перераховані в behaviors ().
- Якщо, будь-який з фільтрів скасовує виконання дії, фільтри що залишилися(як пре-фільтри, так і пост-фільтри) не будуть виконані.
- Виконується дія, якщо пре-фільтрацію пройдено.

Пост-фільтрація

- Застосовуються фільтри оголошені в контролері, в порядку зворотному, перерахованого в behaviors ().

- Застосовуються фільтри оголошені в модулі, в порядку зворотному, перерахованого в behaviors ().
- Застосовуються фільтри оголошені в додатку, в порядку зворотному, перерахованого в behaviors ().

Віджети є багаторазові будівельні блоки, які використовуються в уявленнях для створення складних і гнучко налаштовуваних елементів призначених для користувацького інтерфейсу в рамках об'єктно-орієнтованого підходу. Наприклад, віджет вибору дати (date picker) дозволяє генерувати інтерактивний інтерфейс для вибору дат, надаючи користувачам додатка зручний спосіб для введення даних такого типу. Все, що потрібно для підключення віджета - це додати наступний код в уявлення:

<?php use yii \ bootstrap \ DatePicker; ?>

<?= DatePicker::widget (['name' => 'date'])?>

Рисунок 1.2.2 Використання віджету

У комплект Yii2 входить велика кількість віджетів, наприклад: active form, menu, віджети jQuery UI, віджети Twitter Bootstrap.

Віджети що розробляються повинні бути самодостатніми. Це означає, що для їх використання повинно бути достатньо всього лише додати віджет в уявлення. Домогтися цього буває важко в тому випадку, коли для його функціонування потрібні зовнішні ресурси, такі як CSS, JavaScript, зображення і т.д. Проте Yii2 надає підтримку механізму для роботи з ресурсами asset bundles, який може бути успішно використаний для вирішення даної проблеми.

У разі, коли віджет не містить логіки, а містить тільки код, який відповідає за виведення розмітки, він мало відрізняється від уявлення. Насправді, єдина його відмінність полягає в тому, що віджет являє собою окремий і зручний для поширення клас, в той час як уявлення - це звичайний PHP скрипт, який підходить для використання тільки в конкретному додатку.

Ресурс в Yii2 це файл який може бути заданий в Web сторінці. Це може бути CSS файл, JavaScript файл, зображення або відео файл і т.д. Ресурси розташовуються в Web доступних директоріях і обслуговуються безпосередньо Web серверами.

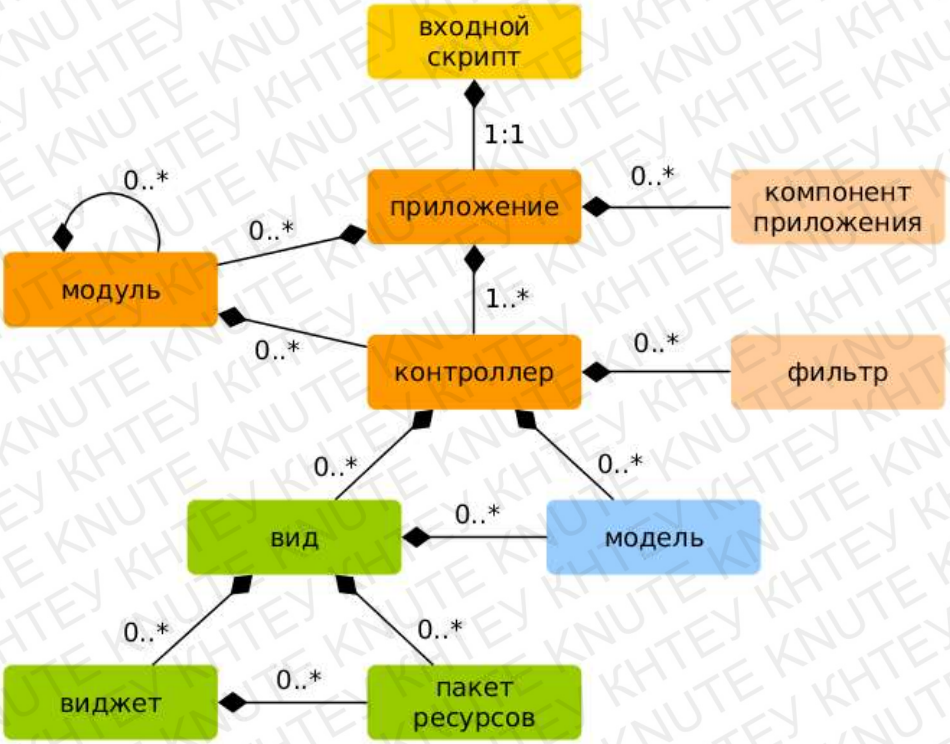

Рисунок 1.2.3 Структурна схема додатку.

При отриманні запиту на додаток, першою чергою виконується попереднє завантаження. Після нього виконується процессініціалізації самого додатку, його схему можна побачити на рисунку 1.2.3.

Попереднє завантаження це процес налаштування робочого середовища до того, як буде запущено програму і оброблений вхідний запит. Попереднє завантаження здійснюється в двох місцях: у вхідному скрипті і в додатку [5].

У вхідному скрипті, реєструються Автозавантажувач класів різних бібліотек. Цей процес включає в себе автозавантажувач класів Composer через autoload.php файл і автозавантажувач класів Yii2 через його Yii файл. Потім вхідний скрипт завантажує конфігурацію програми та створює об'єкт додатку.

У конструкторі додатка відбувається наступний процес попереднього завантаження:

1. Викликається метод preInit, що конфигурує властивості додатка, з найвищим пріоритетом, такі як basePath;
2. Реєструється обробник помилок;
3. Зчитуються дані властивостей додатка згідно заданої конфігурації;
4. Викликається метод init, який в свою чергу викликає метод bootstrap для запуску компонентів попереднього завантаження.
    a. Підключається файл маніфесту vendor/yiisoft/extensions.php;
    b. Створюються і запускаються компоненти попереднього завантаження оголошені в розширеннях;
    c. Створюються і запускаються компоненти програми та / або модулі, оголошені в властивостях попереднього завантаження додатку.

Оскільки попереднє завантаження здійснюється перш ніж буде оброблений кожен запит, то дуже важливо, щоб цей процес був легким і максимально оптимізованим.

Саме тому рекомендується не реєструвати занадто багато компонентів в модулі попереднього завантаження. Цей компонент потрібен тільки тоді, коли він повинен брати участь в повному життєвому циклі процесу обробки запиту. Наприклад, якщо модуль повинен зареєструвати додаткові правила парсингу URL, то він повинен бути зазначений у властивостях попереднього завантаження, щоб нові правила URL були враховані при обробці запиту.

Деякі великі програми можуть мати складну конфігурацію, яка розділена на кілька дрібних файлів. В цьому випадку варто кешувати весь конфігураційний файл і завантажувати його прямо з кешу до створення об'єкта додатка у вхідному скрипті.

Всі запити, оброблювані Yii2 додатком, проходять подібний шлях:

- Користувач створює запит до вхідного скрипту web / index.php.
- Вхідний скрипт завантажує конфігурацію і створює екземпляр додатку для обробки запиту.
- Додаток визначає запитаний маршрут за допомогою компонента request.
- Додаток створює екземпляр контролера для обробки запиту.
- Контролер створює екземпляр дії і виконує фільтри для цієї дії.
- При невдалому виконанні будь-якого фільтра, дія не виконується.
- При успішному виконанні всіх фільтрів, виконується дія.
- Дія завантажує модель даних, можливо, з бази даних.
- Дія рендерить уявлення і передає йому модель даних.
- Результат рендеринга передається в компонент додатка response.
- Компонент response посилає готові дані користувачеві.

Важливою частиною обробки вхідного запиту є роутинг. Він здійснюється в два етапи:

- Вхідний запит розбирається в маршрут і параметри запиту.
- Для обробки запиту створюється дія контролера, відповідно отриманому маршруту.

При використанні простого формату URL, отримання маршруту із запиту полягає в отриманні параметра r з масиву GET.

При використанні ЛЗУ, компонент URL manager шукає серед зареєстрованих правил що задовольняє запиту в маршрут. Якщо таке правило, не знайдено, викликається виняток yii \ web \ NotFoundHttpException.

Після того, як із запиту отриманий маршрут, створюється дія контролера, відповідна цьому маршрутові. Маршрут розділяється на кілька частин, мітками поділу служать прямі слеші. Наприклад, маршрут site / index буде розділений на site і index. Кожна з частин представляє собою ідентифікатор, який може

посилатися на модуль, контролер або дію. Починаючи з першої частини маршруту, додаток слідує наступним алгоритмом для створення модуля (якщо є), контролера і дії:

- Поточним модулем вважаємо додаток.
- Перевіряємо, чи містить карта контролерів поточного модуля поточний ідентифікатор. Якщо містить, відповідно до конфігурації контролера, знайденої в карті, створюємо об'єкт контролера і переходимо в п. 5 для обробки решти маршруту.
- Перевіряємо, чи є модуль, відповідний ідентифікатором в списку модулів (властивість modules) поточного модуля. Якщо є, відповідно до конфігурації модуля, знайденої в списку модулів, створюємо модуль і переходимо в п. 2, вважаючи щойно створений модуль поточним.
- Розглядаємо ідентифікатор як ідентифікатор контролера і створюємо об'єкт контролера. Для решти маршруту виконуємо п. 5.
- Контролер шукає поточний ідентифікатор в його карті дій. У разі знаходження, контролер створює дію, відповідно до конфігурації, знайденої в карті. Інакше, контролер намагається створити вбудовану дію, описану методом, що відповідає поточному ідентифікатору дії.

При виникненні помилок на будь-якому з описаних вище етапів, викликається виняток yii \ web \ NotFoundHttpException, який вказує на помилку в процесі роутінга.

До складу Yii2 входить вбудований обробник помилок, що реалізує функціонал який найчастіше потрібен розробникам. Всі нефатальні помилки PHP (тобто warning, notice) конвертуються в виключення, які можна перехоплювати. Винятки та фатальні помилки PHP відображаються в режимі налагодження з детальним стеком викликів і вихідним кодом. Також можна

використовувати для відображення помилок дію контролера або різні формати відповіді.

За замовчуванням обробник помилок включений, проте його можно вимкнути оголосивши константу YII_ENABLE_ERROR_HANDLER зі значенням false у вхідному скрипті вашого застосування.

Yii2 надає потужну, гнучко налаштовувану і легко розширювану систему логування. Вона дозволяє зручним способом зберігати повідомлення різних типів і фільтрувати їх. Повідомлення можуть бути збережені в файли, бази даних або відправлені на email.

Запис повідомлень логу здійснюється викликом відповідного методу із будь-якої частини додатку.

Yii::debug(): записує повідомлення для відстеження виконання коду програми. Використовується, в основному, при розробці.

Yii::info(): записує повідомлення, що містить будь-яку корисну інформацію.

Yii::warning(): записує тривожне повідомлення при виникненні несподіваного події.

Yii::error(): записує критичну помилку, на яку потрібно, як можна швидше, звернути увагу

Сесії і куки дозволяють зберігати призначені для користувача дані між запитами. При використанні чистого PHP можна отримати доступ до цих даних через глобальні змінні $_SESSION і $_COOKIE, відповідно. Yii2 инкапсулює сесії і куки в об'єкти, що дає можливість звертатися до них в об'єктно-орієнтованому стилі і дає додаткову зручність в роботі.

За аналогією з запитами і відповідями, до сесії можна отримати доступ через компонент додатка session, який за замовчуванням є екземпляром yii \ web \ Session.

Прикладом доступу до даних сесій є наступний код.

$session = Yii::$app->session;

```
$language = $session->get('language');
$language = $session->set('language', 'ua');
```

Рисунок 1.2.5 Доступ до даних сесії

За замовчуванням клас yii\web\Session зберігає дані сесії у вигляді файлів на сервері. Однак Yii2 надає ряд класів, які реалізують різні способи зберігання даних сесії:

- yii\web\DbSession: зберігає дані сесії в базі даних.
- yii\web\CacheSession: зберігання даних сесії в попередньо сконфігуровані компоненті кеша кеш.
- yii\redis\Session: зберігання даних сесії в redis.
- yii\mongodb\Session: зберігання сесії в MongoDB.

Всі ці класи підтримують однаковий набір методів API. В результаті є можливість перемикатися між різними сховищами сесій без модифікації коду програми.

Отже, Yii2 надає широкий функціонал базових можливостей дозволяючи розробнику сконцентруватися на створенні програми і не відволікатися на те, що вже було створено і протестовано широким ком'юніті.

Для управління залежностями фреймворк Yii2 використовує Composer.

Composer - це відносно новий і вже досить популярний менеджер залежностей для PHP. Він надає можливість описати від яких бібліотек залежить проект автоматично встановити потрібні бібліотеки. Особливістю Composer є те, що це не менеджер пакетів в класичному розумінні. Він оперує з сутностями, які розробник називає «пакетами» або бібліотеками, але встановлюються вони всередину кожного проекту окремо, а не глобально.

Composer працює наступним чином:

- Є проект, який залежить від декількох бібліотек.
- Деякі з цих бібліотек залежать від інших бібліотек.

- Розробник описує в своєму проекті ті бібліотеки, від яких безпосередньо залежить його код.
- Composer знаходить потрібні версії необхідних бібліотек для всього проекту, завантажує їх і встановлює в папку проекту.

За замовчуванням, пакети викачуються з офіційного репозиторію packagist.org. Будь-хто може вільно додати туди свій пакет, щоб зробити його установку максимально легкими і зручною для всього світу. Проте пакети можна завантажувати не тільки з packagist.org, але і з будь-якого git, mercurial або svn сховища.

При скачуванні пакетів з github.com або bitbucket.org не потрібно встановленої системи контролю версій, Composer працює через API цих сайтів. Git / hg / svn репозиторій з пакетом може перебувати не тільки на одному з перерахованих вище сайтів, але в будь-якому іншому місці, наприклад, в локальній мережі підприємства або взагалі на локальному жорсткому диску. Крім того, що встановлювана бібліотека не обов'язково повинна бути оформлена у вигляді Composer-пакета, Composer надає можливість зробити установку з будь-якого git / hg / svn сховища довільної структури. При необхідності встановлюваний пакет не обов'язково повинен бути git / hg / svn репозиторієм, він може бути довільним zip файлом доступний за веб-посилнням.

Всі пакети встановлюються в поточну директорію (звідки була виконана команда install), це дозволяє мати кілька різних версій бібліотек при роботі над різними проектами паралельно.

Команда update оновлює всі встановлені (або встановить заново випадково видалені) пакети до свіжих версій. А може і не оновлювати версії до найсвіжіших, якщо створити composer.lock файл - це дозволяє зафіксувати комбінацію з стабільних версій всіх використовуваних в проекті бібліотек.

Після установки пакетів автоматично генерується autoload.php, за допомогою якого можна підключити встановлені бібліотеки в коді вашого проекту. При підготовці Composer-пакета рекомендується використовувати PSR-0 - стандарт розташування і іменування php файлів, щоб autoload зміг їх легко знайти. У будь-якому випадку, автор пакета може описати правила, за якими autoload буде шукати файли тих чи інших класів або неймспейсів. Якщо розробник встановлює бібліотеку, яка оформлена як Composer-пакет (наприклад, довільний git репозиторій з github), то задача опису правил autoload лягає на нього. Таким чином ніякої магії з генеруються autoload.php немає - він вміє завантажувати все, головне, щоб були описані правила.

Розглянемо файл composer.json фреймворку, що наведено у додатку Д.

Цей файл є файлом типу json. JSON (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript) — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу. Розробив і популяризував формат Дуглас Крокфорд.

Першим ключем об’єкту composer.json йде name, що вказує на ім’я пакету, в даному випадку це yiisoft/yii2-app-basic – ім’я пакету базової версії фреймворку.

Далі йде коротке описання і ключові слова, за якими даний пакет може бути релевантним при пошуку.

Ключ homepage вказує на домашню сторінку пакету у WEB.

Ключ type. За замовчуванням це library. Типи пакетів використовуються для логіки індивідуальної установки. Якщо пакет потребує певної спеціальної логіки, може бути визначено конкретний тип. Це може бути symfony-bundle, WordPress-plugin або typo3-cms-ext. Ці типи будуть специфічними для певних проектів, і їм потрібно буде забезпечити програму установки, здатну встановлювати пакети такого типу.

Далі йде секція інформації про support. Спільнота фреймворку надає посилання на форум, де можна читати поради або задавати запитання; сайт де можна залишити баг-репорт; сайт-вікі, іншими словами інструкція та посилання за яким можна завантажити пакунок.

Секція minimum-stability визначає поведінку за замовчуванням для фільтрування пакетів за стабільністю. За замовчанням стабільно, тому що, якщо покладастися на пакет dev, можуть виникати помилки у ньому так як він може бути не дотестований або саме тестуватися.

Всі версії кожного пакунка перевіряються на стабільність, а ті, які менш стабільні, ніж налаштування мінімальної стабільності, будуть проігноровані при вирішенні залежностей за проектом. Доступні параметри (в порядку стабільності): dev, alpha, beta, RC, і stable.

Секція require вказує на список пакетів, що вимагаються цим пакунком. Пакет не буде встановлений, якщо ці вимоги не будуть виконані. Yii2 потребує PHP версії 5.4 або свіжішої, а також пакети yiisoft/yii2, yiisoft/yii2-bootstrap, yiisoft/yii2-swiftmailer.

Require-dev перераховує списки пакетів, необхідні для розробки, тесування тощо цього пакета. За замовчуванням встановлюються вимоги до кореневого пакета. Обидві команди install або update підтримують параметр --no-dev, який запобігає встановленню залежностей dev. Yii2 залежить від yiisoft/yii2-debug – модуль що збирає інформацію про помилки, швидкість виконання запитів у БД, тощо; yiisoft/yii2-gii – модуль для генерації коду, який буде використано для написання додатку у цій роботі; yiisoft/yii2-faker для створення фіктивних даних із ціллю функціонування; а також сторонні пакети codeception/base, codeception/verify та codeception/specify для написання автоматичних програмних тестів.

Секція config.process-timeout вказує на процес тривалості операції git clone, до того як Composer вирішить, що git не поверне записи. Тривалість операції перевиставляється на 1800 секунд [8].

Секція scripts містить команди які будуть виконані після операції composer install і composer create project. Yii2 встановлює свої хуки для виконання операцій налаштування після будь-яких змін у Composer.

У секції extra перераховуються додаткові дані, що будуть передані у скрипти із секції script.

Директива repositories вказує на репозиторії з яких будуть витягатися пакети.

Отже, Yii2 як і більшість сучасний фреймворків залежить від сторонніх пакетів, більша частина яких належить розробникам саме цього фреймворку. Проте для розробки і тестування Yii2 залучає великі, перевірені часом і користувачами пакети для автоматичного тестування коду, які мають свою, незалежну, документацію і використовуються не тільки в цьому проекті.

Yii2 не відноситься до мікрофреймворків, тому директорія vendor, у якій зберігаються залежності наповнена багатьма папками, у яких зберігаються і займають місце на дисту багаточислені файли розширень. Проте для великих додатків, як вказує практика, це не є мінусом, адже так, чи інакше розробникам все ж доведеться залучати готові рішення, або створювати свої для різних задач.

Фреймворк Yii2 надає багато корисних готових рішень одразу після встановлення, що є одним із його плюсів.

1.3 Порівняння із найпопулярнішими фреймворками

Одна з головних переваг, при використанні фреймворків, полягає в тому, що вони мають стандартну структуру. Фреймворки стали популярними з появою елементів інтерфейсу, які мали тенденцію до реалізації стандартної структури для додатків. З їх використанням стало набагато простіше створювати засоби для автоматичного створення графічних інтерфейсів, оскільки структура внутрішньої реалізації коду програми стала відома заздалегідь. Для забезпечення каркасу, зазвичай, використовують підходи об'єктно-орієнтованого програмування, наприклад, частини програми можуть успадковуватися від базових класів фреймворка.

Серед популярних фреймворків сьогодні виділяють:

Laravel – це комплексний фреймворк, призначений для швидкої розробки додатків за допомогою MVC-архітектури. На сьогоднішній день він є найбільш використовуваним з усіх PHP фреймворків і має величезне число шанувальників серед розробників.

У 2015 році Laravel посів перше місце серед PHP-фреймворків в номінціях фреймворк корпоративного рівня і фреймворк для особистих проектів. Отже, дана технологія результат вибору багатьох веб-фахівців.

Плюси: наявність MVC-архітектури (в тому числі для PHP 7); модульне тестування; високий рівень абстракції; спосіб уникнення перевантажень за допомогою динамічних методів; величезна кількість вбудованих функціональних можливостей; надійна система шифрування даних.

Інша основна перевага Laravel - потужний шаблонізатор Blade, що дозволяє вам використовувати простий PHP код в ваших виставах. Blade значно зменшує непотрібне навантаження в вашому додатку, оскільки всі blade-уявлення компілюються в простий PHP-код.

Artisan - це вбудований інструмент Laravel для управління командним рядком за допомогою потужного компонента Symfony Console. Він надає серію корисних команд для розробки програми.

Завдяки тому що Laravel надає високий рівень абстракцій, він підходить для створення веб додатків, що будуть довго підтримуватися і часто змінюватися. Проте така розробка буде займати більше часу на створення підтримуваної архітектури.

Серед слабких сторін необхідно відзначити, що багато розробників вважають цю платформу повільною, розробникам-новачкам буває складно виконувати її коди і класи, складні методи зворотного роутінга [9].

Yii2 – об'єктно-орієнтований компонентний PHP-фреймворк який реалізує парадигму MVC, один із найпопулярніших в СНД. Серед переваг: простота установки; Yii є повністю об'єктно-орієнтованим фреймворком і використовує всі переваги просунутих PHP функцій; Yii framework можна легко налаштувати під свої потреби. Практично кожен компонент фреймворка є розширюваним; Yii тісно інтегрований з Codeception; Yii поставляється з компонентом Security, який надає методи для створення більш безпечних додатків; фреймворк легко налаштувати для кращої продуктивності. Значним плюсом є генератор Gii, який є вбудованим модулем Yii. Можна сгенерувати модель, контролер або відразу модель, контролер і views для операцій CRUD на сутністю, таблицею. У Gii є шаблони генерації коду, які можна змінити, щоб файли генерувалися в тому вигляді, в якому ви хочете. Якщо проект довгий, то можна замислитися складанням різних шаблонів конкретно для цього проекту, що дуже спростить життя і прискорить процес розробки. Також є вбудована підтримка інтернаціоналізації.

Недоліки: сильна зв'язаність класів; доступ до моделей через статичні методи, що дозволяє використовувати їх навіть там, де не потрібно; немає вбудованої інтеграції шаблонізатора (Twig, Smarty).

Yii2 швидкий у налаштуванні і розгортанні. Ідеально підходить для проектів, які потрібно створити якнайшвидше [13].

Symfony – популярний фреймворк для розробки веб-сайтів і веб-додатків.

Робота Symfony2 заснована на використанні розділених (decoupled) компонентів багаторазового використання - Symfony Components. Всі вони вирішують поширені проблеми, з якими доводиться стикатися в процесі веб-розробки. До речі, деякі інші PHP-фреймворки використовують компоненти Symfony2, наприклад:

- Silex (компоненета BrowerKit, CssSelector, DomCrawler, EventDispatcher, HttpFoundation, HttpKernel, Routing, Form, Translation і Validator).
- Behat (компоненета Console, DependencyInjection, EventDispatcher, Finder, Yaml, Config і Translation).
- FLOW3 (компоненета YAML).

В основі популярної CMS Drupal8 також лежить використання компонентів Symfony2, таких як HttpKernel, HttpFoundation, EventDispatcher, YAML, Routing, Twig і багатьох інших.

Плюсами є висока продуктивність завдяки кешуванню байт-коду; надійність; наявність гарної документації; гарна підтримка, велика спільнота; повністю сформований фреймворк. Недоліком є те, що незважаючи на наявність гарної документації даний фреймворк досить складний в освоєнні.

Розробка на Symfony займає багато часу, проте модулі реалізуються із слабою залежністю одне від одного і можуть бути перевикористані, змінені або замінені без внесення безпосередніх змін в інші модулі. Simfony підходить для довготривалих і великих проектів [12].

Zend – це фреймворк, який представляє собою набір професійних PHP-розширень з більш ніж 158 мільйонами інсталяцій. Даний фреймворк використовується для розробки веб-додатків і сервісів за допомогою PHP 5.6+ і

гарантує 100-відсотковий об'єктно-орієнтований код, використовуючи широкий асортимент властивостей мови програмування. В основі архітектури покладено принцип MVC (розподіляє інтерфейс програми і його логіку), також багато бібліотек і компонентів, які дозволяють інтегруватися зі сторонніми проектами, наприклад, YouTube. Починаючи з версії 1.6 включає JavaScript-фреймворк Dojo. А в 2012 році вийшла друга повноцінна версія Zend. Поширюється на безкоштовній основі під ліцензією Open Source Initiativ. Фреймворк використовує Composer для впровадження залежностей пакетів; PHPUnit – для тестування всіх пакетів; Travis CI – в якості служби для безперервного інтеграційного тестування.

Серед сильних сторін слід відмітити: ідеальна заготівля для розробки комерційних додатків; об'єктно-орієнтований; безліч компонентів для валідації, «фідів» і форм; містить незв'язані компоненти; написаний повністю на PHP 5 і E_STRICT-сумісний; частини проекту мало залежать один від одного; підтримуються багато видів СУБД; підтримуються поштові протоколи mbox, Maildir, POP3 і IMAP4; наявність гнучкої системи кешування, яка підтримує різні типи (в пам'яті або в файлової системі). Проте незважаючи на плюси, через складність, Zend не настільки придатний для швидкої розробки веб-додатків, як інші фреймворки.

Найбільш часто використовують компоненти Zend: Zend_Controller, Zend_Layout, Zend_Config, Zend_Db, Zend_Db_Table, Zend_Registry, Zend View Helpers.

Zend framework слід використовувати з кількох причин. Перш за все, це максимальна відповідність стандартам, яке надає фреймворк і можливість багаторазово використовувати код. Адже часто виникають завдання, де необхідно пророблять однотипні процедури. При цьому немає необхідності підлаштовуватися під логіку нового завдання, можна використовувати вже існуючий код, який був написаний для попереднього типового завдання. Крім цього, фреймворк має відкритий вихідний код і велику спільноту розробників.

Група веб-фахівців веде роботу над Zend, виправляє неполадки і пропонує методи їх вирішення [14].

Phalcon – це MVC-орієнтований фреймворк для PHP. На відміну від інших фреймворків для роботи з Phalcon потрібно відносно невелика кількість ресурсів, що призводить до дуже швидкої обробки HTTP-запитів. Дана особливість може стати вирішальною для деяких розробників, що працюють з системами, про які складно сказати що-небудь заздалегідь.

Phalcon надає розробникам інструменти для зберігання даних, такі як власний діалект SQL: PHQL, а також об'єктно-документне відображення Object Document Mapping для MongoDB. Інші особливості даного фреймворка включають також: шаблонизатор, форм-білдери, простоту розробки додатків, що передбачає підтримку на різних мов і т.д. Phalcon є ідеальним варіантом як для створення різних REST API, так і для розробки повноцінних веб-додатків.

Переваги Phalcon - це, перш за все, висока продуктивність і невелике навантаження файлової системи; при суворій типізації спостерігається менша витрата пам'яті та часткова обробка інформації без інтерпретації. Також основними позитивними моментами є висока швидкість і малі перевантаження; автоматичне завантаження; унікальніст – даний фреймворк створювався як розширення мови програмування C; можна користуватися власною базою та її окремими елементами; накладні витрати в додатках мінімізуються за рахунок низкоуровневой організації; за технологіями ORM відбувається взаємодія з базами даних, що в результаті дає дуже велику продуктивність; всі процеси відбуваються досить швидко, завдяки прямому зверненню фреймворка до внутрішніх структур PHP; вбудовані засоби захисту; величезна кількість документацій; орієнтований на розробників.

Недоліком можна відзначини відсутність інтеграції з HHVM.

Оцінюючи вищезазначене бачимо, що Phalcon найкраще підходить для навантажених проектів [11].

Для порівняння популярності розглянутих PHP-фреймворків було обрано п’ятирічний інтервал і інструмент Google Trends.

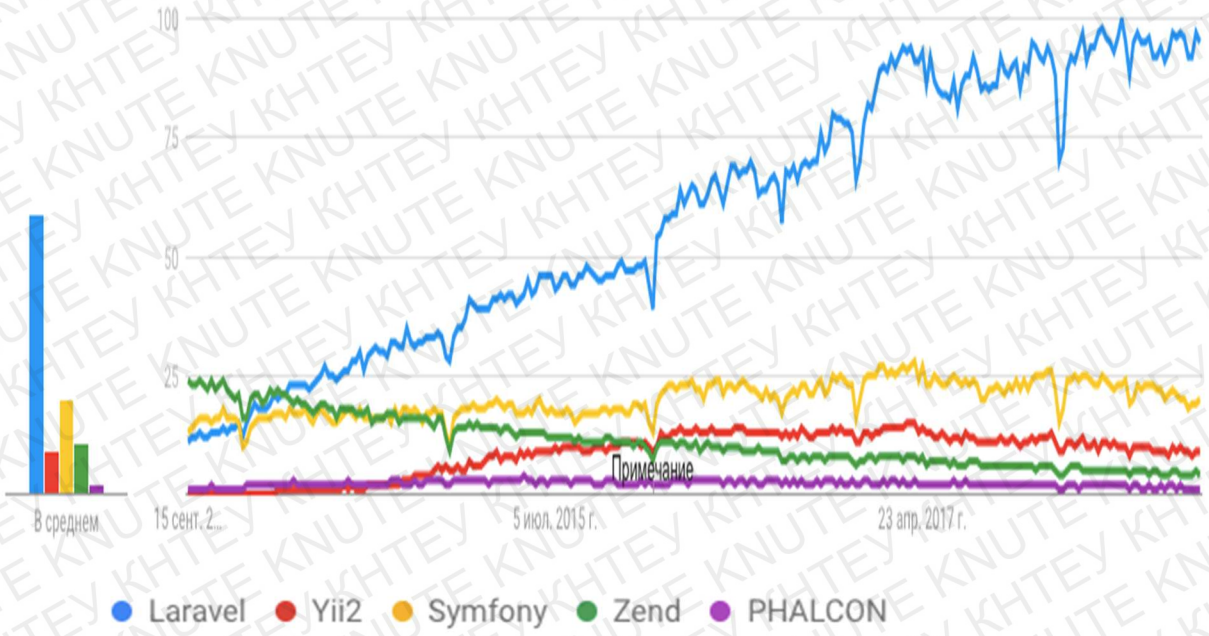

Рисунок 1.3.1 Динаміка популярності фреймворків за вересень 2013 – вересень 2018 років.

На графіку (рис. 1.3.1) ми бачимо що Laravel за останні 5 років укріплює свої позиції серед пошукових запросів у Google. У свою чергу кількість пошуків Zend стає все меншою, отже він втрачає популярність. Yii2 починає набирати популярність у 2014 році, і виходить на той рівень, який займає і досі — 15-20% від запитів по вибірці. Symfony тримає свою частку на рівні 20-25%. Phalkon займає досить малу долю в цій вибірці.

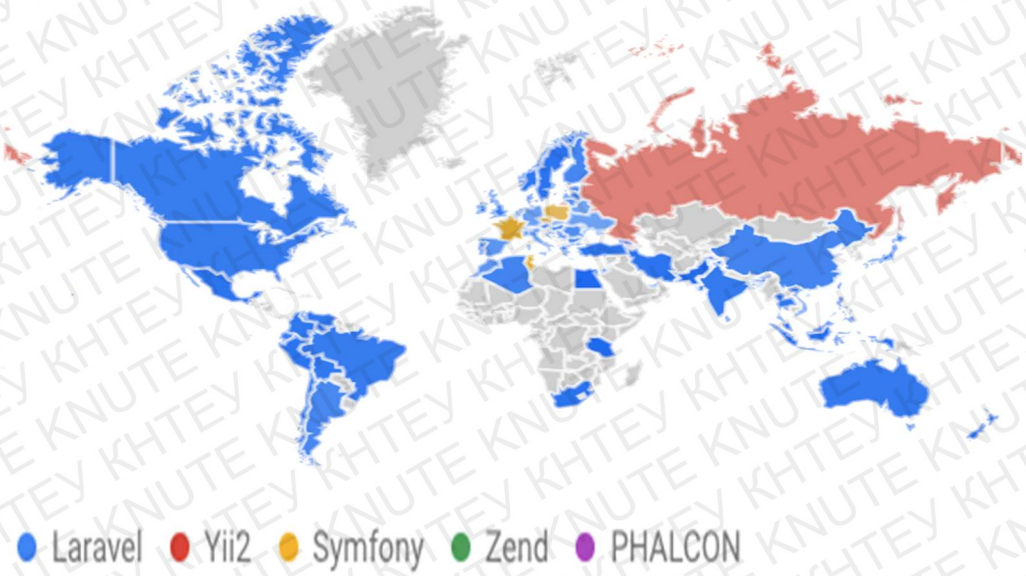

Рисунок 1.3.2 Порівняння по регіонам.

Оцінюючи регіональну зацікавленість (рис. 1.3.2) користувачів, можна побачити, що Laravel переважає майже всюду де є достатня вибірка. У Польші, Франції і Тунісі переважає Symfony, а в Росії — Yii2.

Рисунок 1.3.3 Динаміка в Україні.

На рисунку 1.3.3 бачимо, що в Україні ключові місця на ринку фреймворків займає Laravel та Yii2. Symfony також поширений, а Zend і Phalkon займають незначні позиції.

Бачимо що Laravel є найрозповсюдженішим фреймворком серед досліджуваних, проте його головним конкурентом є Yii2. Сучасні фреймворки

надають функціонал для створення додатків що можуть бути маштабованими коли навантаження на нього збільшуватимуться.

Фреймворк Symfony складається із незалежних компонентів, що надає можливість розробникам, включати їх у свої проекти і фреймворки. Відповідно при перенесенні коду із одного проекту до іншого, розробник має можливість перенести разом усі залежності його коду, що мінімізує ймовірність виникнення програмних помилок.

Розділ 2. Організація розробки на Yii2.

2.1 Аналіз існуючих баз даних

Дуже часто ми бачимо, що використовується основне операційне сховище і якісь додаткові сервіси. Наприклад, для кешування або повнотекстового пошуку.

Інший підхід до архітектури з використанням різних баз даних - це мікросервіси, у кожного з яких може бути своя база даних, яка краще оптимізована для задач саме цього сервісу. Як приклад: основне сховище може бути на MySQL, Redis і Memcache - для кешування, ElasticSearch або Sphinx - для пошуку. І щось на зразок Kafka - щоб передавати дані в систему аналітики, яка часто робилася на чомусь схожому на Hadoop.

Якщо ми говоримо про основне операційне сховище, напевно, у нас є два вибори. З одного боку, ми можемо вибрати реляційні бази даних, з мовою SQL. З іншого боку - щось нереляційне, а далі вже дивитися на підвиди, які доступи в даному випадку.

Якщо говорити про NoSQL-моделі даних, то їх теж досить багато. Найбільш типові - це або key value, або document, або wide column бази даних. Приклади: Memcache, MongoDB, Cassandra, відповідно.

Якщо подивитися на Ranking баз даних, то ми бачимо, що MySQL, згідно з цим рейтингом, - найбільш популярна реляційна база даних, а MongoDB - найбільш популярна нереляційних база даних.

Компанія MongoDB спочатку дуже активно фокусувалася на користувачах MySQL. Тому дуже часто у людей є досвід використання і вибір між цими двома технологіями [16].

Програмне забезпечення MySQL являє собою дуже швидкий багатопотоковий, розрахований на багато користувачів надійний SQL-сервер баз даних (SQL - мова структурованих запитів).

Сервер MySQL призначений як для критичних за завданнями виробничих систем з великим навантаженням, так і для вбудовування в програмне забезпечення масового поширення.

Програмне забезпечення MySQL має подвійне ліцензування. Це означає, що користувачі можуть вибирати, чи використовувати ПО MySQL безкоштовно по загальнодоступній ліцензії GNU General Public License (GPL) або придбати одну зі стандартних комерційних ліцензій MySQL AB.

MySQL - це перевірена технологія, яка використовується великими компаніями більше 15 років. Так як вона використовує стандарт SQL, є можливість досить простої міграції на інші SQL-бази даних. Є можливість транзакцій. Підтримуються складні запити, включаючи аналітику.

MySQL має декілька двигунів таблиць, вони будуть розглянуті у Додатку Е.

При роботі з MyISAM таблицями, необхідно пам’ятати, що вони складно відновлюються після неочікуваного перезавантаження сервера, тож він повинен бути захищений від цього. Утримуйтеся від змін таблиць декількома серверами або утилітою і сервером одночасно [25].

Двигун InnoDB був розроблений спеціально для великих таблиць. Розробники заявляють, що він найшвидший з усіх відомих двигунів для БД заснованих на дисках (множинні тести це підтверджують). Основною специфікацією його є високонавантажені сайти, фінансові транзакції, тощо.

MERGE зручний для реорганізації таблиць.

MEMORY доцільно використовувати для невеликої кількості даних, які повинні бути в оперативній пам’яті для швидкого доступу.

Двигун ARCHIVE відмінно підходить для рідкісних історичних даних. Таблиці не індексуються, і при вставці відбувається стиснення. Транзакції не

підтримуються. Двигун використовується для архівування та отримання архівованих даних.

BLACKHOLE використовують задля потреб реплікації. Майстер-сервер записує дані на диск, а слейв-сервер зчитує, таким чином досягається реплікація.

В MySQL - реляційна база даних. За допомогою реляційної бази даних легко відображати зв'язки між таблицями. Нормалізуючи дані, ми можемо змушувати зміни даних відбуватися атомарно в одному місці. Коли дані денормалізовані, при кожній зміні потрібно модифікувати всі документи.

Результатом будь-якого запиту завжди є таблиця. З одного боку, це просто, з іншого - деякі структури даних не завжди добре лягають на таблицю, що може виявитися незручним на практиці.

Якщо говорити про практичне використання MySQL, то треба розуміти що дані все ж можуть зберігатися денормалізованими. У процесі розробки виникає потреба зберігати JSON, XML або іншу структуру в колонках бази.

У MongoDB структура даних заснована на документах тому зручно зберігати багатовимірні дані веб-додатків. Якщо зберігаємо структуру - щось на зразок асоціативного масиву, то він легко серіалізується в JSON-документ. Розкладати такі дані в реляційну базу даних за різними табличками - завдання більш нетривіальне.

Результати як список документів, у яких може бути абсолютно різна структура - більш гнучке рішення.

Приклад. Ми хочемо зберегти контакт-лист з телефону. Зрозуміло, що є дані, які добре кладуться в одну реляционную табличку: прізвище, ім'я, тощо. Але якщо подивитися на телефони або email-адреси, то в однієї людини їх може бути кілька. Якщо подібне зберігати в правильному реляційному стилі, то дані мують зберігатися в окремих таблицях, потім, при запиті виконується JOIN, для отримання всіх результаті. Це менш зручно, ніж зберігати всі дані в одній колекції, де знаходяться ієрархічні документи.

В MongoDB немає такого поняття як JOIN, воно походить з реляційної структури. Там ми або робимо вбудований документ, що близько до концепту денормалізації, або ми просто зберігаємо ідентифікатор документа в якомусь полі, називаємо це посиланням і далі вибираємо дані, окремим запитом.

Що стосується доступу: там, де ми до реляційних даних використовуємо мову SQL, в MongoDB і багатьох інших NoSQL-базах даних використовується такий стандарт, як CRUD. Цей стандарт говорить, що є операції для створення, читання, видалення і оновлення документів.

SQL vs CRUD - Insert

• SQL

```sql
INSERT INTO book (
  `ISBN`, `title`, `author`
)
VALUES (
  '9780992461256',
  'Full Stack JavaScript',
  'Colin Ihrig & Adam Bretz'
);
```

• CRUD

```
db.book.insert({
  ISBN: "9780992461256",
  title: "Full Stack JavaScript",
  author: "Colin Ihrig & Adam Bretz"
});
```

Рисунок 2.1.1 Порівняння SQL і CRUD – вставка даних

Операція вставки даних відрізняється за синтаксисом, проте все ще лишається зрозумілою в обох випадках.

SQL vs CRUD Update

```sql
UPDATE book
SET price = 19.99
WHERE ISBN = '9780992461256'
```

```
db.book.update(
  { ISBN: '9780992461256' },
  { $set: { price: 19.99 } }
);
```

Рисунок 2.1.2 Порівняння SQL і CRUD – редакція даних

Для оновлення даних в CRUD першим масивом йдуть умови, а другим масив із командами. В даному випадку команда $set оновлює значення поля для всіх документів що задовольняють умови запиту.

SQL vs CRUD - Delete

```sql
DELETE FROM book
WHERE publisher_id = 'SP001';
```

```
db.book.remove({
  "publisher.name": "SitePoint"
});
```

Рисунок 2.1.3 Порівняння SQL і CRUD – видалення даних

Видалення виконується із одним масивом умов.

SQL vs CRUD - Count

```sql
SELECT COUNT(1) FROM book
WHERE publisher_id = 'SP001';
```

```
db.book.count({
  "publisher.name": "SitePoint"
});
```

Рисунок 2.1.4 Порівняння SQL і CRUD – підрахунок документів

Підрахунок, аналогічно із пошуком і видаленням, приймає першим аргументом масив типу “ключ-значення” для формування умов пошуку.

SQL vs CRUD - Aggregation

```sql
SELECT format, COUNT(1) AS `total`
FROM book
GROUP BY format;
```

```
db.book.aggregate([
  { $group:
    {
      _id: "$format",
      total: { $sum: 1 }
    }
  }
]);
```

Рисунок 2.1.5 Порівняння SQL і CRUD – групування даних

Для групування даних у MongoDB використовується агрегаційний фреймворк. З його допомогою можна виконувати як прості запити на групування так і більш складні запити на створення проміжних полів, перебір об’єктів нижчого рівня і створення додаткових умов [22].

Оцінивши можливості обох СУБД, необхідно обрати одну, яку буде використано для майбутнього додатку.

Досліджуваний додаток — веб сервіс, який буде агрегувати інформацію про доставку товарів. Цільова інформація – стан відправлення (відправлено/очікує відправлення/доставлено), вузол або склад на якому знаходиться , вузол на який їде, вузли на яких товар був.

Для додатку буде використано базу даних MySQL тому що немає необхідності зберігати дані у документо-орієнтованому форматі, дані повинні бути консистентними, нормалізованими і фреймворк Yii2 підтримує роботу із вищезазначеною СУБД за замовчуванням.

2.2 Налаштування веб-серверу для розміщення веб-додатку

В якості веб-хостингу буде використовуватися Mirohost. Першим ділом встановлюємо переадресацію нашого доменного імені на IP серверу Міpохосту який нам виділено. Доменне ім’я додатку - Lifttrackeret.com .

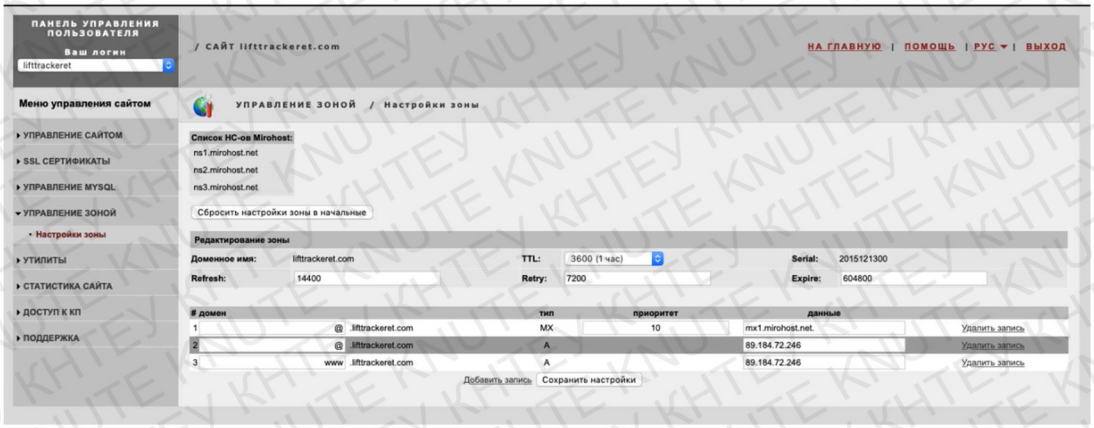

Рисунок 2.2.1 Зв’язка IP адреси серверу і доменного імені

У контрольній панелі знаходимо розділ де конфігуруються логіни і паролі для доступу на сервер за протоколами FTP і SSH. FTP доступ потрібен для завантаження на сервер коду додатку, медіафайлів й іншого статичного контенту. SSH доступ потрібен для доступу до командної стрічки серверу. Використовуючи її можливо встановлювати необхідні програми, встановлювати розширення до існуючих, конфігурувати, редагувати, видаляти їх й інше.

Підключення до сервера через SSH відбувається як на рисунку 1.

```
MacBook-Air-Yevhenii:~ yevheniitarasenko$ ssh lifttrackeret@es8.mirohost.net
lifttrackeret@es8.mirohost.net's password:
```

Рисунок 2.2.2 Підключення до сервера по SSH

Після підключення до сервера перевіряємо яка на ньому встановлена операційна система.

```
root@ae892ae639ff:/var/www/html# uname -a
Linux ae892ae639ff 4.9.49-moby #1 SMP Fri Dec 8 13:40:02 UTC 2017 x86_64 GNU/Linux
```

Рисунок 2.2.3 Перевірка операційної системи

Отже ми бачимо що сервер обслуговує операційна система Linux. Linux - це сімейство Unix-подібних операційних систем на базі ядра Linux, що включають той чи інший набір утиліт і програм проекту GNU, і, можливо, інші компоненти.

Традиційно системами Linux вважаються тільки ті, які включають в якості компонентів основні програми проекту GNU, такі як bash, gcc, glibc, coreutils, GNOME і ряд інших, в зв'язку з чим часто все сімейство іноді ідентифікується як GNU / Linux. Існує проект стандартизації внутрішньої структури Linux-систем - Linux Standard Base, частина з документів якого зареєстровано в якості стандартів ISO; але далеко не всі системи сертифікуються по ньому, і в цілому для Linux-систем не існує будь-якої загальновизнаної стандартної комплектації або формальних умов включення в сімейство. Однак є ряд систем на базі ядра Linux, які не мають в основі залежності від програм GNU, які до Linux-сімейства традиційно не відносять, зокрема такі це операційні системи Android і FirefoxOS [21].

Сервер має встановлені PHP 7.1 і MySQL

```
root@ae892ae639ff:/var/www/html# php -v
PHP 7.1.11 (cli) (built: Nov  4 2017 10:24:56) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2017 Zend Technologies
    with Xdebug v2.5.5, Copyright (c) 2002-2017, by Derick Rethans
```

Рисунок 2.2.4 Перевірка встановленості PHP

Як бачимо встановлена версія PHP 7.1.11. Це актуальна версія мови програмування, що має розширений функціонал і збільшену продуктивність у порівнянні із версією PHP 5.x.

```
root@ae892ae639ff:/var/www/html# mysql -V
mysql  Ver 14.14 Distrib 5.5.62, for debian-linux-gnu (x86_64) using readline 6.3
```

Рисунок 2.2.5 Перевірка встановленості MySQL

Сервер має встановлену актуальну версію MySQL.

Обслуговувати запити буде Nginx. Nginx веб-сервер і поштовий проксі-сервер, що працює на Unix-подібних операційних системах (збірка і дієздатність протестовано на FreeBSD, OpenBSD, Linux, Solaris, Mac OS X, AIX і HP-UX). Nginx позиціонується виробником як простий, швидкий і надійний сервер. Застосування nginx доцільно перш за все для статичних веб-сайтів і як проксі-сервера перед динамічними сайтами.

Можливості Nginx як HTTP-сервера:

- обслуговування незмінних запитів, індексних файлів, автоматичне створення списку файлів, кеш дескрипторів відкритих файлів
- акселерований проксінг без кешування, простий розподіл навантаження і відмовостійкість
- підтримка кешування при акселерованому проксінгу і FastCGI
- акселерована підтримка FastCGI і memcached серверів, простий розподіл навантаження і відмовостійкість
- модульність, фільтри, в тому числі стиснення (gzip), byte-ranges, chunked відповіді, HTTP-аутентифікація, SSI-фільтр
- підтримка SSL
- підтримка PSGI, WSGI

Згідно зі статистикою Netcraft nginx обслуговував або проксіровать 25.28% самих навантажених сайтів в жовтні 2018 року [21].

Налаштовуємо Nginx для сайту Lifttrackeret.com, використовуючи наступну конфігурацію.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name Lifttrackeret.com;
```

```
        index index.php index.html;
        error_log  /var/log/nginx/error.log;
        access_log /var/log/nginx/access.log;
        root /var/www/html/web;

        location / {
            index  index.html index.php;
            if (!-e $request_filename){
                rewrite ^/(.*) /index.php?r=$1 last;
            }
        }

        location ~ \.php$ {
            try_files $uri =404;
            fastcgi_split_path_info ^(.+\.php)(/.+)$;
            fastcgi_pass php:9000;
            fastcgi_index index.php;
            include fastcgi_params;
            fastcgi_param                                   SCRIPT_FILENAME
$document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $fastcgi_path_info;
        }
    }
```

Рисунок 2.2.6 Конфігурація сервера Nginx

Nginx слухає 80 порт, що є стандартним портом для очікування веб-запитів. Директива server_name вказує для якого доменного імені використовувати дану конфігурацію. Директива index вказує який файл у папці відкривати за замовчанням, якщо конкреттний файл не зазначено у запиті.

Директиви error_log і access_log вказують на файли на сервері в яких Nginx буде зберігати логи помилок і логи заборон доступу відповідно. Root вказує на базову папку в яку будуть направлені запити. Директива location ~ \.php$ із субдирективами встановлює правила, згідно якими будуть доступними до запиту тільки файли із розширенням php. Директива localhost / відповідає за побудову людинозрозумілих адрес у Yii2 [10].

Для встановлення фреймворку Yii2 на сервері потрібно встановити composer. Composer - це пакетний менеджер рівня додатків для мови програмування PHP, який надає функціонал з управління залежностями в PHP-додатку. Composer розробили і продовжують підтримувати два програміста Nils Adermann і Jordi Boggiano. Вони почали розробляти Composer в квітні 2011, а перший реліз відбувся 1 березня 2012 року.

Встановлюємо Composer.

```
root@ae892ae639ff:/var/www/html# curl -sS https://getcomposer.org/installer | php
All settings correct for using Composer
Downloading...

Composer (version 1.7.3) successfully installed to: /var/www/html/composer.phar
Use it: php composer.phar

root@ae892ae639ff:/var/www/html# mv composer.phar /usr/local/bin/composer
```

Рисунок 2.2.7 Встановлення composer

Встановивши Composer через нього можна встановити фреймворк Yii2 виконавши наступну команду

composer create-project --prefer-dist yiisoft/yii2-app-basic basic

Команда для встановлення фреймворку Yii2

```
root@ae892ae639ff:/var/www/html# composer create-project --prefer-dist yiisoft/yii2-app-basic basic
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Installing yiisoft/yii2-app-basic (2.0.14)
As there is no 'unzip' command installed zip files are being unpacked using the PHP zip extension.
This may cause invalid reports of corrupted archives. Besides, any UNIX permissions (e.g. executable)
defined in the archives will be lost.
Installing 'unzip' may remediate them.
  - Installing yiisoft/yii2-app-basic (2.0.14): Downloading (100%)
Created project in basic
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 62 installs, 0 updates, 0 removals
  - Installing yiisoft/yii2-composer (2.0.7): Downloading (100%)
  - Installing swiftmailer/swiftmailer (v5.4.12): Downloading (100%)
  - Installing bower-asset/jquery (3.2.1): Downloading (100%)
  - Installing bower-asset/yii2-pjax (2.0.7.1): Downloading (100%)
  - Installing bower-asset/punycode (v1.3.2): Downloading (100%)
  - Installing bower-asset/inputmask (3.3.11): Downloading (100%)
  - Installing cebe/markdown (1.1.2): Downloading (100%)
  - Installing ezyang/htmlpurifier (v4.10.0): Downloading (100%)
  - Installing yiisoft/yii2 (2.0.15.1): Downloading (100%)
  - Installing yiisoft/yii2-swiftmailer (2.0.7): Downloading (100%)
  - Installing bower-asset/bootstrap (v3.3.7): Downloading (100%)
  - Installing yiisoft/yii2-bootstrap (2.0.8): Downloading (100%)
  - Installing yiisoft/yii2-debug (2.0.14): Downloading (100%)
  - Installing bower-asset/typeahead.js (v0.11.1): Downloading (100%)
  - Installing phpspec/php-diff (v1.1.0): Downloading (100%)
  - Installing yiisoft/yii2-gii (2.0.7): Downloading (100%)
  - Installing fzaninotto/faker (v1.8.0): Downloading (100%)
  - Installing yiisoft/yii2-faker (2.0.4): Downloading (100%)
  - Installing sebastian/version (2.0.1): Downloading (100%)
```

```
  - Installing symfony/event-dispatcher (v4.1.7): Downloading (100%)
  - Installing symfony/console (v4.1.7): Downloading (100%)
  - Installing symfony/finder (v4.1.7): Downloading (100%)
  - Installing psr/http-message (1.0.1): Downloading (100%)
  - Installing guzzlehttp/psr7 (1.4.2): Downloading (100%)
  - Installing codeception/base (2.5.1): Downloading (100%)
  - Installing codeception/verify (0.3.3): Downloading (100%)
  - Installing codeception/specify (0.4.6): Downloading (100%)
sebastian/global-state suggests installing ext-uopz (*)
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
symfony/browser-kit suggests installing symfony/process
symfony/event-dispatcher suggests installing symfony/dependency-injection
symfony/event-dispatcher suggests installing symfony/http-kernel
symfony/console suggests installing psr/log-implementation (For using the console logger)
symfony/console suggests installing symfony/lock
symfony/console suggests installing symfony/process
codeception/base suggests installing aws/aws-sdk-php (For using AWS Auth in REST module and Queue modu
le)
codeception/base suggests installing codeception/phpbuiltinserver (Start and stop PHP built-in web ser
ver for your tests)
codeception/base suggests installing flow/jsonpath (For using JSONPath in REST module)
codeception/base suggests installing league/factory-muffin (For DataFactory module)
codeception/base suggests installing league/factory-muffin-faker (For Faker support in DataFactory mod
ule)
codeception/base suggests installing phpseclib/phpseclib (for SFTP option in FTP Module)
codeception/base suggests installing stecman/symfony-console-completion (For BASH autocompletion)
codeception/base suggests installing symfony/phpunit-bridge (For phpunit-bridge support)
Writing lock file
Generating autoload files
> yii\composer\Installer::postCreateProject
chmod('runtime', 0777)...done.
chmod('web/assets', 0777)...done.
chmod('yii', 0755)...done.
> yii\composer\Installer::postInstall
root@ae892ae639ff:/var/www/html#
```

Рисунок 2.2.8 Встановлення фреймворку Yii2

Встановлення фреймворку через Composer пройшло в 3 етапи. Першим етапом відбувається встановлення залежностей прописаних у файлі composer.json і формування файлу composer.lock. Після цього запустилися postCreateProject скрипти, які створили тимчасові папки і надалися права на необхідні директорії.

Тестуємо доступ до сайту перейшовши по посиланню і бачимо результат зображенний на рисунку .

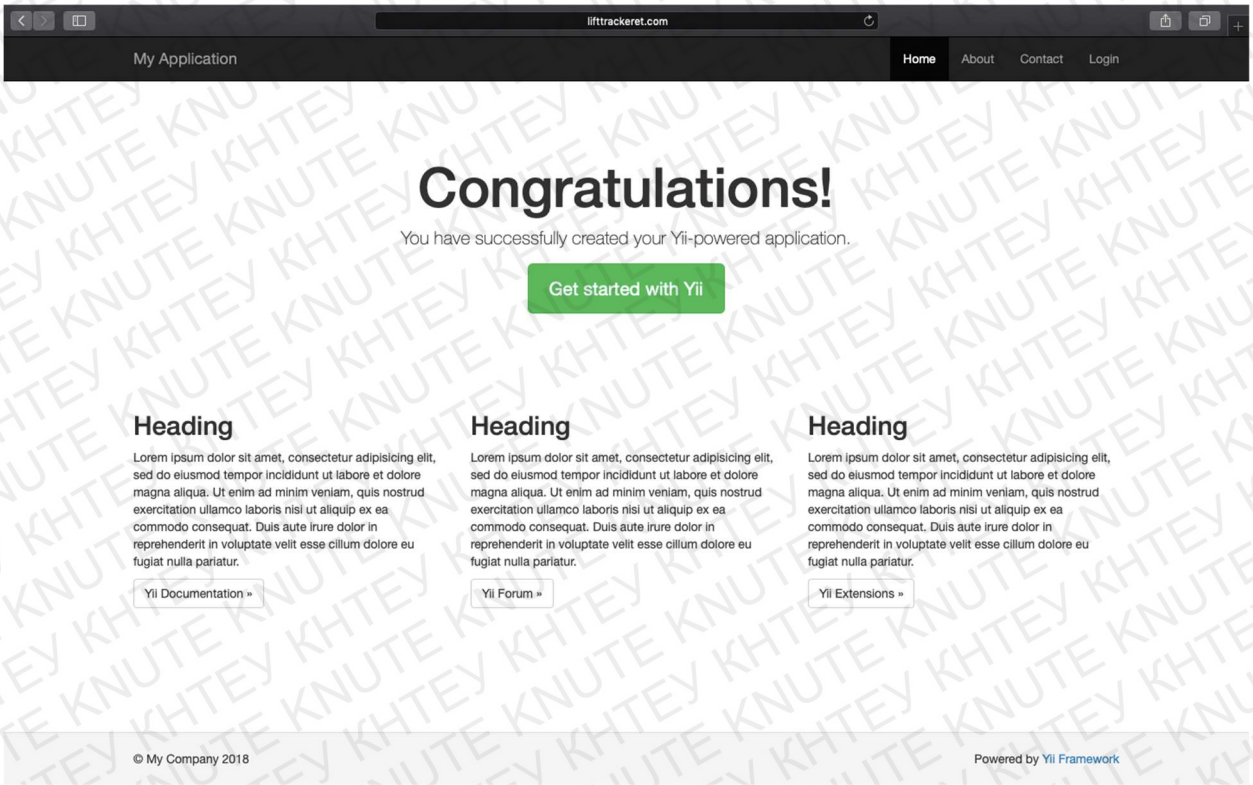

Рисунок 2.2.9 Доступ до сайту у WEB

Отже, бачимо що на даний момент сайт доступний з браузеру, а це означає що він у WEB. Далі нам необхідно підключити додаток до бази даних. Базова сторінка фреймворку Yii2 відображає статичну інформацію, а тому ми не бачимо звіту про помилку підключення до БД. Для того щоб протестувати з'єднання із базою даних ми можемо спробувати виконати міграції із інтерфесу комндної стрічки.

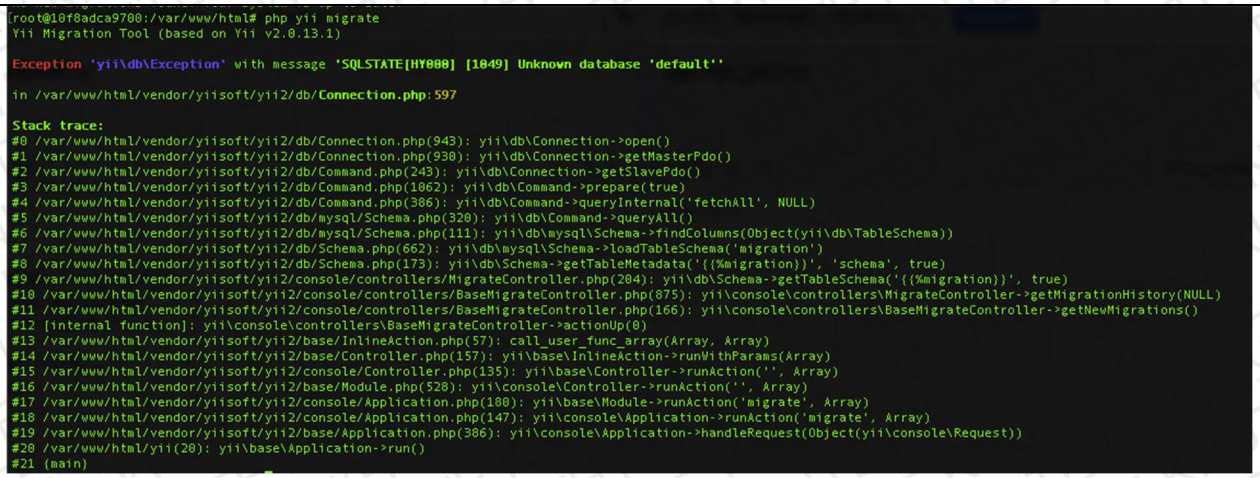

Рисунок 2.2.10 Виконуємо міграції із неналаштованим зв’язком із локальною БД

Для налаштування зв’язку необхідно відредагувати файл ./config/db.php і змінити зміст за замовчанням змістом із сніппету.

```
<?php
return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=Lifttrackeret;charset=utf8;port=3306',
    'username' => 'root',
    'password' => 'root',
    'charset' => 'utf8',
];
```

Рисунок 2.2.11 Конфігурація підключення до БД

Після налаштування підключення до БД намагаємося виконати міграції ще раз.

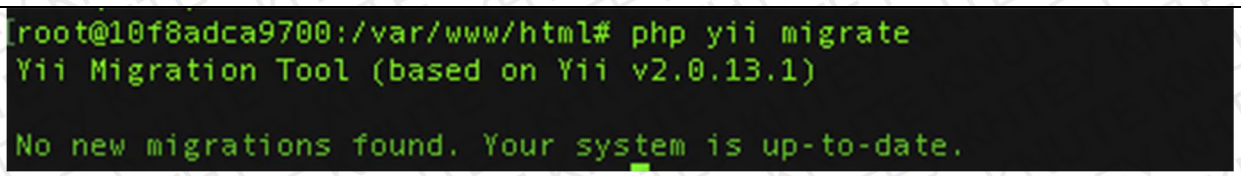

Рисунок 2.2.12 Успішний запуск міграцій

Як бачимо міграції виконалися і тепер додаток має успішне підключення до БД.

2.3 Проектування веб додатку

Спершу потрібно спроектувати структуру бази даних, щоб розуміти які дані нам необхідно отримувати і які ми зможемо надати клієнту.

Перша таблиця буде описувати покупця який замовив товар. Таблиця буде мати назву house і містити колонки id – первинний ключ, і адресу замовника (рисунок 2.3.1)

```
CREATE TABLE `HOUSE` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Рисунок 2.3.1 SQL запит створення таблиці house.

Наступна таблиця буде описувати конкретне замовлення, її назва – Order. Вона містить первинний ключ; колонку house, яка являється зовнішнім ключем і посилається на ідентифікатор таблиці house, аби зв’язати ці таблиці; node буде містити ідентифікатор вузлу на якому знаходиться товар; direction помічає чи перебуває товар у русі у наступний вузол; entrance – це статичне поле, що задається при створенні і вказує на те, з якого складу відправлено товар; поле status вказує на стан доставки; access_token – поле що містить токен за яким буде ідентифікуватися API; поле updated_at містить мітку часу UNIX, коли відбулося останнє оновлення даних.

Для поля access_token було створено унікальний індекс, що гарантує унікальність значення в стовпчику. Саме тому ми можемо бути впевнені у тому що не виникне колізія при роботі з API. Також індекси для колонок access_token і house значно прискорюють пошук по ним.

Запит на створення таблиці наведено на рисунку 2.3.2.

```
CREATE TABLE `Order` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
```

```
  `house` int(11) NOT NULL,
  `node` smallint(6) NOT NULL,
  `direction` smallint(6) NOT NULL,
  `entrance` smallint(6) NOT NULL DEFAULT '0',
  `status` smallint(6) NOT NULL DEFAULT '0',
  `access_token` varchar(255) NOT NULL,
  `updated_at` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `Order_token` (`access_token`),
  KEY `Order_house` (`house`),
  CONSTRAINT `Order_house` FOREIGN KEY (`house`) REFERENCES `house` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Рисунок 2.3.2 SQL запит створення таблиці посилок

Наступна таблиця Order_log буде зберігати історичні записи щодо значень із таблиці Order. Її структура ідентична із таблицею Order за виключенням полів що не потрібні в історичній перпективі.

```
CREATE TABLE `Order_log` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `house` int(11) NOT NULL,
  `node` smallint(6) NOT NULL,
  `direction` smallint(6) NOT NULL,
  `entrance` smallint(6) NOT NULL DEFAULT '0',
  `status` smallint(6) NOT NULL DEFAULT '0',
  `updated_at` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Рисунок 2.3.3 SQL запит на створення таблиці Order_log

Таблиця Order_stop буде зберігати дані щодо того де посилка буде зупинятися на шляху до кінцевого замовника.

Колонка Order_id – зовнішній ключ що пов’язує із таблицею Order. Node це вузол щодо якого зроблено запис. Created_at вказує на час створення запису і arrival_at буде заповнено коли посилка прибуде на даний вузол.

```
CREATE TABLE `Order_stops` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `Order_id` int(11) NOT NULL,
 `node` smallint(6) NOT NULL,
 `created_at` int(11) NOT NULL,
 `arrival_at` int(11) NOT NULL,
 PRIMARY KEY (`id`),
 KEY `Order_stops_arrivalAt` (`arrival_at`),
 KEY `OrderStops_Order` (`Order_id`),
 CONSTRAINT `OrderStops_Order` FOREIGN KEY (`Order_id`) REFERENCES `Order` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Рисунок 2.3.4 Запит SQL для створення таблиці Order_stops

Для організації зв’язку клієнта і сервера, буде використовуватися API з методологією REST.

Прикладни́й програмний інтерфейс (англ. Application Programming Interface, API) — набір визначень взаємодії різнотипного програмного забезпечення. API — це зазвичай метод абстракції між низькорівневим та високорівневим програмним забезпеченням.

В об'єктно-орієнтованих мовах, таких як PHP, прикладний програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з

реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу.

Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити Прикладний Програмний Інтерфейс (ППІ) як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, ППІ визначається набором повідомлень запиту HTTP, також визначається структура повідомлень-відповідей, зазвичай у розширенні мови розмітки XML або в форматі об'єктного запису JavaScript (JSON). У той час як прикладний програмний інтерфейс у Web історично був практично синонімом для веб-служби, останнім часом тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA) на більш прямі передачі репрезентативного стану (REST) стилів веб-ресурсів та ресурсів-орієнтованої архітектури (ROA). Частина цієї тенденції пов'язана з рухом Семантичного веб-ресурсу до Опису Платформ (RDF), Концепції розвитку веб-технологій інженерних онтологій. Прикладні програмні інтерфейси у Web, що дозволяють комбінувати декількома прикладними програмними інтерфейсами в нові додатки називають гібридними.

REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і,

зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути великою [26].

Перша архітектура від якої REST успадковує обмеження — це клієнт-серверна архітектура. Її обмеження вимагає розділення відповідальності між компонентами, які займаються зберіганням та оновленням даних (сервером), і тими компонентами, які займаються відображенням даних на інтерфейсі користувача та реагування на дії з цим інтерфейсом (клієнтом). Таке розділення дозволяє компонентам еволюціонувати незалежно.

Наступним обмеженням є те, що взаємодії між сервером та клієнтом не мають стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту.

Відсутність стану не означає що стану немає. Відсутність стану означає, що сервер не знає про стан клієнта. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може залишити сторінку відкритою протягом кількох років, перш ніж натиснути посилання, і тоді сервер відповість на інший запит. Тим часом сервер може відповідати на запити інших клієнтів, або нічого не робити — для клієнта це не має значення.

Таким чином, наприклад дані про стан сесії (користувача, який автентифікувався) зберігаються на клієнті, і передаються з кожним запитом. Це

покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси, задіяні для цієї операції, без жодного ризику втратити цінну інформацію. Також спрощується моніторинг і зневадження, бо для того аби розібратись, що відбувається в певному запиті, досить подивитись лише на той один запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші.

Мінусом цього обмеження є те, що знижується продуктивність через те, що в кожен запит тепер доводиться додавати дані сесії з клієнта. Також збереження стану на різних клієнтах важче підтримувати, бо реалізації клієнтів можуть відрізнятись, тоді як середовище сервера повністю під контролем розробника.

Додатковим обмеженням стилю REST є те, що системи, написані в цьому стилі, повинні підтримувати кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи зайвих запитів, але також зменшує надійність системи, через те, що дані в кеші можуть бути застарілими.

Користувацька частина додатку буде складатися з трьох частин:

- Закрите API.
- Відкрите API.
- Адміністративна частина.

Закрите API буде використовуватися безпосередньо вузлами для спілкування із сервером з приводу статусу посилки. Так як вузол не буде зчитувати ніякі дані, то будуть задіяні тільки ті методи, які передбачають роботу із даними.

Таблиця 2.3.1 Методи закритого API

Метод	Ендпоінт	Значення
POST	/Order/status	Встановлення статусу

POST	/Order/direction	Встановлення напрямку
POST	/Order/stop-node	Посилка повинна пройти через вузол
DELETE	/Order/stop-node	Посилка прибула на вузол

Відкрите API буде використовуватися для відображення існуючої інформації. Тому буде задіяно тільки метод GET. Цими користувачами можуть бути мобільні і веб додатки, що зацікавлені в інформації інформації про доставку. Серед них можуть бути:

- Покупці, що очікують товар.
- Працівники складів, доставки і менеджменту що планують логістику.
- Інші зацікавлені особи.

Таблиця 2.3.2 Методи відкритого API

Метод	**Ендпоінт**	**Значення**
GET	/Order	Отримання актуальної інформації
GET	/Order/log	Отримання історичної інформації
GET	/Order/stop-node	Отримання списку вузлів які чекають на прибуття посилки

Адміністративна частина буде використовуватися технічною підтримкою, особами, що наповнюють контентом бази даних покупців і, можливо, відділом аналітики.

Розділ 3. Розробка додатку

3.1 Розробка веб додатку API і адмін-панелі на Yii2

Розробку додатку необхідно починати із створення таблиць у базі даних. У зв'язку з тим, що зміна структури бази даних часто вимагає зміни вихідного коду, Yii2 підтримує так звану можливість міграції баз даних, яка дозволяє відстежувати зміни в базах даних за допомогою термінів міграції баз даних, які є системою контролю версій разом з вихідним кодом.

Yii2 надає набір інструментів для міграцій з командного рядка, які дозволяють:

- створювати нові міграції;
- застосовувати міграції;
- скасовувати міграції;
- застосовувати міграції повторно;
- показувати історію і статус міграцій;

Всі ці інструменти доступні через команду yii migrate.

Виконуємо коданду для створення нової міграції, через яку буде створено таблиці у БД.

```
root@10f8adca9700:/var/www/html# php yii migrate/create lifttrack_init
Yii Migration Tool (based on Yii v2.0.13.1)

Create new migration '/var/www/html/migrations/m181111_163025_lifttrack_init.php'? (yes|no) [no]:y
New migration created successfully.
```

3.1.1 Створення файлу міграції

У створеному файлі у файлі app/migrations/ m181111_163025_Ordertrack_init.php знаходиться згенований клас що має два публічних методи: safeUp та safeDown для виконання міграції і її відміни відповідно.

У методі safeUp (рисунок 3.1.2) записуємо комадни для створення таблиць, зовнішніх ключів і індексів.

```
$this->createTable('house', [
        'id' => $this->primaryKey(),
        'address' => $this->string(255)
    ], 'charset=UTF8');
```

Рисунок 3.1.2 Код створення таблиці house

Вищезазначений код буде опрацьований фреймворком і згенерує SQL для створення необхідної нам таблиці у базі даних (Рисунок 3.1.3).

```
$this->createTable('Order', [
        'id' => $this->primaryKey(),
        'house' => $this->integer()->notNull(),
        'node' => $this->smallInteger()->notNull(),
        'direction' => $this->smallInteger()->notNull(),
        'entrance' => $this->smallInteger()->defaultValue(0)->notNull(),
        'status' => $this->smallInteger()->defaultValue(0)->notNull(),
        'access_token' => $this->string(255)->notNull(),
        'updated_at' => $this->integer()->notNull()
    ], 'charset=UTF8');
```

Рисунок 3.1.3 Код створення таблиці Order

Як бачимо у створенні активну участь відіграє клас який унаслідувано класом створеної міграції Migration. Через змінну $this він надає можливість викликати методи створення таблиць createTable, який приймає три параметри – назва таблиці, массив із назвами колонок і їх описанням, характеристики таблиці. Метод primaryKey вказує на те, що колонка буде первісним ключем

таблці; метод integer вказує на те що колонка буде приймати цілочисленні значення; метод notNull вказує на те, що колонка не може бути незаповнена; метод smallInteger вказує на те, що колонка буде цілочисельна із довжиною запису – 6; метод defaultValue деклазує значення за замовчуванням для колонки; метод string вказує, що колонка буде містити строку. Останній параметр із характеристиками таблиці вказує на те, що все характеристики будуть прийняті за замовчанням, окрім того який було перeоб'явлено – кодування строкових стовпців буде UTF8.

Інші таблиці будуть створені ідентичним чином.

Після того як створені таблиці, необхідно додати індекси на стовпці по яким буде виконуватися пошук. Індекс - це відсортований набір значень. В MySQL індекси завжди будуються для якоїсь конкретної колонки або колонок.

```
$this->createIndex('Order_token', 'Order', ['access_token'], true);
```

Рисунок 3.1.4 Код створення індексу до таблиці Order_token

Метод створення індексу приймає 4 параметри. Перший це назва індексу; далі назва таблиці для якої буде створено індекс; після цього йде масив із колонками, по яким буде створено індекс; останнім параметром є флаг, чи є індекс унікальним, чи ні. В нашому випадку створюється унікальний індекс на колонку access_token.

Далі створюються зовнішні ключі.

```
$this->addForeignKey('Order_house', 'Order', 'house', 'house', 'id');
```

Рисунок 3.1.5 Код створення зовнішнього ключа

Метод створення зовнішнього ключа приймає п'ять обов'язкових параметрів – назва, цільова таблиця, колонка цільової таблиці, таблиця на яку

йде посилання, колонка цієї таблиці. Ми створюємо зовнішній ключ від таблиці Order до таблиці house.

Після створення міграції необхідно заповнити метод, яким зміни можна буде відкатити.

```
public function safeDown()
    {
        $this->dropForeignKey('Order_house', 'Order');
        $this->dropForeignKey('OrderStops_Order', 'Order_stops');

        $this->dropTable('Order_log');
        $this->dropTable('Order_stops');
        $this->dropTable('Order');
        $this->dropTable('house');
    }
```

Рисунок 3.1.6 Код методу відкату змін.

Метод dropForeignKey видаляє зовнішній ключ, приймає 2 параметри – назва ключа і назва таблиці в якій він знаходіться. Метод dropTable видаляє таблицю разом із індексами таблиці.

Далі необхідно налаштувати людино-зрозумілі адреси, зараз роутинг відбувається через іднексний файл за допомогою GET параметра r і посилиння виглядає наступним чином - http://Lifttrackeret.com/index.php?r=site/about. Необхідно щоб замість такії реалзації, послиння виглядало http://Lifttrackeret.com/site/about.

Для цього необхідно сконфігувати компонент UrlManager, додавши в файл /config/web у секцію components наступний код

```
'urlManager' => [
```

```
            'enablePrettyUrl' => true,
            'showScriptName' => false,
            'enableStrictParsing' => false,
            'class' => 'yii\web\UrlManager'
        ],
```

Рисунок 3.1.7 Код конфігурації компоненту UrlManager.

Також нам необхідно правильно налаштувати Nginx, що було зроблено раніше. Тепер посилання працюють так як нам це було потрібно.

Тепер можна переходити до створення моделей. Yii2 надає можливість генерувати моделі за допомогою компонента gii. Цей компонент зчитує з бази даних структуру таблиці і генерує модель на її основі.

Для активації компонента необхідно внести зміни у файлі конфігурації аби додати свій IP у якості дозволених.

```
    $config['bootstrap'][] = 'gii';
    $config['modules']['gii'] = [
      'class' => 'yii\gii\Module',
      'allowedIPs' => ['108.39.18.61'],
    ];
```

Рисунок 3.1.8 Код конфігурації модулю gii.

Після того як модуль сконфігуровано, переходимо за посиланням http://Lifttrackeret.com/gii і бачимо результат.

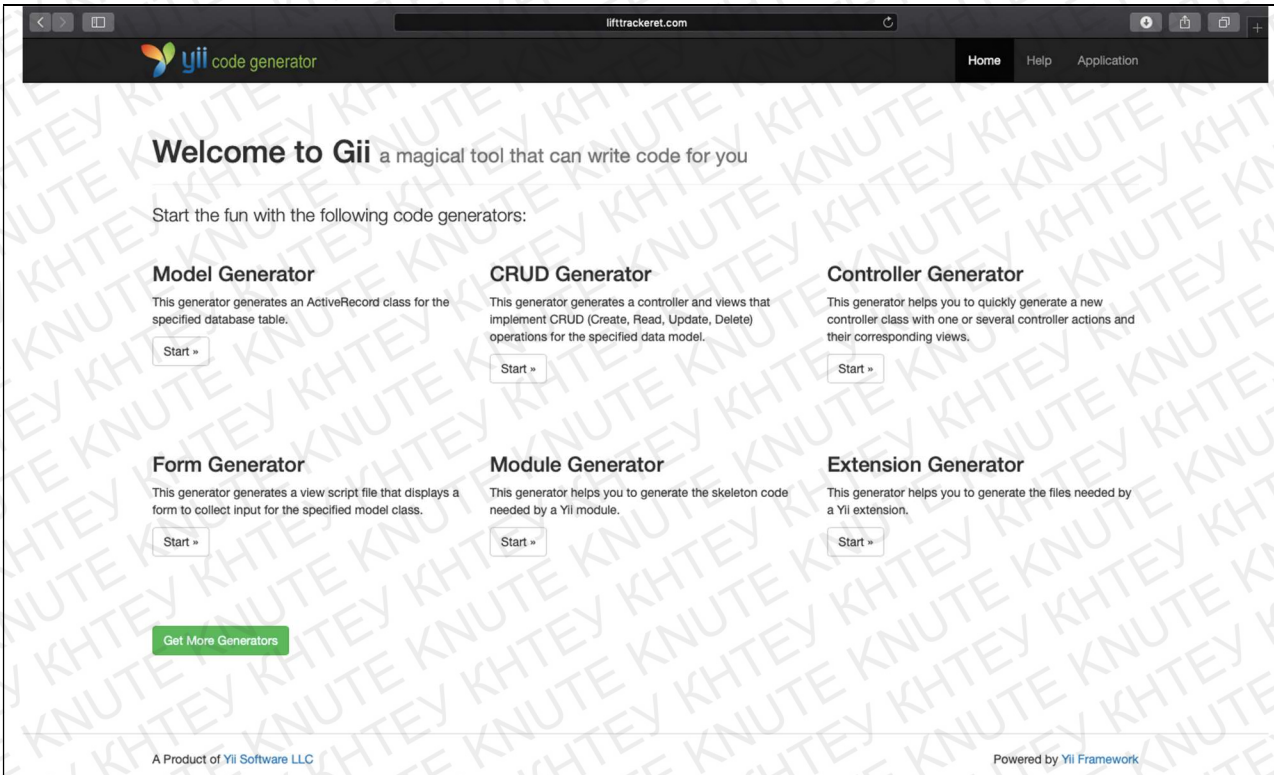

Рисунок 3.1.9 Модуль gii

Для того щоб створити декілька моделей за раз ім’я таблиці написано із зіркою, що означає, що підходять будь-які імена, що починаються із li. Інші налаштування залишаються за замовчанням.

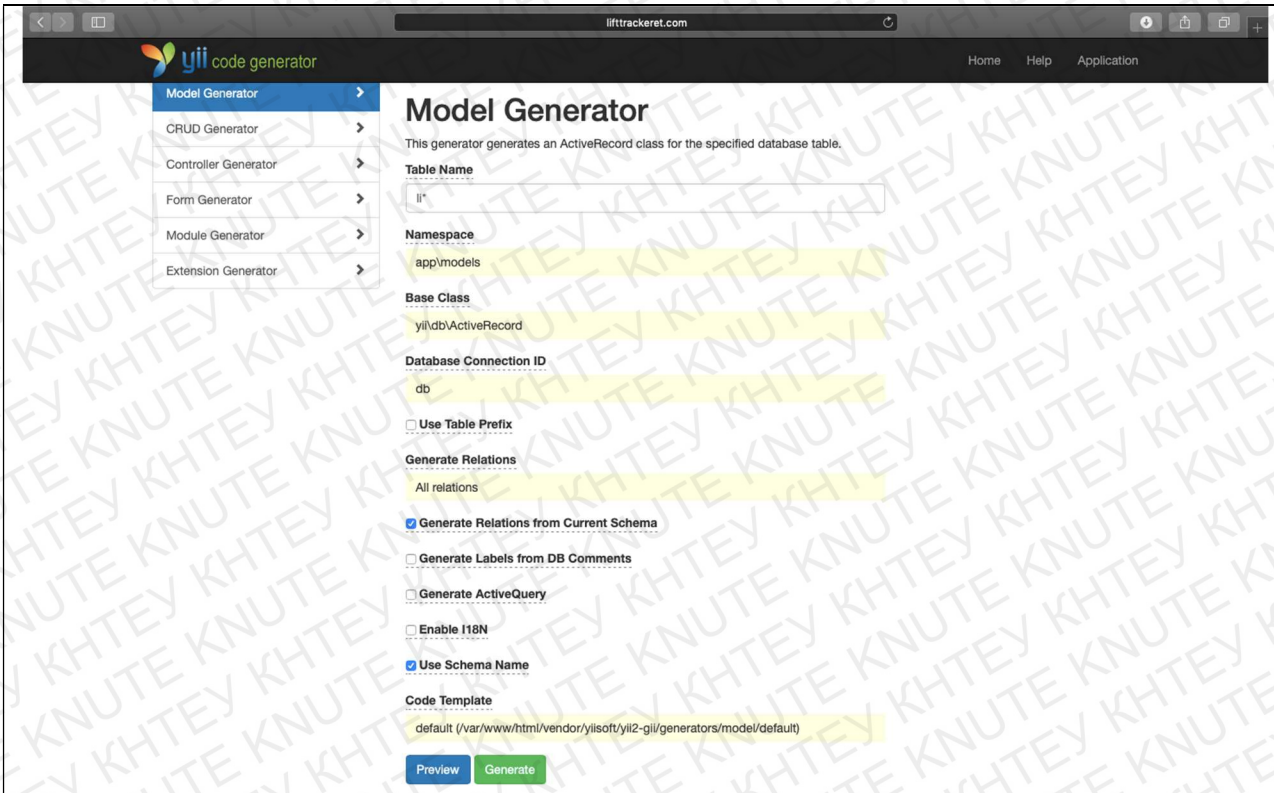

Рисунок 3.1.10 Генерація моделей gii

Після натискання кнопки Preview на екран виводиться список файлів що будуть згенеровані.

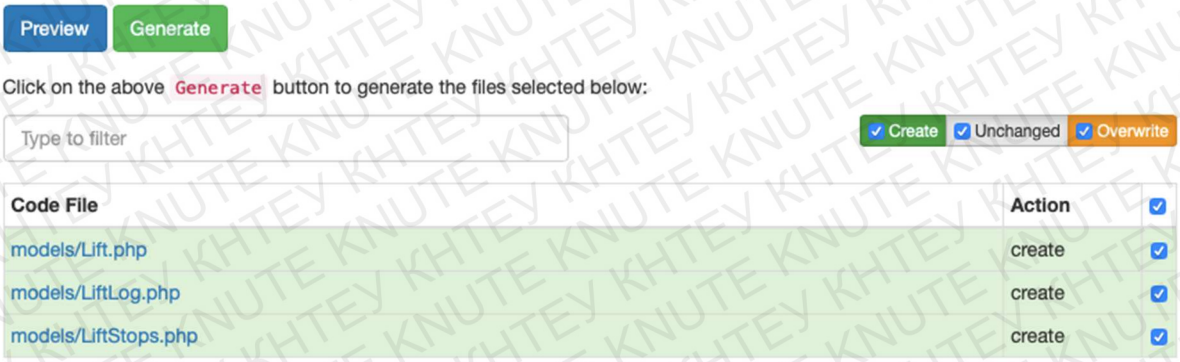

Рисунок 3.1.11 Список файлів що будуть згенеровані

Після того як моделі сгенеровані, їх можна знайти у папці models. Розглянемо для прикладу модель Order що знаходиться у додатку А.

Після блоку із директивами namespace і use, які вказують на простір імен і класи від яких залежить даний клас відповідно, йде безпосередньо клас. Над класом знаходиться блок PHPdoc. PHPDoc — це адаптований стандарт документування Javadoc для використання в PHP. Наразі стандарт коментування має лише формальний статус, проте, планується його закріплення як одного зі стандартів розробки PHP-фреймворків, котрий розробляє група PHP-FIG. Стандарт, який розробляється, отримає номер PSR-5. PHPDoc підтримує як об'єктно-орієнтований, так і процедурний код [24].

Doc-блоки (англ. DocBlock comments) — багаторядкові коментарі в стилі C, розміщені перед елементом, який коментується. Першим символом в коментарі (і на початку кожного рядка коментаря) повинен бути * (символ зірочки). Блоки розділяються порожніми рядками.

Таким чином, Doc-блок заключаються в контейнер, який починається з символів /** та закінчується */.

Для внутрішніх ліній контейнера всі whitespace-символи, які знаходяться до першого символу *, ігноруються.

Пристосування PHPdoc:

- Застосовується для формального опису коду.
- Дозволяє визначити необхідні типи даних вхідних змінних, результату, який отримується внаслідок виконання певних блоків коду.
- Завдяки підтримці в IDE дозволяє організувати коректне автодоповнення коду.
- Дозволяє автоматично створювати документацію до коду.
- Дозволяє пришвидшити рефакторинг коду іншими розробниками.
- Дозволяє швидко згадати, за що відповідає фрагмент коду, не читаючи вміст функції або методу.

- Дозволяє описати для інших програмістів особливості використання фрагменту коду.

Клас має 4 методи. Статичний мето tableName повертає ім’я таблиці із якою пов’язана модель. Метод rules повертає масив із правилами для валідації об’єкту. attributeLabels повертає масив із розшифровкою назв колонок атрибутів. Методи getHouse0 та getOrderStops встановлюють зв’язок між об’єктом і пов’язаними із ним об’єктами із таблиць house і Order_stops відповідно.

Після того як моделі створені потрібно переходити до написання контролеру який буде обробляти запити. Код контроллеру наведено у додатку Б.

Фільтр поведінки у ньому дозволяє робити тільки POST запити на інсуючі ендпоінти і DELETE на ендпоінт stop-node. Метод beforeAction викликається перед викликом методу ендпоінту і перевіряє чи запит є авторизованим чи ні. Якщо запит неавторизований то у відповідь не буде видано нічого і запит буде перервані із кодом 401 – “Потрібна авторизація”. У контроллера встановлюється параметр $enableCsrfValidation = false для того, щоб на нього можнабуло виконувати POST запити не зі сторінок згенерованих фреймворком.

Кожен запит до API оновлює відповідний запис у базі і створює записи логів, які потім можна буде передивитися і проаналізувати.

Контролер для споживацького API наведено у додатку В. Він зроблений за схожою структурою до контролера посилок. Серед особливого те, що авторизація відбувається за іншою таблицею – таблицею користувачів, адже для того щоб створити запит до цього контролера необхідно стати користувачем додатку. Особливістю захисту від атак є те, що контролер

обробляє лише GET запити і не підтримує запити на редакцію і видалення інформації.

Для створення адміністративного кабінету нам потрібно буде сгенерувати контролери і форми для створення, оновлення, видалення і відображення інформації.

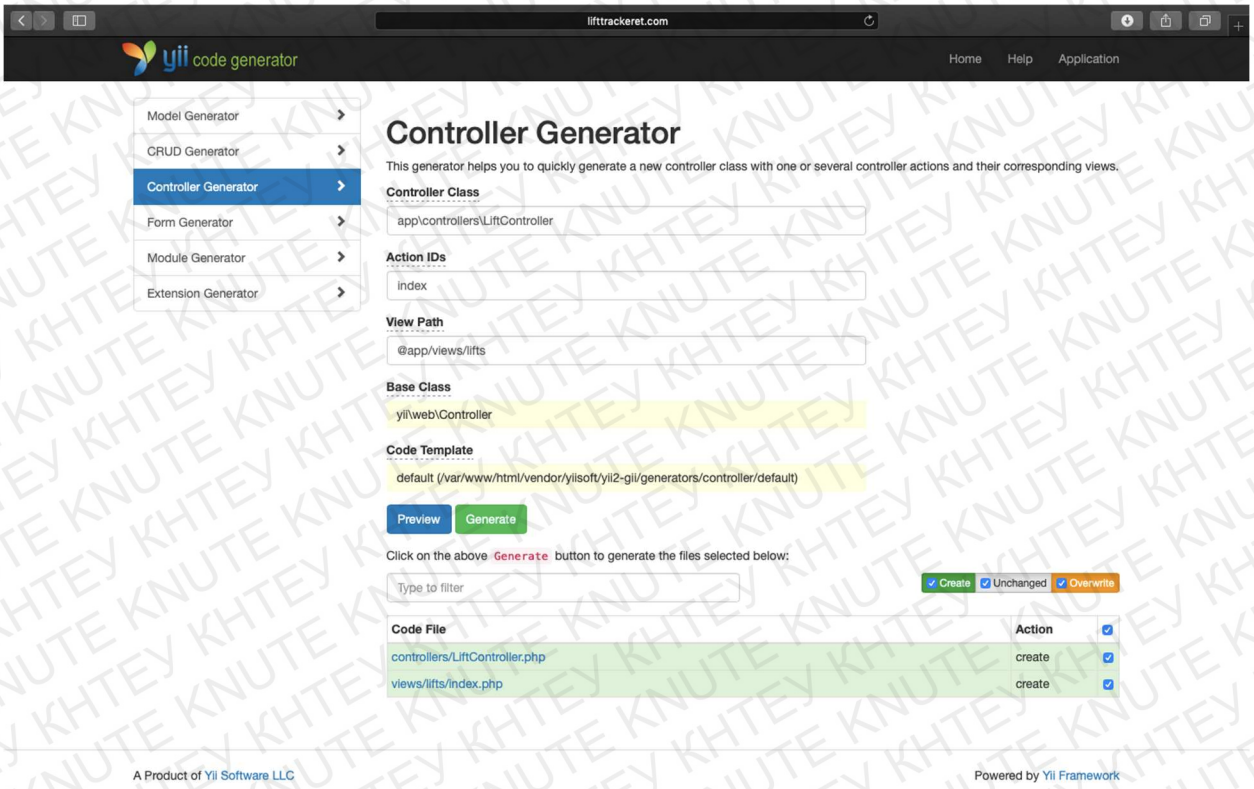

Рисунок 3.1.12 Генерація контролера

Для генерації контролера обираємо необхідний простір імен і папку для файлів відображень.

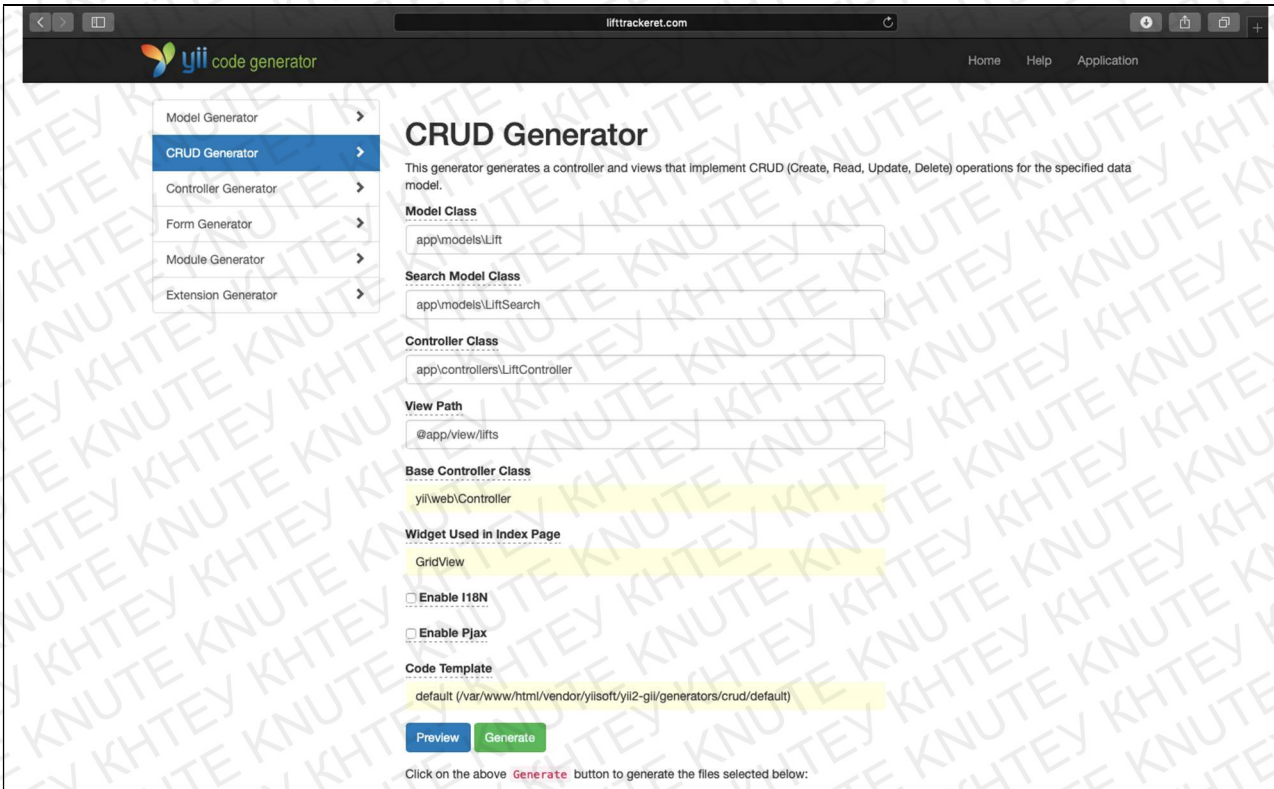

Рисунок 3.1.13 Генерація файлів відображення

Далі генеруємо файли відображень. Для цього переходимо у вкладку CRUD generator і обираємо модель для якої відображення буде генеруватися, контролер і шлях до папки у якій вони будуть сберігатися.

Рисунок 3.1.13 Файли відображення які будуть створені

Потрібно уважно дивитися які файли можуть бути перезаписані, адже контроллер OrderController не створюється у цьому генераторі, а модерується, то зміни, які вже були в нього внесені, можуть бути втрачені при перезаписі. Якщо галочка буде знята, то перезапис не відбудеться.

В цьому випадку перезапис потрібен, адже в контролер OrderController не було внесено ніяких записів, а генератор генерує код, який буде працювати.

OrderController наведено в додатку Г. Серед особливостей можна відмітити, що доступ до нього мають тільки авторизовані на сайті користувачі. Тобто саме адміністратори і ті кому надався доступ. Реалізовано це через поведінку і клас AccessControl у ній.

Отже, у додатку реалізовано API для посилок, яке використовують вузли для надання інформації, користувацьке API, яке може бути використане для відображення інформації користувачеві і адміністративна панель через яку можливо переглядати візуально дані, створювати посилки, покупців й інші записи.

3.2 Тестування веб додатку

Тестування програмного забезпечення — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники
- правильна відповідь для усіх можливих вхідних даних
- виконання функцій за прийнятний час
- практичність
- сумісність з програмним забезпеченням та операційними системами
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення баґів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершено). Процес розробки зазвичай передбачає коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення й забезпечення якості програмного забезпечення, до якого належать всі складові ділового процесу, а не тільки тестування.

Зазвичай, поняття якості обмежується такими поняттями як коректність, надійність, практичність, безпечність, але може містити більше технічних вимог, котрі описані у стандарті ISO 9126. Склад та зміст супутньої документації процесу тестування визначається стандартом IEEE 829—1998 Standard for Software Test Documentation.

Зважаючи на те, що ми маємо чітко описані ендпоінти API і розуміння того як повинно поводити себе програмне забезпечення, для тестування додатку буде використовуватися метод мануального тестування API.

Суть методу полягає в тому, щоб створювати сценарій запитів, виконувати їх вручну і порівнювати вихідні параметри із тим що було очікувано.

Для виконання запитів до API буде використовуватися командна стрічка і програма curl встановлена у ній.

Сценарій 1. Перевірка статусу посилки, повинен бути 1 “Відправлено”. Встановлення посилки статусу 2 “Доставлено”, перевірка статусу, очікуваний результат – статус 2 “Доставлено”.

```
MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$ curl --request GET \
>   --url 'http://lifttrackeret.com/consume-api/status?lift_id=2' \
>   --header 'authorization: Bearer 100-token' \
>   --cookie 'PHPSESSID=d1ee9134d39d02c192c75dd91f3917a4; _csrf=79a6c022f76c92c37f2c64cb13d5bbba53914993278c0687b188505d44066b8ca%253A2%253A%257Bi%253A0%253Bs%253A5%
253A%2522_csrf%2522%253Bi%253A1%253Bs%253A32%253A%2522AnPlQfI5zRWd0LjiNIsLHGWSZlpVozaY%2522%253B%257D' \
>   --form status=1

{"success":true,"status":1}MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$
MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$ curl --request POST \
>   --url http://lifttrackeret.com/open-api/status \
>   --header 'authorization: Bearer 123' \
>   --cookie 'PHPSESSID=d1ee9134d39d02c192c75dd91f3917a4; _csrf=79a6c022f76c92c37f2c64cb13d5bbba53914993278c0687b188505d44066b8ca%253A2%253A%257Bi%253A0%253Bs%253A5%
253A%2522_csrf%2522%253Bi%253A1%253Bs%253A32%253A%2522AnPlQfI5zRWd0LjiNIsLHGWSZlpVozaY%2522%253B%257D' \
>   --form status=2
{"success":true}MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$
MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$ curl --request GET \
>   --url 'http://lifttrackeret.com/consume-api/status?lift_id=2' \
>   --header 'authorization: Bearer 100-token' \
>   --cookie 'PHPSESSID=d1ee9134d39d02c192c75dd91f3917a4; _csrf=79a6c022f76c92c37f2c64cb13d5bbba53914993278c0687b188505d44066b8ca%253A2%253A%257Bi%253A0%253Bs%253A5%
253A%2522_csrf%2522%253Bi%253A1%253Bs%253A32%253A%2522AnPlQfI5zRWd0LjiNIsLHGWSZlpVozaY%2522%253B%257D' \
>   --form status=1
{"success":true,"status":2}MacBook-Air-Yevhenii:docker-nginx-php-mysql yevheniitarasenko$
```

Рисунок 3.2.1 Виконання сценарію 1

Як бачимо перший тест пройдено успішно. Перший запит, із використанням токена користувача, повертає об’єкт, в якому видно що статус має значення 1, потім відбувається запит на зміну статуса, із використанням токена замовлення, сервер відповідає що запит оброблено успішно, третій запит від користувача про статус повідомляє що статус має значення 2. Перший тест пройдено.

Сценарій 2. Перевірка напрямку посилки, повинен бути 1 “Відділення 1”. Встановлення посилки напрямку 2 “Відділення 2”, перевірка напрямку, очікуваний результат – статус 2 “Відділення 2”.

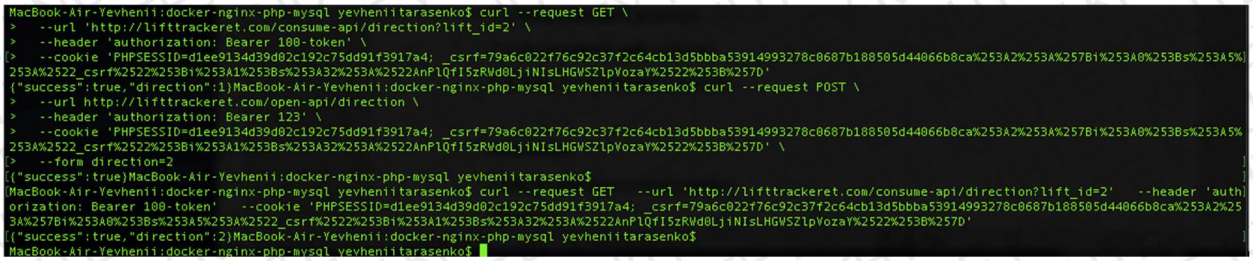

Рисунок 3.2.2 Виконання сценарію 2

Сценарій 2 також виконався успішно, в ньому були задіяні користувацькі API і API вузлу.

Сценарій 3. Перевірка зупинок посилки. Спочатку результат повинен бути порожнім. Встановлюємо нову зупинку на відділеннях 2 і 3. Перевіряємо і ми повинні побачити записи із вузлами 2 і 3.

Рисунок 3.2.3 Виконання сценарію 3

Третій сценарій також виконався успішно. Спочатку ми отримали пусте значення для зупинок, потім додали 2 виклики і отримали їх при перевірці

Тепер необхідно протестувати адміністративну частину. Для цього потрібно авторизуватися у веб-додатку і перейти на сторінку посилок.

На сторінці перегляду посилок можливо переходити за посиланням для створення посилки, редакцію і видалення її.

Для створення нового запису натискаємо зелену кнопку Create Order. Після цього з'явиться сторінка із формою для введення даних про замовлення.

На сторінці створення замовлення потрібно обрати ідентифікатор покупця для якого це замовлення і вказати токен доступу. Інші поля можна залишити зі значеннями за замовчанням.

Після того як необхідні дані було введено потрібно натиснути зелену кнопку Save і сторінку буде переадресовано на сторінку посилки.

Замовлення було створено успішно.

Отже, після тестування додатку стає зрозуміло, що додаток функціонує коректно і видає очікувані результати. Не виявлено програмних помилок будь-якого рівня або неочевидної роботи додатку. Зважаючи на такі результати тести можна вважали пройденими.

3.3 Аналіз отриманої моделі

Отримана модель складається базового набору можливостей, серед яких:

- Адміністрування. Створення, редагування, видалення об’єктів що будуть співпрацювати із API; перегляд і аналіз створеної і накопиченої інформації.
  - API вузла для зміни інформації про посилку. Використовується для наповнення серверної бази даних інформацією.
  - API користувача. Використовується для отримання інформації із сервера і відображення користувачеві.

Отримана модель повністю розроблена і успішно протестована, а це означає що може бути передана у користування користувачам.

Проте у системи є декілька недоліків, які не є критичними при малому навантаженні на сервіс, проте стануть такими при його збільшенні.

При збільшенні кількості посилок, через які вузли будуть виконувати більше запитів на сервер, навантаження на нього буде зростати у прогресії. Адже кожен запит потребує завантаження коду додатку в оперативну пам’ять, підключення до бази даних, виконання авторизації користувача, тощо.

Вирішити цю проблему можливо, якщо перевести підключення закрите API на веб сокети.

WebSocket — це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком. Прикладний програмний інтерфейс WebSocket був стандартизований W3C, крім того протокол WebSocket стандартизований IETF як RFC 6455.

У веб-застосунках доцільно використовувати протокол за необхідності відображення інформації в real-time. У порівнянні з іншими технологіями складний в імплементації на боці сервера. Альтернативна технологія - Server Side Events.

Специфікація протоколу WebSocket визначає дві нові схеми URI, ws: та wss:, для нешифрованого та шифрованого з'єднання відповідно. Поза іменем схеми, решта складових URI визначена загальним синтаксисом URI.

Також необхідним функціоналом є система швидкого інформування про поламку посилки. Вона може бути реалізована безпосередньо у контролері, який обробляє запити на зміну статусу посилки.

Вцілому отримана модель є робочою і готова для використання.

Висновки

Отже, у висновку необхідно зазначити, що:

1. Електронна комерція дуже розвинена в Україні й набуває незвичних форм, таких як комерція в соціальних мережах й мессенджерах.
2. Фреймворк Yii2, як і більшість сучасних фреймворків побудований на парадигмі MVC, а також встановлюється і керує залежностями через менеджер пакетів – Composer.
3. Одразу після встановлення фреймворк Yii2 надає увесь базовий функціонал для створення веб-додатку, для налагодження зв'язку із базою даних необхідно тільки прописати в конфігураційних файлах адресу серверу й логін і пароль для доступу.
4. Для того щоб унікнути рутинних операцій, таких як створення моделей, контролерів і уявлень Yii2 надає компонент gii, який може генерувати типовий код.
5. Yii2 один із найрозповсюженіших PHP-фреймворків у світі.
6. Для роботи із базою даних Yii2 використовує парадигму ActiveRecord. Вона ідеально співпрацює з такою реляційною базою даних як MySQL.
7. Для версіонування структури бази даних Yii2 надає механізм міграцій – запити до бази даних що виконуються тільки один раз і можуть змінювати структуру бази даних або її наповнення.
8. Після безпосередньої розробки додатку обов'язковим етапом викатки є тестування, яке може проводитися як мануально так і автоматично, за допомогою програмних засобів.

Список використаних джерел

1. Андросова Т.В. Міжнародна економіка. У 2-х частинах. Ч. І. Світова система господарювання: навч. посібник / Т.В. Андросова [та ін.]. – Харків: «Видавництво «Форт», 2013. – 287 с.
2. Васвани В. Разработка веб-приложений на PHP / В. Васвани. – Питер, 2014. – 432 с.
3. Гербер Р. Оптимизация ПО. Сборник рецептов / Р. Гербер, А. Бик, К. Смит, К. Тиан. – С-Пб.: Питер, 2010. – 352 с.
4. Дослідження веб-сайтів [Електронний ресурс]. — Режим доступу : https://news.netcraft.com/archives/2018/10/29/october-2018-web-server-survey.html
5. Електронна комерція : навч. посіб. / Полуектова Н. Р., Сабанов С. О. ; Запоріз. ін-т економіки та інформ. технологій. — Запоріжжя : ЗІЕІТ, 2015. — 304 с. : табл. — Бібліогр.: с. 283-285.
6. Електронна комерція : навч. посіб. / Тардаскіна Т. М., Стрельчук Є. М., Терешко Ю. В. ; М-во інфраструктури України, Держ. служба зв’язку, Одес. нац. акад. зв’язку ім. О. С. Попова, Каф. менеджменту та маркетингу. — О. : ОНАЗ ім. О. С. Попова, 2011. — 119 с.
7. Мейерт Д.О. Небольшая книга о HTML/CSS фреймворках [Текст] / Д.О. Мейерт // Орейли. – 2015. – 30 с
8. Офіційний сайт Composer [Електронний ресурс]. – Режим доступу: https://getcomposer.org
9. Офіційний сайт Laravel [Електронний ресурс]. – Режим доступу: https://laravel.com

					КНТЕУ-122-2018		
Зм.	*Аркуш*	*№ документу*	*Підпис*	*Дата*			
Зав. кафедрою		*Краскевич В.Є.*			*Використання фреймворку побудови веб-додатку*	*Сторінка*	*Сторінок*
Керівник		*Нагірна А.М.*				*76*	*102*
Гарант		*Краскевич В.Є.*				*Факультет обліку, аудиту та інформаційних систем, 7м група*	
Розробив		*Тарасенко Є.В.*			*Розробка додатку*		
Перевірив		*Нагірна А.М.*					

10. Офіційний сайт Nginx [Електронний ресурс]. – Режим доступу: https://nginx.org/ru/
11. Офіційний сайт Phalkon [Електронний ресурс]. – Режим доступу: https://phalconphp.com/ru/
12. Офіційний сайт Symfony framework [Електронний ресурс]. – Режим доступу: http://symfony.com/
13. Офіційний сайт Yii framework [Електронний ресурс]. – Режим доступу: https://yiiframework.com.ua/ru/doc/guide/2/start-installation/
14. Офіційний сайт Zend framework [Електронний ресурс]. – Режим доступу: https://framework.zend.com
15. Румянцев, А. П. Світовий ринок послуг [Електронний ресурс] : навч. посіб. : рекомендовано М-вом освіти і науки України / А. П. Румянцев, Ю. А. Коваленко ; М- во освіти і науки України . [Донецьк : ДонНУЕТ, 2011. – 456с.
16. Свинарев Сергей. Еще раз о росте популярности NoSQL.— 2015.— URL: http://www.jetinfo.ru/stati/ silnye-i-slabye-storony-nosql.
17. Сидоров М.О. Створення стилю ефективного програмування / М.О. Сидоров, М.М. Костів // Інженерія програмного забезпечення. – No 3 (15). – 2013.
18. Технологія WEB – програмування : PHP, MYSQL : курс лекций з дисципліни ”Технологія Web-програмування” для студ. 5 курсу спец. ”Інформатика” та для студ. 3 курсу спец. ”Комп’ютер. інженерія” / А. В. Фоменко, Г. О. Козуб ; М-во освіти і науки, молоді та спорту України, ДЗ ”Луган. нац. ун-т ім. Т. Шевченка”. — Луганськ : ЛНУ ім. Т. Шевченка, 2011. — 215 с.
19. Чернишова Л.О. Міжнародна економіка конспект лекцій: для студентів за напрямом 0305 «Економіка і підприємництво» / Л.О. Чернишова, Л.Л.

Носач, В.О. Козуб, Н.Ф. Соболєва – 2-ге вид. перероб. та доп. – Харків: Видавництво «Форт», 2015. – 348 с.

20. Google trends [Електронний ресурс]. – Режим доступу: https://trends.google.com/trends/explore?date=today%205-y&q=Laravel,Yii2,Symfony,Zend,PHALCON
21. Linux, 4 изд. / Стахнов Алексей Александрович; БХВ-Петербург, 2013 - с: 752.
22. MongoDB. The MongoDB 3.0 Manual. — 2016. — URL: https://docs. mongodb.org/manual/
23. Pfeffermann D. Methodological Issues and Challenges in the Production of Official Statistics / D. Pfeffermann / / Journal of Survey Statistics and Methodology. - 2015.
24. PHP-FIG [Електронний ресурс]. – Режим доступу: https://www.php-fig.org
25. PostgreSQL. The PostgreSQL 9.4.5 Documentation.— 2016.— URL: http://www.postgresql.org/docs/9.4/interactive/index.htm.
26. Representational State Transfer (REST) [Електронний ресурс]. – Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Додатки

Додаток А

```
<?php

namespace app\models;

use Yii;

/**
 * This is the model class for table "Order".
 *
 * @property int $id
 * @property int $house
 * @property int $node
 * @property int $direction
 * @property int $entrance
 * @property int $status
 * @property string $access_token
 * @property int $updated_at
 *
 * @property House $house0
 * @property OrderStops[] $OrderStops
 */
class Order extends \yii\db\ActiveRecord
{
    /**
     * @inheritdoc
     */
```

```
    public static function tableName()
    {
        return 'Order';
    }

    /**
     * @inheritdoc
     */
    public function rules()
    {
        return [
            [['house', 'node', 'direction', 'access_token', 'updated_at'], 'required'],
            [['house', 'node', 'direction', 'entrance', 'status', 'updated_at'], 'integer'],
            [['access_token'], 'string', 'max' => 255],
            [['access_token'], 'unique'],
            [['house'], 'exist', 'skipOnError' => true, 'targetClass' => House::className(), 'targetAttribute' => ['house' => 'id']],
        ];
    }

    /**
     * @inheritdoc
     */
    public function attributeLabels()
    {
        return [
            'id' => 'ID',
            'house' => 'House',
```

```
            'node' => 'Node',
            'direction' => 'Direction',
            'entrance' => 'Entrance',
            'status' => 'Status',
            'access_token' => 'Access Token',
            'updated_at' => 'Updated At',
        ];
    }

    /**
     * @return \yii\db\ActiveQuery
     */
    public function getHouse0()
    {
        return $this->hasOne(House::className(), ['id' => 'house']);
    }

    /**
     * @return \yii\db\ActiveQuery
     */
    public function getOrderStops()
    {
        return $this->hasMany(OrderStops::className(), ['Order_id' => 'id']);
    }
}
```

```
<?php

namespace app\controllers;

use app\models\Order;
use app\models\OrderLog;
use app\models\OrderStops;
use Yii;
use yii\web\Controller;
use yii\filters\VerbFilter;
use yii\web\Response;

class OpenApiController extends Controller
{
    public $enableCsrfValidation = false;

    /**
     * @var Order $identity
     */
    protected $identity;

    /**
     * @inheritdoc
     */
    public function behaviors()
    {
        return [
```

```
            'verbs' => [
                'class' => VerbFilter::className(),
                'actions' => [
                    'status' => ['post'],
                    'direction' => ['post'],
                    'stop-node' => ['post', 'delete'],
                ],
            ],
        ];
    }

    public function beforeAction($action)
    {
        $authHeader = Yii::$app->request->getHeaders()->get('Authorization');
        if ($authHeader !== null && preg_match('/^Bearer\s+(.*?)$/', $authHeader, $matches)) {
            $identity = Order::findIdentityByAccessToken($matches[1], get_class($this));
            if ($identity === null) {
                return $this->handleFailure(Yii::$app->response);
            }

            $this->identity = $identity;
        } else {
            return $this->handleFailure(Yii::$app->response);
        }

        Yii::$app->response->format = Response::FORMAT_JSON;
```

```
    return parent::beforeAction($action);
}

protected function handleFailure(Response $response)
{
    $response->setStatusCode(401);
    return false;
}

public function actionStatus() {
    $status = Yii::$app->request->post('status');
    if (!$status) {
        return [
            'success' => false,
            'message' => 'parameter status missed'
        ];
    }
    if (!in_array($status, Order::statuses())) {
        return [
            'success' => false,
            'message' => 'status out of range'
        ];
    }

    if ($status != $this->identity->status) {
        $this->saveState();
        $this->identity->status = $status;
```

```
            $this->identity->updated_at = time();
            $this->identity->save();
        }
        return ['success' => true];
    }

    public function actionDirection() {
        $direction = Yii::$app->request->post('direction');
        if (!$direction) {
            return [
                'success' => false,
                'message' => 'parameter direction missed'
            ];
        }
        if (!in_array($direction, Order::directions())) {
            return [
                'success' => false,
                'message' => 'direction out of range'
            ];
        }

        if ($direction != $this->identity->direction) {
            $this->saveState();
            $this->identity->direction = $direction;
            $this->identity->updated_at = time();
            $this->identity->save();
        }
        return ['success' => true];
```

```
    }

    public function actionStopNode() {
      $node = Yii::$app->request->{
        Yii::$app->request->isPost ? 'post' : 'get'
      }('node');
      if (!$node) {
        return [
          'success' => false,
          'message' => 'parameter node missed'
        ];
      }

      if (Yii::$app->request->getMethod() == "DELETE") {
        $model = OrderStops::findOne([
          'Order_id' => $this->identity->id,
          'node' => $node,
          'arrival_at' => 0
        ]);
        if (!$model) {
          return [
            'success' => true,
            'warning' => 'nothing deleted'
          ];
        }
        $model->arrival_at = time();
        return [ 'success' => $model->save() , 'message' => $model-
>getErrors()];
```

```
        } else {
            $model = new OrderStops();
            $model->setAttributes([
                'Order_id' => $this->identity->id,
                'node' => $node,
                'created_at' => time(),
                'arrival_at' => 0
            ]);
            return [
                'success' => $model->save(),
                'message' => $model->getErrors()
            ];
        }
    }

    protected function saveState()
    {
        $OrderLog = new OrderLog();
        $OrderLog->setAttributes($this->identity->getAttributes());
        $OrderLog->save();
    }
}
```

Додаток В

```
<?php

namespace app\controllers;
```

```
use app\models\Order;
use app\models\OrderLog;
use app\models\OrderStops;
use app\models\User;
use Yii;
use yii\helpers\ArrayHelper;
use yii\web\Controller;
use yii\filters\VerbFilter;
use yii\web\Response;

class ConsumeApiController extends Controller
{
    /**
     * @var User $identity
     */
    protected $identity;
    /**
     * @var Order $Order
     */
    protected $Order;

    /**
     * @inheritdoc
     */
    public function behaviors()
    {
        return [
            'verbs' => [
```

```
                'class' => VerbFilter::className(),
                'actions' => [
                    'status' => ['get'],
                    'direction' => ['get'],
                    'stop-node' => ['get'],
                ],
            ],
        ];
    }

    public function beforeAction($action)
    {
        $authHeader = Yii::$app->request->getHeaders()->get('Authorization');
        if ($authHeader !== null && preg_match('/^Bearer\s+(.*?)$/', $authHeader, $matches)) {
            $identity = User::findIdentityByAccessToken($matches[1], get_class($this));
            if ($identity === null) {
                return $this->handleFailure(Yii::$app->response);
            }

            $this->identity = $identity;
        } else {
            return $this->handleFailure(Yii::$app->response);
        }

        Yii::$app->response->format = Response::FORMAT_JSON;
```

```
        $Order_id = (int)Yii::$app->request->get('Order_id');
        if (!$Order_id) {
            return $this->handleFailure(Yii::$app->response, 400);
        }

        $model = Order::findOne($Order_id);
        if (!$model) {
            return $this->handleFailure(Yii::$app->response, 404);
        }
        $this->Order = $model;

        return parent::beforeAction($action);
    }

    protected function handleFailure(Response $response, int $code = 401)
    {
        $response->setStatusCode($code);
        return false;
    }

    public function actionStatus() {
        return [
            'success' => true,
            'status' => $this->Order->status
        ];
    }

    public function actionDirection() {
```

```
            return [
                'success' => true,
                'direction' => $this->Order->direction
            ];
        }

        public function actionStopNode() {
            $stops = OrderStops::find()
                ->select(['node','created_at'])
                ->where([
                    'Order_id' => $this->Order->id,
                    'arrival_at' => 0
                ])
                ->asArray()
                ->all();
            return [
                'success' => true,
                'stops' => ArrayHelper::map($stops, 'created_at', 'node')
            ];
        }

    }
```

```
<?php

namespace app\controllers;

use Yii;
use app\models\Order;
use app\models\OrderSearch;
use yii\filters\AccessControl;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

/**
 * OrderController implements the CRUD actions for Order model.
 */
class OrderController extends Controller
{
    /**
     * @inheritdoc
     */
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'rules' => [
                    [
```

```
                'allow' => true,
                'roles' => ['@'],
            ],
        ],
    ],
    'verbs' => [
        'class' => VerbFilter::className(),
        'actions' => [
            'delete' => ['POST'],
        ],
    ],
];
}

/**
 * Lists all Order models.
 * @return mixed
 */
public function actionIndex()
{
    $searchModel = new OrderSearch();
    $dataProvider = $searchModel->search(Yii::$app->request->queryParams);

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}
```

```
/**
 * Displays a single Order model.
 * @param integer $id
 * @return mixed
 * @throws NotFoundHttpException if the model cannot be found
 */
public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}

/**
 * Creates a new Order model.
 * If creation is successful, the browser will be redirected to the 'view' page.
 * @return mixed
 */
public function actionCreate()
{
    $model = new Order();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]);
    }

    return $this->render('create', [
```

```
        'model' => $model,
    ]);
}

/**
 * Updates an existing Order model.
 * If update is successful, the browser will be redirected to the 'view' page.
 * @param integer $id
 * @return mixed
 * @throws NotFoundHttpException if the model cannot be found
 */
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]);
    }

    return $this->render('update', [
        'model' => $model,
    ]);
}

/**
 * Deletes an existing Order model.
 * If deletion is successful, the browser will be redirected to the 'index' page.
 * @param integer $id
```

```
     * @return mixed
     * @throws NotFoundHttpException if the model cannot be found
     */
    public function actionDelete($id)
    {
        $this->findModel($id)->delete();

        return $this->redirect(['index']);
    }

    /**
     * Finds the Order model based on its primary key value.
     * If the model is not found, a 404 HTTP exception will be thrown.
     * @param integer $id
     * @return Order the loaded model
     * @throws NotFoundHttpException if the model cannot be found
     */
    protected function findModel($id)
    {
        if (($model = Order::findOne($id)) !== null) {
            return $model;
        }

        throw new NotFoundHttpException('The requested page does not exist.');
    }
}
```

```
{
  "name": "yiisoft/yii2-app-basic",
  "description": "Yii 2 Basic Project Template",
  "keywords": ["yii2", "framework", "basic", "project template"],
  "homepage": "http://www.yiiframework.com/",
  "type": "project",
  "license": "BSD-3-Clause",
  "support": {
    "issues": "https://github.com/yiisoft/yii2/issues?state=open",
    "forum": "http://www.yiiframework.com/forum/",
    "wiki": "http://www.yiiframework.com/wiki/",
    "irc": "irc://irc.freenode.net/yii",
    "source": "https://github.com/yiisoft/yii2"
  },
  "minimum-stability": "stable",
  "require": {
    "php": ">=5.4.0",
    "yiisoft/yii2": "~2.0.5",
    "yiisoft/yii2-bootstrap": "~2.0.0",
    "yiisoft/yii2-swiftmailer": "~2.0.0"
  },
  "require-dev": {
    "yiisoft/yii2-debug": "~2.0.0",
    "yiisoft/yii2-gii": "~2.0.0",
    "yiisoft/yii2-faker": "~2.0.0",

    "codeception/base": "^2.2.3",
```

```
    "codeception/verify": "~0.3.1",
    "codeception/specify": "~0.4.3"
  },
  "config": {
    "process-timeout": 1800
  },
  "scripts": {
    "post-install-cmd": [
      "yii\\composer\\Installer::postInstall"
    ],
    "post-create-project-cmd": [
      "yii\\composer\\Installer::postCreateProject",
      "yii\\composer\\Installer::postInstall"
    ]
  },
  "extra": {
    "yii\\composer\\Installer::postCreateProject": {
      "setPermission": [
        {
          "runtime": "0777",
          "web/assets": "0777",
          "yii": "0755"
        }
      ]
    },
    "yii\\composer\\Installer::postInstall": {
      "generateCookieValidationKey": [
        "config/web.php"
```

```
        ]
    }
},
"repositories": [
    {
        "type": "composer",
        "url": "https://asset-packagist.org"
    }
]
}
```

MyISAM	InnoDB	MERGE	HEAP (MEMORY)	ARCHIVE	CSV	BLACKHOLE
транзакцій немає	макс. диск: 64Тб	Використовується для об'єднання однакових таблиць в одну	транзакцій немає	макс. диск: немає обмежень	зберігає таблиці в CSV форматі	дані йдуть «в нікуди»
макс. диск: 256Тб	повна підтримка транзакцій (4 рівня ізоляції)	таблиці повинні мати ідентичну структуру	блокування таблиці	блокування запису не працює	дозволяє редагувати таблиці зовнішніми додатками	пишуться логи
блокування таблиць	блокування запису (не таблиці), два види блокувань (SHARED, EXCLUSIVE)	порядок стовпців повинен збігатися	реплікація	DELETE, REPLACE, UPDATE, ORDER BY, тип BLOB INSERT буферизується і «зливається» з великою затримкою	погано документований, є відкриті баги	
повнотекстовий пошук	повнотекстовий індекс: немає	DROP не видаляє вихідних таблиць	макс. довжина ключа: 500 байт			
відсутня кластеризація	безпечна для транзакцій	таблиці можуть бути в іншій базі даних	всі дані губляться при зупинці сервера (сама таблиця залишається)	дуже повільний SELECT		
відсутня підтримання цілісності і зовнішні ключі	індекси кластеризуються для «типових запитів»	можна використовувати для алиасів (для однієї таблиці)	формат зберігання: завжди fixed-length row			
реплікація	підтримка цілісності (зовнішні ключі)	не можна користуватися FULLTEXT	пам'ять не вивільняється при видаленні			

		search	запису (використовується для вставки нових)
макс. індексів: 64	може використовуватися на ОС з обмеженим розміром файлу	не можна змішувати тимчасові і не тимчасові таблиці	
макс. записів: 2 ^ 32	безліч налаштувань для оптимізації	повільна при читанні по ключу	
макс. довжина ключа 1000 байт	дозволяє використовувати Raw Disk для таблиць в обхід ФС	REPLACE не працює	
ключі займають місце на диску до (макс.): (key_length + 4) /0.67	за замовчуванням включений AUTOCOMMIT (SET autocommit = 1)	не відслідковуються зміни в структурі вихідних таблиць (таблиця буде поламана)	
чутливі до «падіння» сервера, складно відновлювати	автоматичне виявлення дедлоків (deadlocks)		
при відсутності «дірок» (gaps) - вставки не конкуренті (блокування не відбувається)			
можливо зберігати файли			

даних і індексів на різних пристроях кожен стовпець може мати свою систему кодування макс. сума довжин VARCHAR і CHAR: 64к						