МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Створення власного CRUD-генератора на PHP – фреймворку Codeigniter на основі аналізу бібліотек генерації коду фреймворків Yii2 та Laravel**

Виконав: студент 2 курсу, групи 8.1212-іпз-2-дн
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Д. О. Мельник
(ініціали та прізвище)
Керівник доцент, к.т.н. Н.П. Полякова
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення

Рівень вищої освіти другий (магістерський)

Спеціальність 121 Інженерія програмного забезпечення
(код та назва)

Освітня програма Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Т.В. Критська
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Мельнику Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1.Тема роботи Створення власного CRUD-генератора на PHP – фреймворку Codeigniter на основі аналізу бібліотек генерації коду фреймворків Yii2 та Laravel

керівник роботи доцент, к.т.н. Н.П.Полякова

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від 09.10. 2023 р. №1577-с

2. Строк подання студентом кваліфікаційної роботи 05.03.2024

3. Вихідні дані магістерської роботи
- комплект нормативних документів;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження технологій, методів та засобів проектування веб-сайтів;
- створення програмного продукту та його опис;
- розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
15 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.22	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.22	виконано
3	Аналіз існуючих методів рішення	13.09-14.09.22	виконано
4	Розробка додатку на Laravel	15.09-20.09.22	виконано
5	Вивчення бібліотеки Laravel/Nayjest Grid	21.09-26.09.22	виконано
6	Розробка додатку на Yii2	27.09-28.09.22	виконано
7	Вивчення бібліотеки Yii2/Gii	29.09-15.10.22	виконано
8	Вивчення бібліотеки Codeigniter4/GroceryCRUD	16.10-01.11.22	виконано
9	Розробка генератору коду на Codeigniter4	02.11-20.11.22	виконано
10	Розробка шаблону контролера	21.11.22-01.03.23	виконано
11	Розробка шаблону моделі	02.03.23-15.03.23	виконано
12	Розробка шаблонів представлень	16.03.23-15.04.23	виконано
13	Тестування додатків	16.04.23-15.05.23	виконано

Студент _____ Мельник Д. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Полякова Н. П.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 98

Рисунків: 11

Таблиць: 2

Джерел: 14

Лістингів: 28

Мельник Д. О. Порівняння код генератора Gii Yii2 та Laravel Nayjest/Grids та розробка генератора CRUD на Codeigniter 4 : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Н. П. Полякова. Запоріжжя : ЗНУ, 2023. 100 с.

Мета і завдання дослідження полягають у вивчені структури зберігання структури таблиць бази даних та вивченні і розробці CRUD генератора мовою програмування php на базі структури таблиць проекту.

У процесі дослідження були розглянуті приклади CRUD контролерів на різних фреймворках та на основі структури цих контролерів був розроблений генератор CRUD контролеру на фреймворці Codeigniter 4.

Ключові слова: PHP-фреймворк, CRUD, MySQL, Codeigniter, Laravel, Yii, база даних.

SUMMARY

Pages: 98

Figures: 11

Tables: 2

Sources: 14

Listings: 28

Melnyk D. O. A Comparison of Gii Yii2 Generator Code and Laravel Nayjest/Grids, and Development of a CRUD Generator in Codeigniter 4: Master's thesis in the specialty 121 "Software Engineering" / academic supervisor N. P. Polyakova. Zaporizhzhia: ZNU, 2023. 100 p.

The aim and objectives of the research involve studying the structure of storing table structures in a database and exploring the development of a CRUD generator using the PHP programming language based on the project's table structure.

During the research process, examples of CRUD controllers were examined in various frameworks, and based on the structure of these controllers, a CRUD controller generator was developed on the Codeigniter 4 framework.

Keywords: PHP framework, CRUD, MySQL, Codeigniter, Laravel, Yii, database.

ЗМІСТ

ВСТУП

Актуальність теми

Тема генерації CRUD (створення, читання, оновлення, видалення) контролерів є актуальною в розробці програмного забезпечення, особливо в контексті веб-додатків. CRUD-операції є базовими операціями для управління даними в багатьох додатках, і їх автоматизація може значно спростити і прискорити процес розробки.

Генерація CRUD контролерів дозволяє автоматично створювати основний функціонал для кожного з етапів CRUD-операцій (створення, читання, оновлення, видалення) на основі моделі даних. Це може включати генерацію коду для маршрутів, контролерів, переглядів та моделей, які використовуються для взаємодії з базою даних.

Однією з головних переваг генерації CRUD контролерів є економія часу і зусиль розробника. Замість написання повторюваного коду для кожної операції CRUD, розробник може скористатися інструментом, який автоматично генерує базовий функціонал. Це зменшує кількість повторюваного коду, підвищує продуктивність та знижує ймовірність помилок.

Крім того, генерація CRUD контролерів сприяє стандартизації розробки. Код, створений за допомогою генератора, має однакову структуру та інтерфейс для всіх операцій CRUD. Це полегшує зрозуміння коду розробниками, спрощує супровід та підтримку додатків.

Зважаючи на зростаючу популярність веб-розробки та необхідність швидкого розгортання додатків, тема генерації CRUD контролерів залишається актуальною. Інструменти, які надають можливість автоматично генерувати CRUD контролери, широко використовуються у різних фреймворках та бібліотеках розробки, що сприяє покращенню продуктивності розробників та якості програмного забезпечення.

Мета і завдання дослідження

Мета і завдання дослідження полягають у досліджені структури зберігання таблиць бази даних та вивченні і розробці CRUD генератора мовою програмування php на базі структури таблиць проекту.

Об’єкт дослідження

Структура бази даних проекту, способи експортування структури бази даних проекту за допомогою SQL скриптів.

Предмет дослідження

Згенеровані CRUD контролери, за допомогою скрипта, який генерує їх на базі властивостей колонок таблиць бази даних.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей існуючих PHP фреймворків.
2. Аналіз SQL запитів, які необхідні для парсингу структури таблиці у базі даних.
3. Аналіз бібліотек генерації CRUD контролерів.
4. Аналіз різних CRUD контролерів, які розроблені без використання CRUD генераторів коду.
5. Аналіз методів в контролері, які потрібні для видалення, пошуку та посторінкової навігації.
6. Синтез знань, отриманих в процесі дослідження проблеми та методів її вирішення.
7. Експериментування з використанням різних способів реалізації методів “create”, “edit”, “delete”, “view”.
8. Експериментування зі способами валідації сутності на основі структури таблиці.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що була розроблена нова бібліотека-скрипт для генерації CRUD контролера, моделі, та представлення мовою php на фреймворці Codeigniter 4

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що бібліотеку — скрипт можна використовувати для генерації CRUD контролера, моделі, та представлення у нових проектах з базою даних mysql будь-якої складності.

Апробація одержаних результатів

Результати дослідження були представлені на XVI науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023» [14], а також на III Всеукраїнській науково-практичній конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» Інженерного навчально-наукового інституту Запорізького національного університету [9].

Глосарій

CRUD - це абревіатура, що означає чотири основні операції, які можна виконувати з даними у багатьох системах керування базами даних (СКБД) або програмах розробки програмного забезпечення — create (створити), read (читати), update (оновити), delete (видаляти).

PHP (Hypertext Preprocessor) - це скриптова мова програмування, яка використовується для розробки веб-додатків і динамічних веб-сторінок.

PHP-фреймворк - це набір інструментів, бібліотек і структур, які допомагають розробникам побудувати веб-додатки швидко і ефективно.

MySQL - це відкрите програмне забезпечення, яке представляє собою систему управління базами даних (СКБД). Вона розроблена для зберігання, керування та доступу до великих обсягів даних.

MVC (Model-View-Controller) - це архітектурний шаблон, який використовується в розробці програмного забезпечення для поділу додатку на три основні компоненти: модель (Model), представлення (View) та контролер (Controller).

Контролер (Controller) є одним з компонентів архітектурного шаблону MVC (Model-View-Controller). Контролер виконує роль посередника між моделлю (дані) та представленням (інтерфейс користувача).

Модель (Model): Відповідає за представлення даних та бізнес-логіки додатку. Модель забезпечує доступ до даних, виконує операції збереження, оновлення та отримання даних і може містити логіку, пов'язану з обробкою цих даних.

Представлення (View): Відповідає за відображення даних користувачу. Представлення відображає дані, отримані з моделі, в підходящому форматі для користувача. Воно може бути веб-сторінкою, шаблоном або іншим інтерфейсом користувача.

CodeIgniter - це легкий і швидкий PHP-фреймворк, призначений для розробки веб-додатків. Він надає просту та елегантну архітектуру, що сприяє швидкий розробці і розгортанню додатків.

Yii - це високопродуктивний PHP-фреймворк, призначений для розробки веб-додатків.

Laravel - це популярний PHP-фреймворк з відкритим вихідним кодом, який використовується для розробки веб-додатків і веб-сервісів. Він заснований на концепції MVC (Model-View-Controller) і надає широкий набір інструментів та функціоналу для швидкої і ефективної розробки додатків.

Gii - це генератор коду (Code Generator) для фреймворку Yii. Він дозволяє автоматично створювати початковий код для моделей, контролерів, форм

та інших компонентів, що прискорює процес розробки веб-додатків на основі Yii.

Nayjest/Grids - це PHP-бібліотека, яка надає можливості для швидкої розробки та відображення сіток (таблиць) з даними у веб-додатках. Вона пропонує простий і зручний спосіб побудови та налаштування сіток з можливістю сортування, фільтрації, пагінації, редагування і багатьох інших функцій.

Миграція MySQL (MySQL migration) - це процес перенесення (міграції) структури бази даних MySQL з однієї версії або стану в інший. Це включає створення, зміну або видалення таблиць, стовпців, індексів, обмежень, процедур, функцій і т.і.

Controller action (дія контролера) - це функція або метод, який виконується в контексті контролера веб-додатку. В контексті архітектурного шаблону MVC (Model-View-Controller), дії контролера відповідають на запити користувача та виконують необхідну логіку для обробки цих запитів.

Сутність (Entity) - об'єкт або поняття, яке представляється та зберігається в базі даних. Сутність визначається набором атрибутів (полів), які описують її характеристики або властивості.

Паттерн (англ. pattern) в програмуванні є рекомендованою або стандартною способом організації коду для вирішення типових проблем або завдань. Це концептуальна модель, яка описує розроблене рішення, яке можна використовувати в аналогічних ситуаціях.

HTML (HyperText Markup Language) - це стандартна мова розмітки, що використовується для створення структури та відображення веб-сторінок у Інтернеті. HTML визначає структуру документа за допомогою різних тегів (tags), які оточують різні елементи контенту, такі як заголовки, параграфи, зображення, покликання та інші.

Валідація (Validation) у програмуванні відноситься до процесу перевірки коректності та відповідності даних заданим правилам або форматам. Валідація

використовується для переконання, що введені дані є допустимими та відповідають очікуваному формату, перед тим як вони будуть оброблені або збережені.

SQL запит (SQL query) - це команда або запит, який виконується в базі даних з метою отримання, зміни, видалення або вставки даних. SQL-запити використовуються для взаємодії з базою даних і виконання різних операцій над даними.

Zend Framework (ZF) - це відкритий PHP-фреймворк, який надає розширені інструменти та компоненти для розробки веб-додатків. Він базується на архітектурі Model-View-Controller (MVC) і дозволяє розбити додаток на логічні компоненти, такі як моделі, контролери та представлення.

CakePHP - це відкритий PHP-фреймворк для розробки веб-додатків. Він заснований на архітектурі Model-View-Controller (MVC) і надає зручні та потужні інструменти для швидкої розробки додатків з мінімальним зусиллям.

Layout (макет) - це HTML код, який використовується для створення однорідного зовнішнього вигляду для різних сторінок вашого веб-додатку. Він включає заголовок, навігаційне меню, футер та інші елементи, які з'являються на багатьох сторінках додатку.

jQuery - це широко використовувана бібліотека JavaScript, яка спрощує роботу з маніпулюванням HTML-документами, обробкою подій, виконанням асинхронних запитів до сервера (AJAX) і багато іншого. Вона надає зручний інтерфейс для взаємодії з DOM-елементами, роботи з анімацією, обробки подій та виконання HTTP-запитів.

AJAX (Asynchronous JavaScript and XML) - це технологія, яка дозволяє виконувати асинхронні запити до сервера з використанням JavaScript без необхідності перезавантаження сторінки.

JavaScript - це мова програмування, яка використовується для розробки динамічних інтерактивних веб-сторінок. Вона є однією з ключових технологій у веб-розробці і використовується для створення різноманітних функцій та ефектів, взаємодії з користувачем, обробки даних.

CSS (Cascading Style Sheets) - це мова стилізації веб-документів, яка використовується для оформлення та зовнішнього вигляду веб-сторінок. За допомогою CSS можна задавати розміри, кольори, шрифти, межі, фони, вирівнювання та інші властивості для елементів HTML, щоб зробити сторінку більш привабливою та структурованою.

HTML-форма (HTML form) - це елемент мови розмітки HTML, який дозволяє створювати інтерактивні форми на веб-сторінках. Вона дозволяє користувачам вводити дані і надсилати їх на сервер для подальшої обробки. Форми можуть містити різні елементи вводу, такі як текстові поля, радіо-кнопки, прапорці, списки вибору, кнопки тощо.

CRM (Customer Relationship Management) - поняття, що охоплює концепції, котрі використовуються компаніями для управління взаємовідносинами зі споживачами, включаючи збір, зберігання й аналіз інформації про споживачів, постачальників, партнерів та інформації про взаємовідносини з ними.

ORM (Object–relational mapping) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних»

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ІСНУЮЧИХ PHP ФРЕЙМВОРКІВ НА ОСНОВІ ДОДАТКУ CRM

1.1 Огляд проведених досліджень та наукових джерел

Код генератор CRUD PHP є інструментом, який допомагає швидко створювати прості операції створення, читання, оновлення та видалення даних до бази даних. CRUD (створення, читання, оновлення та видалення) є базовими операціями, які використовуються в більшості веб-додатків для взаємодії з базою даних. Наразі існує декілька PHP-фреймворків у яких підтримується код генерація. Наприклад: Laravel, Symfony, Yii, Phalcon.

Для вирішення проблем, викликаних зростаючою складністю проектів, було запропоновано декілька методів програмування на PHP, таких як процедурне кодування PHP та об’єктно-орієнтоване програмування. У результаті цього швидкого розвитку з’явилося кілька фреймворків, таких як PHP-фреймворк CodeIgniter, щоб полегшити завдання розробки. Дійсно, як зазначено в [1], вони були вдосконалені та «стали зручними інструментами для розробників для ефективного створення складних програм».

У різних фреймворках є різні обмеження по можливостям генерації коду. У деяких фреймворках бібліотека код генерації не підтримує генерацію серверної валідації, у деяких не підтримується також генерація клієнтської валідації [1].

У [2] проводится порівняння існуючих PHP фреймворків за версією PHP, ORM, та рушиїм представлень:

Таблиця 1 — *Порівняння php фреймворків*

Назва фреймворку	Ліцензія	Версія PHP	ORM	Тип генератора коду	Рушій представлень	CRUD генератор
Laravel	MIT	>= 5.4	Eloquent	Artisan CLI	Blade	
Symfony	MIT	>=5.5.9	Doctrine 2, Propel	CLI	PHP, Twig	Sensio Generator Bundle
Codeigniter	MIT	>= 5.1.6	Doctrine, Active Record		PHP, Twig	
Yii	BSD	>=5.4	Database Access Object, Active Record	Yii, CLI, Gii, (Web based)	Razor, Smarty, Twig	Gii
Cake PHP	MIT	>=5.2.8	Custom (Cake PHP's standard)	CLI	Custom, Smarty, Twig	Scaffolding
Phalcon	BSD		Phalcon	Phalcon Developer Tools	Volt, PHP	Scaffolding
Zend	BSD	>=5.3	Doctrine			

Вибір PHP фреймворків є доцільним для розробки додатків, через його популярність (популярність PHP фреймворків сягає 81.7%[3] від усіх веб-додатків у інтернет), документованість, та підтримку та подальшій розвиток мови програмування PHP.

Код генератор CRUD PHP зазвичай використовується для створення адміністративних панелей, веб-сайтів-каталогів, інтернет-магазинів та інших веб-додатків, де є потреба у простих операціях з базою даних [4]. Він може також збільшити швидкість розробки і зменшити кількість потрібних розробників, що робить його корисним інструментом для проектів з обмеженими бюджетами або термінами. Генератор коду робить розробку веб-додатків більш ефективною та зменшує кількість ручної роботи.

Зазвичай, розробники використовують генератор CRUD, щоб створити шаблонний код для операцій CRUD на основі бази даних, яку вони використовують. За допомогою спеціально розробленої форми веб-інтерфейсу, розробник може вибрати таблиці в базі даних та задати параметри, такі як назви стовпців, типи даних та інші характеристики [6].

Після того, як розробник встановив налаштування генератора CRUD, програма генерує код для кожної операції CRUD - створення, читання, оновлення та видалення. Зазвичай, згенерований код включає в себе класи моделі (Model), які відображають таблиці в базі даних та забезпечують методи для виконання операцій CRUD над ними. Крім того, згенерований код також може містити контролер (Controller) та представлення (Views) для кожної операції CRUD.

У [4] дослідник пропонує використовувати наступні типи HTML тегів для вводу даних у залежності від типу даних поля таблиці, таблиця 2

Таблиця 2 — *HTML-теги, які відповідіють до типу поля*

Тип поля	HTML-тег
DATE	<input type=’text’/>
DATETIME	<input type=’text’/>
REAL	<input type=’number’/>
DOUBLE	<input type=’number’/>
INT	<input type=’number’/>
VARCHAR	<input type=’text’/>
TEXT	<textarea></textarea>

1.2 Огляд PHP фреймворку Codeigniter

CodeIgniter 4 - це остання версія популярного фреймворку для розробки веб-додатків на мові програмування PHP. Ось деякі переваги, які пропонує CodeIgniter 4:

Легка встановлюється та налаштовується: CodeIgniter 4 дуже простий у встановленні та налаштуванні. Він поставляється з мінімально необхідними налаштуваннями та завдяки цьому може бути швидко встановлений та запущений.

Швидкодія: CodeIgniter 4 пропонує високу швидкодію завдяки своєму легковагому ядру. Він працює швидше, ніж багато інших PHP-фреймворків.

Простий у використанні: CodeIgniter 4 простий у використанні. Він має чітку документацію та дуже добре структуровану архітектуру, що дозволяє легко орієнтуватися в його функціональності.

Розглянемо структуру коду контроллера на прикляді додатку CRM розробленого на Codeigniter 3

Лістинг 1 *Приклад створеного коду без генерації методу index*

```
function index()
{
     $where = $this->portal['is_master'] ? array() : array('portal_id' =>
$this->portal_id);
    if ($this->is_post()) {
      $categories          =          $this->category_model->get($where,$this-
>calculate_order(),          $this->portal['items_per_page'],          $this-
>calculate_offset());
        $total = $this->category_model->get_count($where);
        $this->calculate_options($total);
        if ($categories) {
            $this->response['result']['html'] = $this->load->view(
                  'category/category_list',
                  array('categories' => $categories),
                  true
            );
        }
        echo json_encode($this->response);
    } else {
        $this->scripts[] = 'js/includes/category__index.js';
        $total = $this->category_model->get_count(array());
        $options = $this->calculate_options($total);
        $json_options = json_encode($options);
        $this->js_init[] = "category__index.init($json_options);";
        $this->display('category/index', $this->data);
    }
}
```

Цей код представляє дію (action) з контролера. Ім'я цієї дії - index(). Ця дія відповідає за відображення списку категорій або фільтрованих категорій на сторінці.

Давайте розберемо функцію крок за кроком:

Функція index() починається зі створення масиву $where, який містить умови фільтрації для запиту до бази даних. В залежності від значення $this->portal['is_master'] фільтр за portal_id може бути включений або виключений. Якщо $this->portal['is_master'] істинне, то умова фільтрації буде порожнім масивом, що дозволить отримати всі категорії. Інакше, умова фільтрації містить лише portal_id, щоб отримати категорії, які належать до певного порталу.

У наступному розділі коду перевіряється, чи надіслано POST-запит на сервер. Це може означати, що користувач вибрав якийсь фільтр або виконав пошук, і список категорій повинен бути оновлений.

Якщо POST-запит був надісланий, функція викликає метод get() моделі category_model, передаючи параметри для умови $where, сортування ($this->calculate_order()), ліміту на кількість елементів та зміщення ($this->portal['items_per_page'], $this->calculate_offset()). В результаті отримується колекція $categories з категоріями, що відповідають умовам. Також знаходиться загальна кількість категорій у $total.

Далі, за допомогою методу $this->calculate_options($total) обчислюються опції для розділу пагінації (скільки сторінок, які показувати, поточна сторінка і т. д.).

Перевіряється, чи є отримані категорії, і якщо так, то вони передаються у шаблон category_list для подальшого відображення списку категорій.

Завершується ця гілка коду за допомогою echo json_encode($this->response);, що дозволяє повернути результати у форматі JSON. Це вказує на те, що код призначений для AJAX-взаємодії, оскільки відповідь віддається у форматі JSON, який часто використовується для асинхронних запитів.

Якщо POST-запит не надійшов або не було знайдено категорії (це можливо, якщо $id не було передано або такої категорії не існує), код переходить до іншої гілки.

Для випадку, коли POST-запит не надійшов, виконується блок else, який відповідає за відображення сторінки списку категорій без фільтрів. В цьому випадку додавані скрипти та налаштування для пагінації ($this->scripts[], $this->js_init[]) і потім сторінка рендериться за допомогою display('category/index', $this->data), де category/index - це шаблон, який відповідає за відображення списку категорій.

У перегляді category/index можна використовувати змінну $category для відображення списку категорій або фільтрованих категорій.

Цей код передбачає використання деякого фреймворка або системи управління контентом (CMS), що дозволяє зробити AJAX-запит на сервер для оновлення списку категорій без перезавантаження сторінки. Також у коді застосовується Eloquent ORM для роботи

Розглянемо структуру коду методу edit на прикладі додатку CRM розробленого на Codeigniter 3

Лістинг 2 *Приклад створеного коду без генерації методу edit*

```
function edit($category_id = null) {
    if ($this->is_post() && !ifset($_POST, 'confirm')) {
        $this->response['message']                                          =
lang('are_you_sure_you_want_to_save_category');
        $this->response['result']['html']           =           $this->load-
>view('/dialog_content/common', $this->response, true);
        echo json_encode($this->response);
    } else if ($this->is_post() && ifset($_POST, 'confirm')) {
        $category = $_POST['category'];
        $category['portal_id']    =    ifset($category,    'portal_id')    ?
$category['portal_id'] : $this->portal_id;
        $updated = $this->category_model->save($category);
        $this->response['message']          =          $updated          ?
lang('action_done_successfully') : lang('some_error_has_occurred');
        $this->response['error'] = !$updated;
        $this->response['result']['redirect_to'] = '/category';
        echo json_encode($this->response);
    } else {
        $this->scripts[] = 'js/includes/category__edit.js';
        $this->js_init[] = "category__edit.init();";
        $this->data['portals'] = $this->portal_model->get(array());
        $this->data['category']           =           $this->category_model-
>get_item_by_id($category_id);
        $this->display('category/edit', $this->data);
    }
}
```

Цей код представляє дію (action) з контролера або модуля веб-додатку. Ім'я цієї дії - edit($category_id). Ця дія відповідає за редагування категорії або збереження змін в базі даних.

Давайте розберемо функцію крок за кроком:

Функція edit($category_id = null) приймає один необов'язковий параметр $category_id, який відповідає ідентифікатору категорії, яку потрібно редагувати. За замовчуванням, $category_id має значення null, що означає, що нова категорія буде створена.

У першому розділі коду перевіряється, чи надісланий POST-запит на сервер. Якщо такий запит надійшов і поле 'confirm' не встановлено в POST-даних, це означає, що користувач намагається зберегти зміни у категорії. Тоді функція формує JSON-відповідь з підтвердженням збереження і відображає діалогове вікно для підтвердження дії.

Якщо POST-запит надісланий і поле 'confirm' встановлено в POST-даних, це означає, що користувач підтвердив збереження категорії. Тоді функція отримує дані форми категорії з $_POST['category'] і змінює або додає portal_id в залежності від наявності цього поля в даних. Потім виконується збереження категорії за допомогою методу save() моделі category_model. Після цього функція повертає JSON-відповідь з результатом збереження і можливим перенаправленням на сторінку списку категорій.

Якщо POST-запит не надійшов або підтвердження збереження не було здійснено, функція виконує гілку else, яка відповідає за відображення сторінки редагування категорії.

В цій гілці додаються скрипти (category__edit.js) та налаштування для ініціалізації редактора (category__edit.init();). Також отримуються дані про портали з моделі portal_model і дані про категорію з моделі category_model з вказаним $category_id. Усі ці дані передаються у шаблон category/edit для відображення на сторінці редагування.

У перегляді category/edit можна використовувати змінні $portals і $category для відображення даних про портали та категорію на сторінці редагування.

Цей код передбачає використання деякого фреймворка або системи управління контентом (CMS), яка дозволяє зробити AJAX-запит на сервер для оновлення списку категорій без перезавантаження сторінки. Також у коді застосовується Eloquent ORM для роботи з базою даних та обробки даних, надісланих користувачем через форму редагування.

Розглянемо структуру коду методу view на прикладі додатку CRM розробленого на Codeigniter 3.

Лістинг 3 *Приклад створеного коду без генерації методу view*

```
function view($category_id = null) {
    if ($category_id) {
        $this->scripts[] = 'js/includes/category__view.js';
        $this->js_init[] = "category__view.init();";
        $this->data['category']            =            $this->category_model-
>get_item_by_id($category_id);
        if    ($this->data['category']    &&    $this->calculate_access($this-
>data['category'])) {
            $this->display('category/view', $this->data);
        } else {
            $this->display('includes/not_found', $this->data);
        }
    }
}
```

Цей код представляє собою функцію з контролера або модуля веб-додатку. Ім'я функції, view(), і вона призначена для перегляду деталей категорії.

Давайте розберемо функцію крок за кроком:

Функція view() має параметр $category_id, який за замовчуванням має значення null. Це означає, що коли функція викликається без передачі аргументу, $category_id буде мати значення null.

Функція перевіряє, чи передано значення $category_id. Якщо так, то вона виконує наступні дії:

a. Додає певні скрипти та ініціалізацію JavaScript до сторінки. Ці скрипти відповідають за деякі функціональність сторінки перегляду категорії.

b. Звертається до моделі category_model для отримання даних про категорію за заданим $category_id. Отримані дані зберігаються у змінній $this->data['category'].

c. Перевіряє, чи існує категорія з таким ID ($this->data['category']) та чи має поточний користувач доступ до перегляду цієї категорії. Функція calculate_access() використовується для перевірки доступу.

d. Якщо категорія і доступ існують, то відображається сторінка з даними категорії (category/view) за допомогою методу display().

e. Якщо категорію не знайдено або у користувача немає доступу до перегляду, відображається сторінка з повідомленням про помилку або "не знайдено" (includes/not_found) за допомогою методу display().

Цей код передбачає асинхронний (Ajax) або стандартний синхронний спосіб виклику функції. Якщо $category_id передається у вигляді аргумента, це означає, що користувач звернувся до сторінки перегляду певної категорії. Інакше, може бути здійснений інший вид поведінки, наприклад, перенаправлення на іншу сторінку або відображення списку усіх категорій.

Отже ми розглянули 3 методи (index, edit, view) у контроллері CRUD. Ці методи є типовими у всіх CRUD контроллерах сутностей.

Загальне у структурі методів CRUD контроллера є те що у кожному методі є виборка сутності/сутностей, для показу, редагування, або видалення.

1.3 Огляд PHP фреймворку Yii

Yii2 - це потужний, високопродуктивний та простий у використанні фреймворк для розробки веб-додатків на мові програмування PHP. Він розроблений з метою прискорення процесу розробки та зменшення кількості коду, необхідного для створення веб-додатків.

Основні переваги Yii2:

Простота використання: Yii2 має просту та логічну структуру, що дозволяє розробникам швидко навчитися використовувати фреймворк та створювати високоякісні веб-додатки.

Швидкодія: Yii2 - це високопродуктивний фреймворк, що дозволяє створювати веб-додатки, які працюють швидко та ефективно, навіть при великому обсязі даних.

Модульність: Yii2 підтримує модульну структуру, що дозволяє розробникам легко організовувати свій код та повторно використовувати його в інших проектах.

Вбудована аутентифікація та авторизація: Yii2 має вбудовану систему аутентифікації та авторизації, що дозволяє розробникам швидко та легко захистити свої веб-додатки від несанкціонованого доступу.

Велика кількість розширень та плагінів: Yii2 має велику кількість розширень та плагінів, що дозволяють розробникам розширювати функціональність своїх веб-додатків.

Підтримка AJAX: Yii2 підтримує AJAX, що дозволяє розробникам створювати динамічні веб-додатки, які не потребують перезавантаження сторінки.

Підтримка RESTful API: Yii2 має вбудовану підтримку RESTful API, що дозволяє розробникам створювати високоякісні та ефективні API.

Підтримка тестування: Yii2 має вбудовану підтримку тестування, що дозволяє розробникам автоматизувати тестування своїх веб-додатків.

Підтримка кешування: Yii2 має вбудовану підтримку кешування, що дозволяє зберігати дані у пам'яті, що підвищує швидкість виконання запитів до бази даних.

Підтримка багатомовності: Yii2 підтримує багатомовність, що дозволяє розробникам легко перекладати свої веб-додатки на інші мови.

Активна спільнота розробників: Yii2 має велику та активну спільноту розробників, що дозволяє отримувати швидку допомогу та підтримку в разі потреби.

Загалом, Yii2 - це потужний та надійний фреймворк для розробки веб-додатків на мові PHP з великою кількістю функціональності та гнучкою архітектурою.

Розглянемо структуру коду методу actionIndex на прикладі додатку CRM розробленого на Yii2

Лістінг 4 *Приклад нагенерованого коду за допомогою Yii2 / Gii метод actionIndex*

```
public function actionIndex()
{
    $searchModel = new CategorySearch();
    $dataProvider = $searchModel->search($this->request->queryParams);
```

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

Цей код представляє дію (action) з контролера або модуля веб-додатку. Ім'я цього методу є actionIndex(), і вона викликається при зверненні до відповідної сторінки (індексної сторінки).

Давайте розберемо функцію крок за кроком:

Функція actionIndex() створює об'єкт $searchModel класу CategorySearch. Припускається, що CategorySearch - це клас, який використовується для пошуку категорій або фільтрації даних.

За допомогою методу search($this->request->queryParams), виконується пошук або фільтрація категорій з використанням параметрів запиту queryParams. $dataProvider містить результати пошуку у вигляді певного провайдера даних, який зазвичай використовується для відображення даних у перегляді (view).

Після виконання пошуку або фільтрації даних, функція повертає результати у вигляді відображення (view) з індексною сторінкою ('index'). Це означає, що дані будуть відображені на веб-сторінці, і відповідний перегляд буде використаний для відображення результатів пошуку або фільтрації.

При відображенні сторінки index, функція передає змінні $searchModel та $dataProvider, які будуть доступні у перегляді (view). Це дозволяє використовувати ці дані для відображення результатів пошуку або фільтрації на сторінці.

Цей код передбачає використання шаблонного рушія або фреймворка, де вираз render('index', ...) використовується для відображення відповідного перегляду з певними змінними. В додатках на основі фреймворка, такий як Yii, Yii2 або Laravel, цей підхід є типовим для організації інтерфейсу користувача та відображення даних на сторінках.

Розглянемо структуру коду методу actionUpdate на прикладі додатку CRM розробленого на Yii2

Лістінг 5 *Приклад нагенерованого коду за допомогою Yii2 / Gii метод actionUpdate*

```
public function actionUpdate($id)
{
    $model = $this->findModel($id);
    if ($this->request->isPost && $model->load($this->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->id]);
    }
    return $this->render('update', [
        'model' => $model,
    ]);
}
```

Цей код представляє дію (action) з контролера або модуля веб-додатку. Ім'я цієї дії, є actionUpdate($id), і вона викликається при зверненні до сторінки редагування певного об'єкта за його ідентифікатором.

Давайте розберемо функцію крок за кроком:

Функція actionUpdate($id) приймає один параметр $id, який відповідає ідентифікатору об'єкта, який потрібно відредагувати.

Функція спочатку викликає метод findModel($id), який шукає модель з вказаним ідентифікатором $id. Метод findModel виконує запит до бази даних або інші дії для знаходження цього об'єкта. Отримана модель зберігається у змінній $model.

Функція перевіряє, чи прийшов POST-запит (запит на збереження змін) та чи модель вдалося завантажити дані з POST-запиту за допомогою методу load($this->request->post()). load() використовується для автоматичного заповнення моделі даними з POST-запиту, якщо такі дані існують. Потім перевіряється, чи вдалося зберегти модель за допомогою методу save(), який зберігає зміни у базі даних.

Якщо збереження пройшло успішно, користувач буде перенаправлений на сторінку перегляду відредагованої моделі, використовуючи метод redirect(['view', 'id' => $model->id]). Тобто, після збереження, користувач буде перенаправлений на сторінку, яка відображає деталі збереженої моделі.

Якщо збереження не було успішним або прийшов GET-запит (перший раз звернулися до сторінки редагування), функція рендерить перегляд (view) для сторінки редагування ('update') та передає туди модель $model для відображення на сторінці.

Цей код передбачає використання шаблонного рушія або фреймворка, де вираз render('update', ...) використовується для відображення відповідного перегляду з певними змінними. В додатках на основі фреймворка, таких як Yii, Yii2 або Laravel, цей підхід є типовим для організації інтерфейсу користувача для редагування даних.

Розглянемо структуру коду методу actionCreate на прикладі додатку CRM розробленого на Yii2

Лістінг 6 *Приклад нагенерованого коду за допомогою Yii2 / Gii метод*

```
actionCreate public function actionCreate()
{
    $model = new Category();
    if ($this->request->isPost) {
        if ($model->load($this->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        }
    } else {
        $model->loadDefaultValues();
    }
    return $this->render('create', [
        'model' => $model,
    ]);
}
```

Цей код представляє дію (action) з контролера або модуля веб-додатку. Ім'я цієї дії, є actionCreate(), і вона викликається при зверненні до сторінки створення нового об'єкта (категорії).

Давайте розберемо функцію крок за кроком:

Функція actionCreate() створює новий екземпляр моделі Category за допомогою коду $model = new Category();. Ця модель представляє сутність "категорія", і вона використовується для створення та збереження нових категорій у базі даних.

Функція перевіряє, чи прийшов POST-запит (запит на створення нової категорії).

Якщо POST-запит прийшов, функція викликає метод load($this->request->post()), щоб спробувати завантажити дані з POST-запиту в модель $model. Після цього перевіряє, чи вдалося зберегти модель, використовуючи метод save(). Якщо збереження успішне, користувач буде перенаправлений на сторінку перегляду нової категорії (['view', 'id' => $model->id]).

Якщо POST-запит не прийшов, функція викликає метод loadDefaultValues(), який заповнює атрибути моделі значеннями за замовчуванням. Це може бути корисно, якщо деякі атрибути мають значення за замовчуванням, які потрібно встановити при створенні нового об'єкта.

В будь-якому випадку (чи прийшов POST-запит або ні), функція рендерить перегляд (view) для сторінки створення нової категорії ('create') та передає туди модель $model для відображення на сторінці створення.

Цей код передбачає використання шаблонного рушія або фреймворка, де вираз render('create', ...) використовується для відображення відповідного перегляду з певними змінними. В додатках на основі фреймворка, таких як Yii, Yii2 або Laravel, цей підхід є типовим для організації інтерфейсу користувача для створення нових записів у базі даних (наприклад, створення нових категорій).

Лістінг 7 *Приклад нагенерованого коду за допомогою Yii2 / Gii метод*

```
actionView public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}
```

Цей код представляє дію (action) з контролера або модуля веб-додатку. Ім'я цієї дії, є actionView($id), і вона викликається при зверненні до сторінки перегляду деталей певного об'єкта за його ідентифікатором.

Давайте розберемо функцію крок за кроком:

Функція actionView($id) приймає один параметр $id, який відповідає ідентифікатору об'єкта, деталі якого потрібно переглянути.

Функція викликає метод findModel($id), який шукає модель з вказаним ідентифікатором $id. Метод findModel виконує запит до бази даних або інші дії для знаходження цього об'єкта. Отримана модель зберігається у змінній $model.

Потім функція рендерить перегляд (view) для сторінки перегляду деталей об'єкта ('view') та передає туди знайдену модель $model для відображення на сторінці.

На сторінці перегляду (view) можна використовувати дані з $model для відображення деталей об'єкта. Таким чином, користувач може переглядати інформацію про об'єкт з вказаним $id.

Цей код передбачає використання шаблонного рушія або фреймворка, де вираз render('view', ...) використовується для відображення відповідного перегляду з певними змінними. В додатках на основі фреймворка, таких як Yii, Yii2 або Laravel, цей підхід є типовим для організації інтерфейсу користувача для перегляду деталей окремих об'єктів у базі даних.

Отже ми розглянули 4 методи (actionIndex, actionCreate, actionUpdate, actionView) у контроллері CRUD. Ці методи є типовими у всіх CRUD контроллерах сутностей у різних фреймвокрах, але є несуттєві відмінності у назві методів, та у способі виборки сутності із бази даних.

1.4 Огляд PHP фреймворку Laravel

Laravel - це потужний відкритий фреймворк для веб-розробки, побудований на мові програмування PHP. Він має простий, але елегантний синтаксис, який дозволяє розробникам швидко та легко створювати високоякісні веб-додатки.

Laravel пропонує широкий спектр функцій, включаючи маршрутизацію, керування базами даних, кешування, сесії, аутентифікацію користувачів, обробку запитів, підтримку електронної пошти, перевірку форм та багато іншого. Ці функції дозволяють розробникам зосередитися на реалізації бізнес-логіки, не турбуючись про рутинні задачі розробки веб-додатків.

Фреймворк Laravel також має високий рівень безпеки та захисту від атак, що забезпечує надійність веб-додатків, побудованих на ньому.

Крім того, Laravel має активну та привітну спільноту розробників, яка постійно допомагає вирішувати проблеми, підтримує фреймворк і розширює його функціональні можливості. Laravel також пропонує широкий спектр документації та підтримки, що дозволяє розробникам швидко і легко знайти відповіді на свої запитання.

Завдяки своїм перевагам, Laravel є одним з найпопулярніших фреймворків для розробки веб-додатків на PHP, з активною спільнотою розробників та безліччю ресурсів для навчання та розробки.

Стиль програмування: Laravel використовує стиль програмування "Model-View-Controller" (MVC), що дозволяє розробникам легко організовувати свій код та розділяти його на логічні компоненти.

Контроль версій бази даних: Laravel має вбудовану підтримку міграцій баз даних, що дозволяє зберігати та контролювати версію баз даних.

Аутентифікація та авторизація: Laravel має вбудовану підтримку аутентифікації та авторизації користувачів, що дозволяє розробникам легко налаштувати систему безпеки своїх додатків.

Роутинг: Laravel має просту та зрозумілу систему роутингу, що дозволяє розробникам легко налаштовувати маршрутизацію запитів у своїх додатках.

Шаблонізація: Laravel має вбудовану підтримку шаблонізації Blade, що дозволяє розробникам легко створювати та організовувати HTML-шаблони своїх додатків.

Пакетний менеджер: Laravel має вбудований пакетний менеджер Composer, що дозволяє розробникам легко додавати та видаляти залежності у своїх проектах.

Підтримка тестування: Laravel має вбудовану підтримку тестування, що дозволяє розробникам автоматизувати тестування своїх веб-додатків.

Підтримка RESTful API: Laravel має вбудовану підтримку RESTful API, що дозволяє розробникам легко створювати та налаштовувати API для своїх додатків.

Розглянемо структуру коду CRUD контроллеру на прикладі додатку CRM розробленого на Laravel

Лістінг 8 *Приклад CRUD контроллера для таблиці categories з бази даних для типової crm метод edit*

```
public function edit(CategoryRequest $request, $id = null) {
	$category = new Category();
	if ($id) {
		$category = Category::where(['id' => $id])->get()->first();
		if (!$category) {
			return abort(404);
		}
	}
	if ($request->isMethod('POST')) {
		$data = $request->all();
		if ($id && $category) {
			$category->update($data);
			return redirect('category');
		} else if (empty($id)) {
			$category = $category->create($data);
			return redirect('category');
		}
	}
	return view('category/edit', compact('category'));
}
```

Цей код представляє функцію-контролер для редагування або створення категорії в базі даних. Припускається, що використовується фреймворк, який підтримує Eloquent ORM (наприклад, Laravel), а також використовується клас CategoryRequest, який, є валідаційним класом для обробки даних, отриманих від користувача.

Давайте розберемо функцію крок за кроком:

Функція edit(CategoryRequest $request, $id = null) приймає два параметри: $request, який представляє запит користувача, і $id, який відповідає ідентифікатору категорії для редагування. Параметр $id має значення за замовчуванням null, тому може бути необов'язковим.

Створюється новий екземпляр моделі Category за допомогою $category = new Category();. Цей екземпляр моделі використовуватиметься для створення нової категорії або зміни існуючої.

Функція перевіряє, чи переданий параметр $id. Якщо $id переданий, то виконується запит до бази даних для пошуку категорії за вказаним ідентифікатором. Якщо категорію не знайдено, функція відповідає з кодом помилки 404.

Якщо метод запиту виконується через HTTP POST (користувач намагається відправити дані форми для збереження змін), функція отримує дані форми за допомогою $request->all().

В залежності від того, чи передано $id, функція виконує оновлення існуючої категорії або створення нової. Якщо $id передано і модель $category не порожня, викликається метод update($data) на екземплярі моделі $category, що дозволяє оновити атрибути категорії з новими даними, переданими з форми. Якщо $id порожній (не передано), створюється новий запис категорії з даними з форми за допомогою методу create($data).

Після зміни або створення категорії, користувач буде перенаправлений на сторінку списку категорій ('category') за допомогою redirect('category').

Якщо метод запиту не POST або ще не виконано створення або зміни категорії, функція відображає перегляд (view) для редагування категорії ('category/edit') і передає змінну $category для відображення даних про категорію на сторінці редагування.

Цей код передбачає використання Laravel або подібного фреймворка з підтримкою Eloquent ORM для зручної роботи з базою даних та обробки запи-

тів користувача. Крім того, він використовує клас CategoryRequest, який, напевно, містить правила валідації для перевірки вхідних даних перед збереженням або оновленням категорії.

Лістінг 9 *Приклад CRUD контроллера для таблиці categories з бази даних для типової crm метод view*

```
public function view($id) {
      $category = Category::where(['id' => $id])->get()->first();
      return view('category/view', compact('category'));
}
```

Цей код виглядає як функція-контролер для відображення деталей категорії на сторінці. Припускається, що використовується фреймворк, який підтримує Eloquent ORM (наприклад, Laravel) для взаємодії з базою даних.

Давайте розберемо функцію крок за кроком:

Функція view($id) приймає один параметр $id, який відповідає ідентифікатору категорії, деталі якої потрібно переглянути.

В першому рядку функції виконується запит до бази даних з використанням Eloquent ORM. Запит шукає категорію за вказаним ідентифікатором $id, використовуючи метод where(['id' => $id])->get(). Метод get() повертає колекцію моделей, які відповідають умові запиту, але для отримання першого результату використовується метод first().

Отримана модель категорії зберігається у змінній $category.

Функція викликає view('category/view', ...) для відображення відповідного перегляду (category/view). Цей перегляд використовується для відображення деталей категорії. Дані про категорію передаються у перегляд за допомогою compact('category'), де змінна $category знаходиться в контексті перегляду.

В перегляді category/view можна використовувати дані про категорію ($category) для відображення її деталей на сторінці.

Цей код передбачає використання Laravel або подібного фреймворка, який має Eloquent ORM для зручної роботи з базою даних. Запит до бази даних

відбувається автоматично завдяки ORM, що забезпечує велику зручність та безпеку взаємодії з даними. Підхід з використанням view() та compact() допомагає передати дані до перегляду для відображення їх у вигляді HTML на сторінці.

Розглянемо структуру коду методу delete на прикладі додатку CRM розробленого на Laravel

Лістінг 10 *Приклад CRUD контроллера для таблиці categories з бази даних для типової crm метод delete*

```
public function delete($id) {
	$category = Category::where(['id' => $id])->get()->first();
	$message = 'Category not found';
	if ($category) {
		$message = 'Error deleting category';
		if ($category->delete()) {
			$message = 'Category deleted';
		}
	}
	return response()->json([
		'message' => $message,
	]);
}
```

Цей код, представляє функцію-контролер для видалення категорії з бази даних. Припускається, що використовується фреймворк, який підтримує Eloquent ORM (наприклад, Laravel) для взаємодії з базою даних.

Давайте розберемо функцію крок за кроком:

Функція delete($id) приймає один параметр $id, який відповідає ідентифікатору категорії, яку потрібно видалити.

В першому рядку функції виконується запит до бази даних з використанням Eloquent ORM, шукаючи категорію за вказаним ідентифікатором $id. Отримана модель категорії зберігається у змінній $category.

Змінна $message ініціалізується рядком 'Category not found', який буде використовуватися, якщо категорія не знайдена.

Функція перевіряє, чи категорія знайдена. Якщо так, то виконується спроба видалення категорії.

Після спроби видалення категорії, змінна $message перепризначається рядком 'Error deleting category', якщо видалення не вдалося.

Якщо видалення відбулося успішно, змінна $message перепризначається рядком 'Category deleted'.

Після всіх перевірок і видалення, функція повертає відповідь у форматі JSON, використовуючи response()->json(...). У відповіді міститься масив з одним елементом 'message', який містить змінну $message. Цей масив містить повідомлення про результат видалення категорії.

Цей код передбачає використання Laravel або подібного фреймворка з підтримкою Eloquent ORM для зручної роботи з базою даних. Запит до бази даних відбувається автоматично завдяки ORM, що забезпечує зручність і безпеку взаємодії з даними. Використання response()->json(...) допомагає повернути результат у форматі JSON, що є зручним для асинхронного (Ajax) взаємодії зі сторінкою.

Лістінг 11 *Приклад CRUD моделі для таблиці categories з бази даних для типової crm*

```
<?php namespace App;
use Illuminate\Database\Eloquent\Model;
use Nayjest\Grids\EloquentDataProvider;
use Nayjest\Grids\FieldConfig;
use Nayjest\Grids\FilterConfig;
use Nayjest\Grids\ObjectDataRow;
```

```
class Category extends Model {
    protected $keyType = 'integer';
    protected $fillable = ['parent_id', 'name', 'created_at', 'updated_at'];

    public function category() {
        return $this->belongsTo('App\Category', 'parent_id');
    }

    public function users() {
        return        $this->belongsToMany('App\User',        'user_category',
'category_id', 'user_id');
    }

    public static function nayjestGridFuzzyName() {
        return (new FieldConfig)
            ->setName('category_id')->setLabel('Categories')
            ->setCallback(function ($val, ObjectDataRow $row) {
                $result = '';
                $model = $row->getSrc();
                $categories = $model->categories;
                foreach ($categories as $category) {
                    $result .=
 "<a                                    href='/category/view/{$category->id}'
target='_blank'>{$category['name']}</a>, ";
                }
                $result = trim($result, ', ');
                return $result;
            })
            ->addFilter(
                (new FilterConfig)
                    ->setName('name')
                    ->setFilteringFunc(function   ($val,   EloquentDataProvider
$dp) {
                        if (!empty($val)) {
                            $items = explode(',', $val);
                            $dp->getBuilder()
                                ->where(function ($query) use ($items) {
                                    foreach ($items as $item) {
                                        if (!empty(trim($item))) {
                                            $item = trim($item);
                                $query->orWhere('categories.name', 'LIKE', "%$item%");
                                        }
                                    }
                                });
                        }
                    }
                )
            );
    }
}
```

Цей код представляє модель Category для фреймворку Laravel, яка використовує пакет Nayjest\Grids для роботи з грідами (таблицями даних) і забезпечення підтримки фільтрів для грідів. Модель відповідає за взаємодію з таблицею "categories" в базі даних та містить деякі зв'язки з іншими моделями.

Давайте розглянемо деякі ключові елементи моделі:

Зв'язки:

belongsTo('App\Category', 'parent_id'): Зв'язок "належить до" (belongsTo) з моделлю Category, де поле "parent_id" вказує на батьківську категорію.

belongsToMany('App\Website', 'website_category', 'category_id', 'website_id'): Зв'язок "належить до багатьох" (belongsToMany) з моделлю Website через проміжну таблицю "website_category" з полями "category_id" та "website_id".

belongsToMany('App\User', 'user_category', 'category_id', 'user_id'): Зв'язок "належить до багатьох" (belongsToMany) з моделлю User через проміжну таблицю "user_category" з полями "category_id" та "user_id".

Метод nayjestGridFuzzyName(): Цей метод використовується для конфігурації гріда (таблиці даних) і фільтрації даних по полю "name". Він додає фільтр для поля "name" за допомогою методу addFilter(), що дозволяє фільтрувати записи за частковим збігом значення.

Властивості моделі:

$table: Назва таблиці, з якою пов'язана модель ("categories").

$primaryKey: Поле, яке виступає в якості первинного ключа таблиці ("id").

$keyType: Тип автоінкрементного ідентифікатора ("integer").

$fillable: Масив із переліком дозволених полів для масового присвоєння, які можуть заповнюватись масивом даних.

Ця модель дозволяє працювати з даними категорій із зазначенням зв'язків між категоріями, веб-сайтами та користувачами. Також, вона надає можливість фільтрувати дані засобами Nayjest\Grids, що полегшує роботу з даними та побудовою таблиць для відображення.

Лістінг 12 *Приклад CRUD представлення для таблиці categories з бази даних для типової crm, представлення редагування*

```
@extends('layouts.app')
@section('content')
    <div class="wrapper" id="dashboard-fullpage"><!--Start First Section-->
        <section class="section section-mobile" id="section1">
            <div class="container">
                <div class="intro-wrapper">
                    <h2 class="intro__headline"></h2>
                    <div class="box-body">
                        <br>
                        @if($category['id'])
```

```
                            <a  href="/category/view/{{  $category['id']  }}"
class="btn btn-success">View</a>
                        @endif
                        <a href="/category" class="btn btn-success">List of the
categories</a>

                        <div class="category-create">
                            <h1>Create/edit Category</h1>
                            @if (count($errors))
                            <div class="alert alert-danger" role="alert">
                                @foreach($errors->all() as $error)
                                    <div>
                                        {{$error}}
                                    </div>
                                @endforeach
                            </div>
                            @endif
                            <div class="category-form">
                                <form  id="w0"  action="/category/edit<?php  if
($category['id']) { ?>/{{$category['id']}}<?php } ?>" method="post">
                                    <input     type="hidden"     name="_token"
id="csrf-token" value="{{ Session::token() }}" />
                                    <div   class="form-group   field-category-
name">
                                        <label           class="control-label"
for="category-name">
Name</label>
                                        <input  type="text"  id="category-name"
class="form-control  @if($errors->has('name'))  has-error  @endif"  name="name"
value="{{ old('name') ? old('name') : $category['name'] }}" maxlength="64">
                                        <div class="help-block">
@if ($errors->has('name')){{ $errors->first('name') }} @endif
</div>
                                    </div>
                                    <div class="form-group">
                                        <button type="submit" class="btn btn-
success">
Save</button>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </section>
        <!--End First Section-->
    </div>
@endsection
```

Це код представлення у шаблоні Blade для сторінки створення/редагування категорії у веб-додатку. Давайте розглянемо його детальніше:

@extends('layouts.app'): Це директива Blade, яка вказує, що цей шаблон розширює (використовує) шаблон "layouts.app". Це означає, що у цьому шаблоні будуть відображатись елементи з "layouts.app", такі як заголовок, меню, тощо.

@section('content'): Це директива Blade, яка визначає секцію з ім'ям "content". Усе, що міститься між @section('content') і @endsection, буде відображено в тілі "content" сторінки.

: Це обгортка сторінки, яка створює блок з ID "dashboard-fullpage" для використання налаштувань віджетів на сторінці.

section і div з атрибутом class="section section-mobile" id="section1": Це блок з ID "section1", який містить контент першого розділу сторінки.

: Це обгортка для елементів вводу та відображення даних сторінки.

<h2 class="intro__headline"></h2>: Це заголовок для розділу.

: Це контейнер, який містить контент форми створення/редагування категорії.

View: Це кнопка для перегляду деталей категорії. Якщо відомий id категорії, то вона створює посилання на сторінку перегляду цієї категорії.

List of the categories: Це кнопка для переходу до списку всіх категорій.

: Це блок для форми створення/редагування категорії.

<h1>Create/edit Category</h1>: Заголовок форми, який залежить від того, чи є id категорії.

@if (count($errors)): Це умова, яка перевіряє, чи є які-небудь помилки валідації форми. Якщо так, вони будуть відображені в блокі alert-danger.

@foreach($errors->all() as $error): Це цикл, який виводить всі помилки валідації форми.

@if($errors->has('name')) has-error @endif: Це умова, яка додає клас "has-error" до елемента форми з помилкою валідації, щоб виділити його стилізацією помилки.

<form id="w0" action="/category/edit<?php if ($category['id']) { ?>/{{$category['id']}}<?php } ?>" method="post">: Це форма для створення/редагування категорії. Дія форми залежить від того, чи відомий id категорії.

@csrf: Це директива Blade, яка генерує токен CSRF для захисту від атак CSRF (Cross-Site Request Forgery).

<input type="text" id="category-name" class="form-control @if($errors->has('name')) has-error @endif" name="name" value="{{ old('name') ? old('name') : $category['name'] }}" maxlength="64">: Це поле вводу для назви категорії. Воно має клас "has-error", якщо валідація для цього поля має помилки. Значення вводу заповнюється з масиву $category, або якщо воно не встановлено, з введених користувачем даних, які були надіслані з форми (old('name')).

<button type="submit" class="btn btn-success">Save</button>: Це кнопка "Зберегти", яка надсилає дані форми на сервер для збереження.

<a href="<?= site_url('categories') ?>" class="btn btn-link">Cancel: Це кнопка "Скасувати", яка веде до списку категорій.

@endsection: Кінцева директива Blade для закриття секції "content".

Лістинг 13 *Приклад CRUD представлення для таблиці categories з бази даних для типової crm, представлення списку*

```
@extends('layouts.app')

@section('content')
  <div class="wrapper" id="dashboard-fullpage">
    <!--Start First Section-->
    <section class="section section-mobile" id="section1">
      <div class="container">
        <div class="intro-wrapper">
          <h2 class="intro__headline"></h2>
          <br>
          <a  href="/category/edit/{{  $category['id']  }}"  class="btn  btn-
success">Edit</a>
          <a   href="/category"   class="btn   btn-success">List   of   the
categories</a>
```

```
          <div class="box-body">
            <div>
              <div id="bc_w0" class="box box-primary">
              <div class="box-header">
                  <h3 class="box-title">
                     <i class="fa fa-eys"></i> {{$category['name']}}
                  </h3>
                </div>
                <div class="box-body">
                  <table  id="w1"  class="table  table-striped  table-bordered
detail-view">
                    <tbody>
                      <tr>
                        <th>ID</th><td>{{ $category['id'] }}</td>
                      </tr>
                      <tr>
                        <th>Name</th><td>{{ $category['name'] }}</td>
                      </tr>
                      <tr>
                        <th>Users</th>
                        <td>
                          @if ($category['users'])
                            @foreach($category['users'] as $user)
                              <a             href="/user/view/{{$user['id']}}"
target="_blank">
                                <?php echo App\User::displayName($user); ?>
                              </a>, <br>
                            @endforeach
                          @endif
                        </td>
                      </tr>
                      <tr>
                        <th>Created at</th>
                        <td>{{ $category['created_at'] }}</td>
                      </tr>
@endsection
```

Цей код є шаблоном сторінки "view" для відображення деталей категорії на веб-додатку на фреймворку Laravel. Шаблон використовується для виведення даних про категорію та пов'язані з нею дані (користувачі, веб-сайти). Структура сторінки включає заголовок, додаткові посилання для редагування та перегляду списку категорій.

Давайте розглянемо деякі ключові елементи шаблону:

@extends('layouts.app') та @section('content'): Шаблон успадковує вміст з іншого шаблону "layouts.app" і вставляє вміст сторінки в секцію @yield('content') шаблону "layouts.app".

: Контейнер для сторінки.

<section class="section section-mobile" id="section1">: Секція для першого розділу сторінки з класами "section" та "section-mobile".

: Контейнер для контенту першого розділу сторінки.

<h2 class="intro__headline"></h2>: Заголовок розділу - можливо, вміст для заголовка вставляється динамічно.

Edit та List of the categories: Посилання для редагування категорії та перегляду списку категорій.

: Контейнер для виведення деталей категорії.

<table id="w1" class="table table-striped table-bordered detail-view">: Таблиця для виведення деталей категорії.

Кожен рядок таблиці має назву та значення деталей категорії, такі як ID, Name, Users, Websites, Created at та Updated at. Деякі значення виводяться динамічно змінними, які передаються з контролера.

В цілому, цей шаблон дозволяє створювати сторінки для перегляду деталей категорій із можливістю редагування та перегляду списку категорій та пов'язаних з ними даних (користувачі, веб-сайти). Контент сторінки може бути встановлений динамічно залежно від даних, які передаються з контролера.

Лістинг 14 *Приклад представлення лейаута*

```
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="csrf-token" content="{{ csrf_token() }}">
  <title>{{ config('app.name', 'Laravel') }}</title>
  <script src="{{ asset('js/app.js') }}"></script>
  <link rel="dns-prefetch" href="//fonts.gstatic.com">
  <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
<body>
  <div id="app">
    <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
      <div class="container">
        <a class="navbar-brand" href="{{ url('/') }}">
          {{ config('app.name', 'Laravel') }}
```

```
          <img src="/logo.png" width="150" height="37"/>
        </a>
        <button  class="navbar-toggler"  type="button"  data-toggle="collapse"
data-target="#navbarSupportedContent"   aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="{{ __('Toggle navigation') }}">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
          <ul class="navbar-nav mr-auto"></ul>
          <ul class="navbar-nav ml-auto">
            <!-- Authentication Links -->
            @guest
              <li class="nav-item">
                <a class="nav-link" href="{{ route('login') }}">
                  {{ __('Login') }}
                </a>
              </li>
              @if (Route::has('register'))
                <li class="nav-item">
                  <a class="nav-link" href="{{ route('register') }}">
                    {{ __('Register') }}
                  </a>
                </li>
              @endif
            @else
              <li class="nav-item dropdown">
                <a id="navbarDropdown" href="#" role="button">
                  {{ Auth::user()->name }}<span class="caret"></span>
                </a>
                <div class="dropdown-menu dropdown-menu-right"
                  aria-labelledby="navbarDropdown">
                  <a class="dropdown-item" href="{{ route('home') }}">
                    {{ __('Home') }}
                  </a>
                  <a class="dropdown-item" href="{{ route('category') }}">
                    {{ __('View Categories') }}
                  </a>
                  <a class="dropdown-item" href="{{ route('tag') }}">
                    {{ __('View Tags') }}
                  </a>
                  <a class="dropdown-item"
                     document.getElementById('logout-form').submit();">
                    {{ __('Logout') }}
                  </a>
                  <form id="logout-form" action="{{ route('logout') }}"
                    method="POST" style="display: none;">
                    @csrf
                  </form>
                </div>
              </li>
            @endguest
          </ul>
        </div>
      </div>
    </nav>
    <main class="py-4">
      @yield('content')
    </main>
  </div>
  @stack('scripts')
</body>
</html>
```

Цей код є шаблоном для головної сторінки (layout) веб-додатку на фреймворку Laravel. Він визначає заголовок сторінки, підключає основні скрипти та стилі, налаштовує навігаційне меню та відображає контент, який передається з дочірніх сторінок (view) через секцію @yield('content').

Давайте розглянемо деякі ключові елементи шаблону:

<!doctype html>: Оголошення типу документа - HTML5.

<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">: Вказання мови сторінки, що використовується для ідентифікації локалізації.

<head>: Заголовок документа, де підключаються скрипти, стилі та мета-теги.

<meta name="csrf-token" content="{{ csrf_token() }}">: Мета-тег для збереження CSRF-токену, що використовується для захисту від атак на міжсайтову подіб'ємність (CSRF).

<script src="{{ asset('js/app.js') }}"></script>: Підключення скрипту app.js, який містить JavaScript код додатку.

<link href="{{ asset('css/app.css') }}" rel="stylesheet">: Підключення стилів з файлу app.css.

: Головний контейнер додатку, куди будуть вбудовуватись дочірні сторінки.

<nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">: Навігаційне меню.

@yield('content'): Секція, куди будуть вставлятись вміст дочірніх сторінок.

@stack('scripts'): Секція, куди можна додавати скрипти, які будуть включені перед закриваючим тегом </body>.

Секція навігаційного меню включає різні посилання в залежності від статусу користувача (зареєстрований або не зареєстрований) та його ролі (наприклад, "Home", "View Categories", "View Tags" тощо).

В цілому, цей шаблон забезпечує структуру для збудованого веб-додатку на платформі Laravel, включаючи стилізацію та навігаційне меню. Контент сторінок буде вставлятись у визначену секцію, що дозволяє легко створювати сторінки з власним вмістом та відображенням.

На рисунках 1,2,3,4 зображено представлення редагування та перегляду сутності та перегляд списку сутностей категорії у фреймворці Laravel.

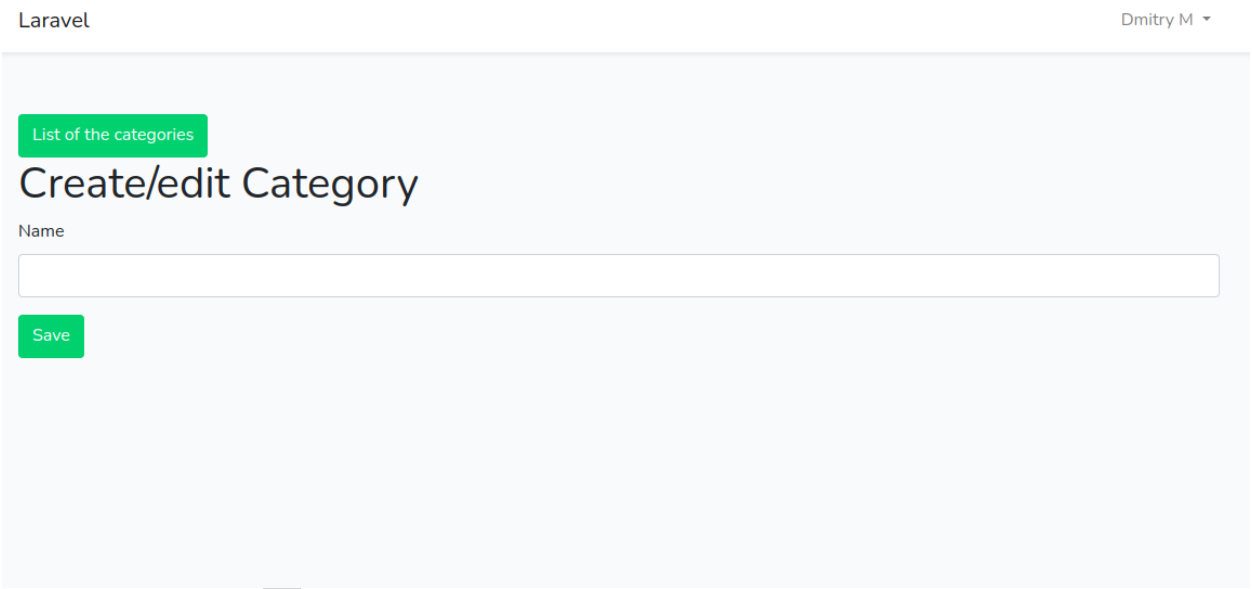

Рисунок 1 — *Вигляд представлення створення категорії*

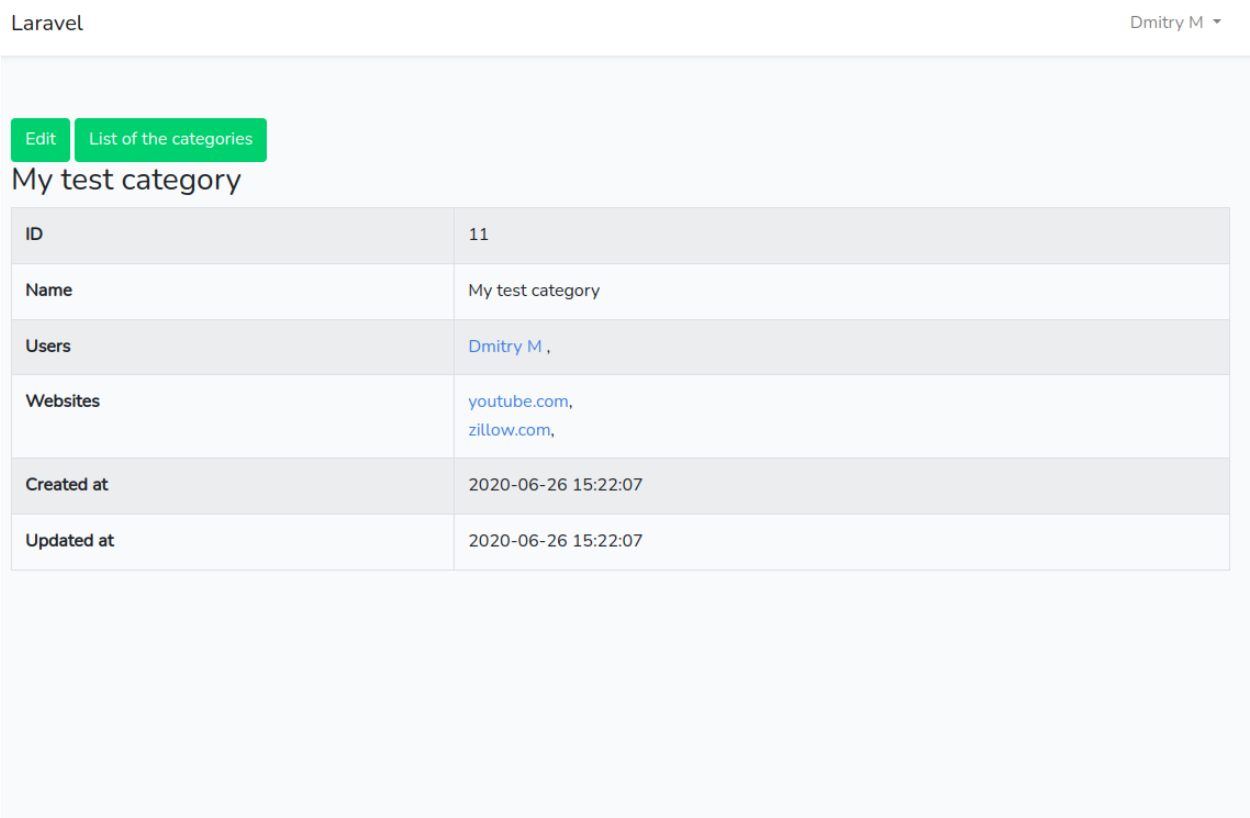

Рисунок 2 — *Вигляд представлення перегляду категорії*

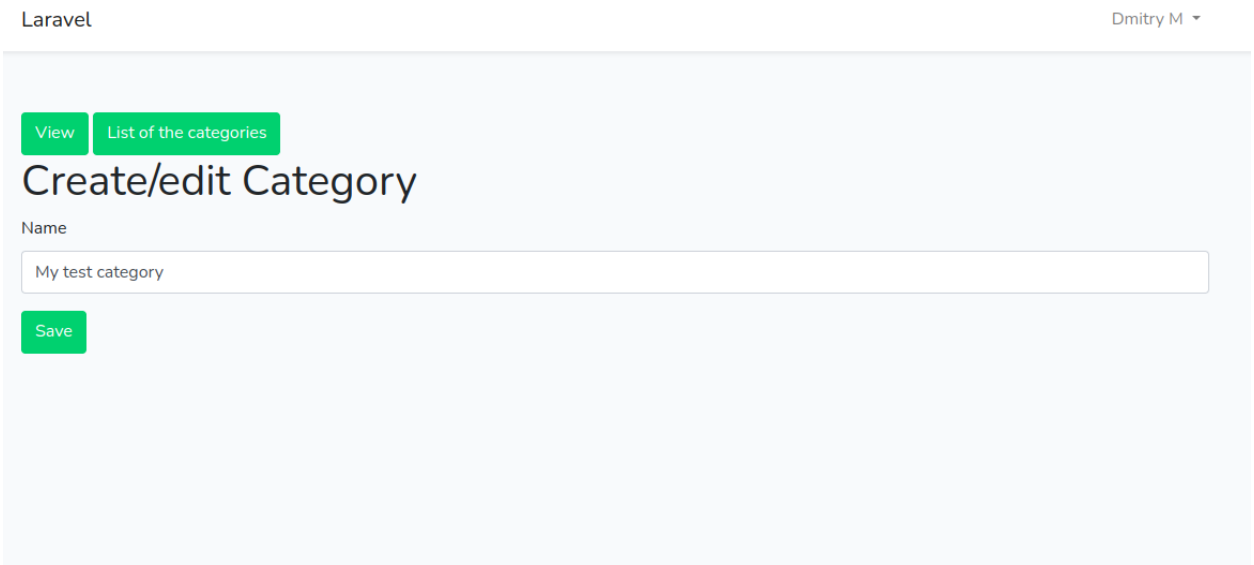

Рисунок 3 — *Вигляд представлення редагування категорії*

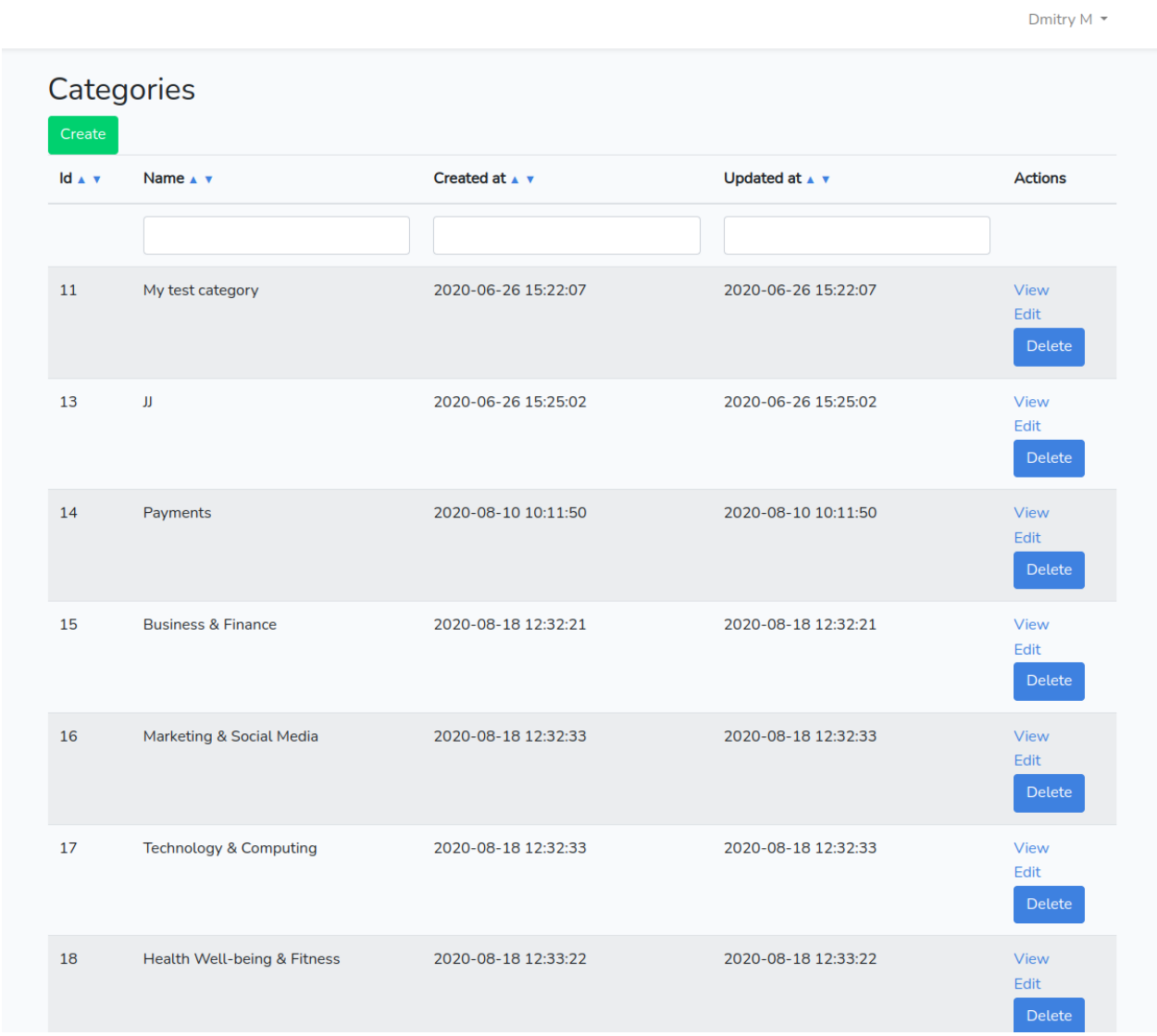

Рисунок 4 — *Вигляд представлення перегляду категорій*

1.5 Огляд PHP фреймворку Zend2

Переваги Zend Framework 2:

Масштабованість: ZF2 дозволяє побудувати великі, масштабовані додатки з використанням модульної архітектури.

Гнучкість: фреймворк надає велику кількість компонентів та бібліотек, які можна використовувати для розширення та налаштування функціональності.

Міцна безпека: ZF2 має вбудовані інструменти безпеки, такі як фільтрація та валідація даних, захист від XSS-атак, контроль доступу і т.д.

Підтримка стандартів: фреймворк дотримується PSR-стандартів, таких як PSR-3 (логування), PSR-4 (автозавантаження класів) та інших.

Недоліки Zend Framework 2:

Складність: фреймворк має велику кількість компонентів та може вимагати додаткового часу для оволодіння.

Документація: в порівнянні з іншими фреймворками, документація для Zend Framework 2 може бути не такою широкою та оновленою.

У Zend Framework 2 (ZF2) реалізація CRUD (Create, Read, Update, Delete) здійснюється за допомогою паттерну проектування MVC (Model-View-Controller). Нижче розглянуто, як працює CRUD MVC у Zend Framework 2:

Модель (Model):

Модель представляє доступ до даних і бізнес-логіку вашого додатка.
У ZF2 ви можете створити окремий клас моделі, який відповідає за взаємодію з базою даних або іншим джерелом даних.
Модель містить методи для створення, отримання, оновлення та видалення даних (CRUD операції).

Представлення (View):

Представлення відповідає за відображення даних користувачеві.

У ZF2 представлення можуть бути реалізовані за допомогою шаблонів (наприклад, використовуючи Zend\View).

Представлення отримують дані від контролера та відображають їх у відповідному форматі (HTML, JSON тощо).

Контролер (Controller):

Контролер обробляє HTTP-запити користувача та взаємодіє з моделлю та представленням.

У ZF2 контролери відповідають за обробку маршрутів та визначення дій, що пов'язані з CRUD операціями.

Контролер отримує дані від HTTP-запиту, викликає відповідні методи моделі для виконання CRUD операцій, а потім передає отримані дані в представлення для відображення.

Маршрутизація:

У ZF2 ви можете налаштувати маршрутизацію для визначення шляхів (URL) та відповідних контролерів та дій для обробки HTTP-запитів.
Маршрутизація визначає, який контролер та дія повинні бути викликані для кожного HTTP-запиту.

Усі ці компоненти разом дозволяють реалізувати CRUD операції в Zend Framework 2. Наприклад, для створення запису, контролер отримує дані з HTTP-запиту, передає їх моделі для створення запису у базі даних, а потім відправляє відповідь в представлення для відображення результату.

Важливо відзначити, що дані CRUD оперують на рівні моделі, а представлення і контролери відповідають за обробку та відображення цих даних. Залежно від реалізації та потреб вашого проекту, можуть використовуватись додаткові компоненти та бібліотеки, наприклад, ORM (Object-Relational Mapping) для спрощення взаємодії з базою даних.

1.6 Огляд PHP фреймворку CakePHP

CakePHP є популярним фреймворком PHP, і він має свої переваги та недоліки. Ось деякі з них:

Переваги CakePHP:

Швидкість розробки: CakePHP пропонує конвенцію над конфігурацією, що дозволяє швидко створювати додатки. Він має готовий шаблон проекту, автозавантаження класів та зрозумілу структуру каталогів.
Компонентна архітектура: Фреймворк розбитий на компоненти, що дозволяє використовувати тільки ті функціональні можливості, які вам потрібні, і додає гнучкість до проекту.

Вбудовані функції безпеки: CakePHP надає різні механізми для захисту додатків, такі як попередження від SQL-ін'єкцій, хешування паролів і контроль доступу.

Готові компоненти: Фреймворк поставляється з великою кількістю готових компонентів, таких як аутентифікація, авторизація, робота з базою даних, кешування і багато іншого.

Активна спільнота: CakePHP має велику та активну спільноту, що забезпечує підтримку, документацію, розширення та плагіни для фреймворку.

Недоліки CakePHP:

Навчання: Хоча CakePHP має швидку швидкість розробки, навчання фреймворку може бути вимогливим. Потрібно зрозуміти його конвенції та структуру, щоб ефективно використовувати його можливості.

Високі вимоги до сервера: Деякі функції CakePHP можуть вимагати більше ресурсів сервера, що може впливати на продуктивність, особливо для великих проектів.

Обмежена гнучкість: Хоча CakePHP пропонує швидкість розробки завдяки конвенціям, це також може обмежувати гнучкість в налаштуванні деяких аспектів додатку. Ви можете зіткнутися з обмеженнями, якщо ваші потреби відхиляються від стандартних конвенцій фреймворку.

Кожен фреймворк має свої переваги та недоліки, і вибір фреймворку залежить від потреб вашого проекту та вашого досвіду розробки.

У CakePHP, реалізація CRUD (Create, Read, Update, Delete) відбувається автоматично за допомогою паттерну проектування MVC (Model-View-Controller) та функцій, що надаються фреймворком. Нижче розглянуто, як працює CRUD MVC у CakePHP:

Модель (Model):

Модель в CakePHP представляє доступ до даних та бізнес-логіку вашого додатка.

У випадку CRUD операцій, CakePHP автоматично генерує модель на основі схеми бази даних. Це дозволяє легко працювати з таблицями та записами в базі даних.

Модель містить методи для створення, отримання, оновлення та видалення даних (CRUD операції).

Представлення (View):

Представлення відповідає за відображення даних користувачеві.

У CakePHP представлення можуть бути створені за допомогою шаблонів (наприклад, використовуючи .ctp-файли).

Представлення отримують дані від контролера та відображають їх у відповідному форматі (HTML, JSON тощо).

Контролер (Controller):

Контролер обробляє HTTP-запити користувача та взаємодіє з моделлю та представленням.

У CakePHP контролери відповідають за обробку маршрутів та визначення дій, що пов'язані з CRUD операціями.

Контролер отримує дані від HTTP-запиту, викликає відповідні методи моделі для виконання CRUD операцій, а потім передає отримані дані в представлення для відображення.

Маршрутизація:

У CakePHP ви можете налаштувати маршрутизацію для визначення шляхів (URL) та відповідних контролерів та дій для обробки HTTP-запитів.

Маршрутизація визначає, який контролер та дія повинні бути викликані для кожного HTTP-запиту.

Усі ці компоненти разом дозволяють легко реалізувати CRUD операції в CakePHP. Фреймворк автоматично генерує базовий код для створення, отримання, оновлення та видалення даних на основі моделі та схеми бази даних. Ви також можете настроїти та налаштувати більш специфічні CRUD операції, використовуючи вбудовані методи та функціональні можливості CakePHP.

1.7 Порівняння php фреймворків

Отже проаналізувавши CRUD контроллери сутностей у різних фреймворках є декілька відмінностей у назвах (таблиця 1) та способах реалізації вибоpки сутності.

Таблиця 3 — *Порівняння назв методів у CRUD контроллерах різних php фреймворків*

Фреймворк	Codeigniter 3,4	Laravel 7	Yii2	Zend Framework 2
Метод списку сутностей	index()	index()	actionIndex()	indexAction()
Метод додавання сутності	edit($id=null)	edit(ItemRequest $request, $id = null)	actionCreate()	addAction()
Метод редагування сутності			actionUpdate($id)	
Метод перегляду сутності	view($id=null)	view($id)	actionView($id)	viewAction($id)
Метод видалення сутності	delete()	delete($id)	actionDelete($id)	deleteAction()
Метод вибірки сутності по первинному полю			findModel($id)	getItem()

1.8 Висновки з розділу 1

CRUD (Create, Read, Update, Delete) контролери в різних PHP фреймворках зазвичай мають спільний функціонал і структуру, оскільки CRUD-операції є основними операціями, які виконуються з базами даних.

Ось кілька спільних рис із CRUD контролерів різних PHP фреймворків:

Маршрутизація: Усі CRUD контролери визначають шляхи (routes), за допомогою яких здійснюється доступ до різних операцій CRUD. Наприклад,

шлях "/users" може бути використаний для виклику операцій CRUD, пов'язаних з користувачами.

Операції CRUD: Кожен CRUD контролер має методи для виконання операцій CRUD. Зазвичай це методи з назвами, що відповідають операціям: create, read, update, delete. Наприклад, метод "create" використовується для створення нового запису, "read" для отримання інформації про запис, "update" для оновлення запису і "delete" для видалення запису.

Робота з моделями: CRUD контролери зазвичай взаємодіють з моделями, які представляють дані у базі даних. Вони можуть використовувати методи моделей для створення, зчитування, оновлення та видалення записів.

Валідація даних: CRUD контролери часто включають в себе логіку валідації даних, щоб переконатися, що передані дані коректні перед виконанням операцій CRUD. Наприклад, вони можуть перевіряти обов'язкові поля, формати даних, унікальність значень тощо.

Обробка помилок: Контролери зазвичай містять обробку помилок, якщо операції CRUD не вдаються або стаються інші помилки. Це може включати повідомлення про помилку, відправку відповіді з помилковим статусом.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ІСНУЮЧИХ БІБЛІОТЕК ГЕНЕРАЦІЇ CRUD КОНТРОЛЛЕРІВ

2.1 Дослідження бібліотеки PHP Yii2 / Gii

Gii (або "Generator II") - це генератор коду для фреймворку веб-розробки Yii. Gii дозволяє розробникам швидко та легко створювати шаблони коду для створення моделей, контролерів та переглядів для веб-додатків.

Gii має простий та інтуїтивний інтерфейс користувача, що дозволяє вибирати налаштування генерації коду з використанням різноманітних параметрів. Крім того, Gii дозволяє зберігати налаштування генерації коду як шаблони, що можуть бути використані в майбутньому для швидкого створення нових елементів додатку.

Gii також має безліч готових шаблонів коду для створення різних елементів додатку, таких як моделі, контролери та представлення. Розробники можуть вибирати з багатьох доступних шаблонів та налаштовувати їх, щоб вони відповідали їх потребам.

Gii може бути використаний як самостійний інструмент, або вбудований в інші інструменти Yii. Наприклад, Gii може бути використаний в консольній програмі, яка генерує код автоматично на основі бази даних.

Завдяки своїм можливостям Gii є потужним інструментом для розробників Yii, що дозволяє їм ефективно створювати високоякісний код для веб-додатків шляхом автоматизації рутинної роботи зі створенням нових елементів додатку.

Gii, генератор коду для фреймворка Yii, використовує різноманітні патерни програмування для створення якісного та масштабованого коду.

Деякі з патернів програмування, які використовуються в Gii, включають:

- Factory Method - для створення різних типів об'єктів в залежності від обраного шаблону генерації коду.
- Singleton - для забезпечення того, що лише один екземпляр Gii створюється та використовується в програмі.
- Observer - для реалізації подій та сповіщень при створенні різних компонентів коду.
- Template Method - для визначення загальної структури генерації коду, але з деякими варіаціями, щоб різні елементи коду могли бути згенеровані з різними параметрами.
- Strategy - для реалізації різних алгоритмів генерації коду в залежності від обраного шаблону.

Ці патерни програмування допомагають забезпечити гнучкість та масштабованість коду, який створюється за допомогою Gii, дозволяючи розробникам легко додавати нові шаблони та компоненти до генерованого коду.

Gii, як і більшість генераторів коду, використовує метадані про структуру бази даних для генерації CRUD-операцій. Конкретно, для отримання списку полів у таблиці, Gii використовує інформацію, яка міститься у схемі бази даних, яку вибирає користувач при налаштуванні генератора.

Наприклад, якщо використовується MySQL база даних, Gii може отримати список полів у таблиці за допомогою запиту до інформаційної схеми бази даних:

Лістинг 15 *SQL запит для вибору списку полей у таблиці*

```
SELECT COLUMN_NAME
```

```
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'database_name' AND TABLE_NAME = 'table_name';
```

Таким чином, Gii може автоматично генерувати код CRUD-операцій для таблиці, використовуючи інформацію про структуру таблиці та зберігаючи її відповідно до встановлених користувачем налаштувань.

Gii може автоматично виявляти зовнішні ключі (foreign keys) в таблиці та використовувати їх для визначення зв'язків між таблицями та генерації коду CRUD-операцій. Коли вибрана таблиця для генерації, Gii перевіряє її схему бази даних та використовує запит до інформаційної схеми бази даних для отримання списку зовнішніх ключів (foreign keys), пов'язаних з вибраною таблицею.

Наприклад, якщо ви використовуєте MySQL базу даних, Gii може отримати список зовнішніх ключів у таблиці за допомогою запиту до інформаційної схеми бази даних:

Лістинг 16 *SQL запит для вибору списку вторинних полей у таблиці*

```
SELECT COLUMN_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = 'database_name'
    AND TABLE_NAME = 'table_name'
    AND REFERENCED_TABLE_NAME IS NOT NULL;
```

Після того, як Gii отримує список зовнішніх ключів, він використовує їх, щоб визначити, які таблиці пов'язані з вибраною таблицею та створити посилання на відповідні CRUD-операції у згенерованому коді. Якщо зовнішні таблиці мають власні CRUD-операції, Gii може також згенерувати посилання на ці операції.

2.2 Дослііження бібліотеки PHP Laravel / Nayjest Grid

Nayjest/Grids - це PHP-бібліотека для створення таблиць з даними в браузері. Вона дозволяє відобразити дані з бази даних у вигляді таблиці та надає зручні інструменти для сортування, фільтрації, пошуку та пагінації цих даних.

Основні функції та можливості Nayjest/Grids:

- сортування даних в таблиці за допомогою кліків по заголовках стовпців таблиці.
- фільтрація даних в таблиці за допомогою спеціальних елементів керування.
- пошук даних в таблиці за допомогою текстового поля.
- пагінація даних в таблиці для поділу великої кількості даних на менші частини.
- можливість додавати до таблиці додаткові елементи керування, такі як кнопки та покликання.
- підтримка різних типів даних, включаючи текст, числа, дати, зображення та інші.
- підтримка декількох рівнів вкладеності для створення складних таблиць зі зв'язаними даними.
- можливість додавати власні колонки та елементи керування до таблиці.
- легкий у використанні та налаштовуванні. Має бути перелік ненумерований

Загалом, Nayjest/Grids дозволяє легко та швидко створювати таблиці з даними у веб-додатках з високою функціональністю та можливостями для користувача. Вона підтримується активною спільнотою розробників та постійно оновлюється з новими можливостями та функціями.

Nayjest/Grids використовує наступні паттерни проектування:

Builder - цей паттерн використовується для створення складних об'єктів крок за кроком. У Nayjest/Grids він дозволяє визначити конфігурацію таблиці крок за кроком, встановлюючи параметри, такі як заголовок стовпців, типи даних, фільтри, сортування, пагінацію та інші.

Decorator - цей паттерн дозволяє додавати нові функціональність до існуючих об'єктів, не змінюючи їхньої структури. У Nayjest/Grids він використовується для додавання нових елементів керування до таблиці, таких як кнопки та посилання.

Strategy - цей паттерн використовується для визначення різних алгоритмів, які можуть бути застосовані до об'єктів, не змінюючи їхньої структури. У Nayjest/Grids він використовується для визначення різних стратегій сортування, фільтрації та пагінації таблиці.

Composite - цей паттерн використовується для об'єднання пов'язаних об'єктів у єдину структуру, що дозволяє їм працювати як єдиний об'єкт. У Nayjest/Grids він використовується для відображення складних таблиць зі зв'язаними даними, де кожен елемент таблиці може мати свої власні підпорядковані елементи.

2.3 Структура бази даних додатку

Розглянемо структуру бази даних (рисунок 5) на прикладі розробленого додатку CRM.

Список таблиць та колонок:

categories - id, name, parent_id, created_at, updated_at

companies - id, name, created_at, updated_at

countries - id, code, name

languages - id, name, iso_639-1

prices - id, name, words_number

roles - id, name

tags - id, name, created_at, updated_at

user_category - user_id, category_id

user_company - user_id, company_id

user_price - user_id, price_id, price_value,id

user_role - user_id, role_id

user_website - user_id, website_id

users - id, name, firstname, lastname, website_address, phone, email, email_verified_at, password, remember_token, detailed_requirements,

minimum_da, notes, opportunities, examples_of_work, expertise, rating, hubspot_id, hubspot_vid, hubspot_profile_url, created_at, updated_at, is_trusted

website_category - website_id, category_id

website_tag - website_id, tag_id

websites - id, da, website, email, further_notes, site_url, cpl, cpl_currency, cpp, cpp_currency, do_follow, link_policy, is_trusted, source, added_by_user, country_id, language_id, created_at, updated_at, links_allowed

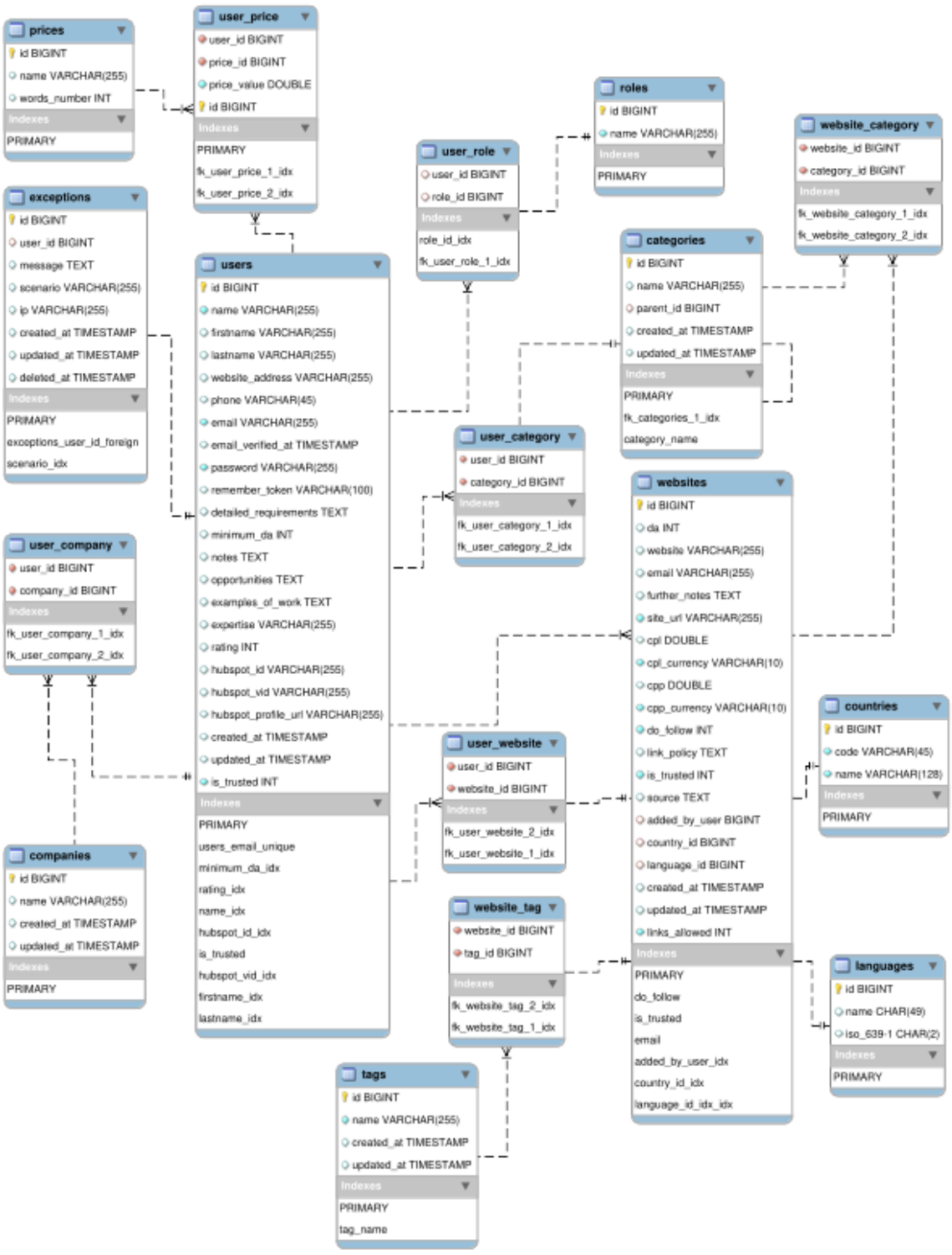
prices
id BIGINT
name VARCHAR(255)
words_number INT
Indexes
PRIMARY
user_price
user_id BIGINT
price_id BIGINT
price_value DOUBLE
id BIGINT
Indexes
PRIMARY
fk_user_price_1_idx
fk_user_price_2_idx
roles
id BIGINT
name VARCHAR(255)
Indexes
PRIMARY
user_role
user_id BIGINT
role_id BIGINT
Indexes
role_id_idx
fk_user_role_1_idx
website_category
website_id BIGINT
category_id BIGINT
Indexes
fk_website_category_1_idx
fk_website_category_2_idx
exceptions
id BIGINT
user_id BIGINT
message TEXT
scenario VARCHAR(255)
ip VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
deleted_at TIMESTAMP
Indexes
PRIMARY
exceptions_user_id_foreign
scenario_idx
users
id BIGINT
name VARCHAR(255)
firstname VARCHAR(255)
lastname VARCHAR(255)
website_address VARCHAR(255)
phone VARCHAR(45)
email VARCHAR(255)
email_verified_at TIMESTAMP
password VARCHAR(255)
remember_token VARCHAR(100)
detailed_requirements TEXT
minimum_da INT
notes TEXT
opportunities TEXT
examples_of_work TEXT
expertise VARCHAR(255)
rating INT
hubspot_id VARCHAR(255)
hubspot_vid VARCHAR(255)
hubspot_profile_url VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
is_trusted INT
Indexes
PRIMARY
users_email_unique
minimum_da_idx
rating_idx
name_idx
hubspot_id_idx
is_trusted
hubspot_vid_idx
firstname_idx
lastname_idx
categories
id BIGINT
name VARCHAR(255)
parent_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
PRIMARY
fk_categories_1_idx
category_name
user_category
user_id BIGINT
category_id BIGINT
Indexes
fk_user_category_1_idx
fk_user_category_2_idx
websites
id BIGINT
da INT
website VARCHAR(255)
email VARCHAR(255)
further_notes TEXT
site_url VARCHAR(255)
cpl DOUBLE
cpl_currency VARCHAR(10)
cpp DOUBLE
cpp_currency VARCHAR(10)
do_follow INT
link_policy TEXT
is_trusted INT
source TEXT
added_by_user BIGINT
country_id BIGINT
language_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
links_allowed INT
Indexes
PRIMARY
do_follow
is_trusted
email
added_by_user_idx
country_id_idx
language_id_idx_idx
user_company
user_id BIGINT
company_id BIGINT
Indexes
fk_user_company_1_idx
fk_user_company_2_idx
user_website
user_id BIGINT
website_id BIGINT
Indexes
fk_user_website_2_idx
fk_user_website_1_idx
countries
id BIGINT
code VARCHAR(45)
name VARCHAR(128)
Indexes
PRIMARY
companies
id BIGINT
name VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
PRIMARY
website_tag
website_id BIGINT
tag_id BIGINT
Indexes
fk_website_tag_2_idx
fk_website_tag_1_idx
languages
id BIGINT
name CHAR(49)
iso_639-1 CHAR(2)
Indexes
PRIMARY
tags
id BIGINT
name VARCHAR(255)
created_at TIMESTAMP
updated_at TIMESTAMP
Indexes
PRIMARY
tag_name

Рисунок 5 —*UML діаграмма бази даних додатку CRM*

2.4 Висновки з розділу 2

Gii yii2 та Laravel Nayjest/Grids - це два різних інструменти, які мають свої відмінні особливості та призначення. Gii забезпечує швидкий та простий спосіб створення базової структури проекту та генерувати код для багатьох

елементів, тоді як Nayjest/Grids дозволяє створювати таблиці з розширеними функціональними можливостями. В залежності від потреб проекту, можна вибрати той інструмент, який найкраще відповідає вимогам розробки.

Yii2 Gii Generator може значно пришвидшити розробку додатків в Yii2-фреймворку. Він автоматизує багато рутинних завдань, що пов'язані з створенням моделей, контролерів, представлень та інших елементів додатку.

Gii генерує основний код, що допомагає уникнути рутинної роботи, такої як написання коду CRUD-операцій, налаштування правил валідації, створення міграційних файлів і т.і. Це дозволяє розробникам зосередитися на складніших задачах, таких як дизайн та бізнес-логіка додатку.

Крім того, Gii може генерувати код на основі існуючих таблиць баз даних, що також може дуже значно прискорити процес розробки.

Таким чином, використання Yii2 Gii Generator може дуже сильно пришвидшити розробку додатків в Yii2-фреймворку і зменшити кількість рутинних завдань, що пов'язані зі створенням основних елементів додатку.

РОЗДІЛ 3 РОЗРОБКА CRUD ГЕНЕРАТОРА ДО ФРЕЙМВОРКУ CODEIGNITER4

3.1 Розробка шаблону контролера

Процес розробки шаблону типового CRUD контролера для генерації контролера на основі таблиці з бази даних може включати наступні кроки:

Аналіз таблиці: Спочатку потрібно провести аналіз таблиці в базі даних, з якої потрібно згенерувати контролер. Проаналізувати структуру таблиці, поля, зв'язки, обмеження і будь-які особливості даних, котрі ви плануєте використовувати в контролері.

Визначення операцій CRUD: CRUD відноситься до чотирьох основних операцій, які можна виконувати з базою даних - Create (створення), Read (читання), Update (оновлення) і Delete (видалення).

Створення шаблону контролера: На основі вищезазначеного аналізу створимо шаблон контролера, який міститиме методи для кожної операції CRUD. Наприклад, методи create(), read(), update() і delete(), які відповідатимуть відповідним операціям.

Реалізація методів CRUD: Для кожної операції CRUD реалізуємо відповідний метод у контролері. Наприклад, метод create() може отримувати дані від користувача, перевіряти їх, створювати новий запис у таблиці та повертати результат.

Обробка запитів і відповідей: Контролер повинен обробляти запити від користувача і надавати відповіді. Це може включати обробку параметрів запиту, валідацію даних, виконання відповідних операцій з базою даних і формування відповідей у вигляді JSON, HTML або інших форматів даних.

Тестування і налагодження: Після реалізації контролера проведемо тестування для перевірки його роботи. Переконаємося, що всі операції CRUD працюють належним чином, а також врахуємо можливі помилки і винятки, які можуть виникнути при взаємодії з базою даних.

Інтеграція з іншими компонентами: Контролер може бути частиною більшої архітектури програмного забезпечення. Його можна інтегрувати з іншими компонентами, такими як модель (доступ до даних), представлення (відображення даних) та маршрутизатор (направлення запитів до відповідних методів контролера).

Це загальний опис процесу розробки шаблону типового CRUD контролера для генерації контролера на основі таблиці з бази даних. Реалізація може варіюватися в залежності від обраного фреймворку або мови програмування.

Розробимо сторінку вибору таблиць для генерації CRUD (Рис. 6 та 7). Select — це елемент за допомогою, якого можна вибрати таблицю. Також реалізуємо javascript бібліотеку select2 для більшій зручності користувача.

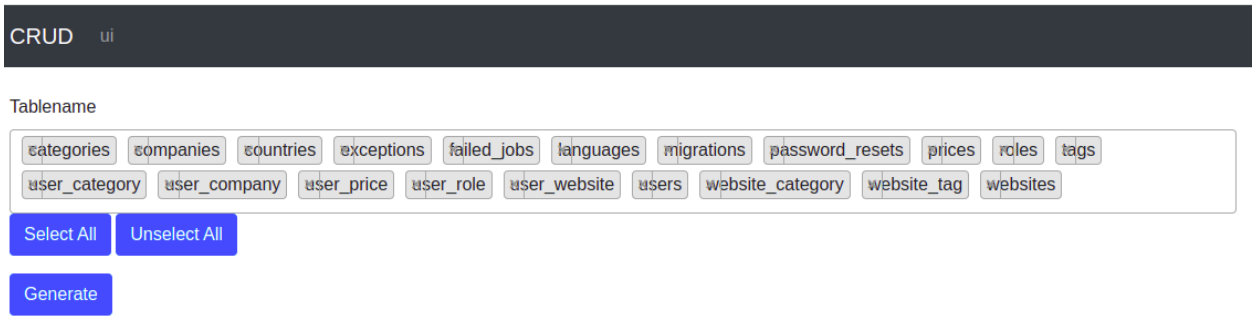

Рисунок 6 — *Вигляд представлення генератора*

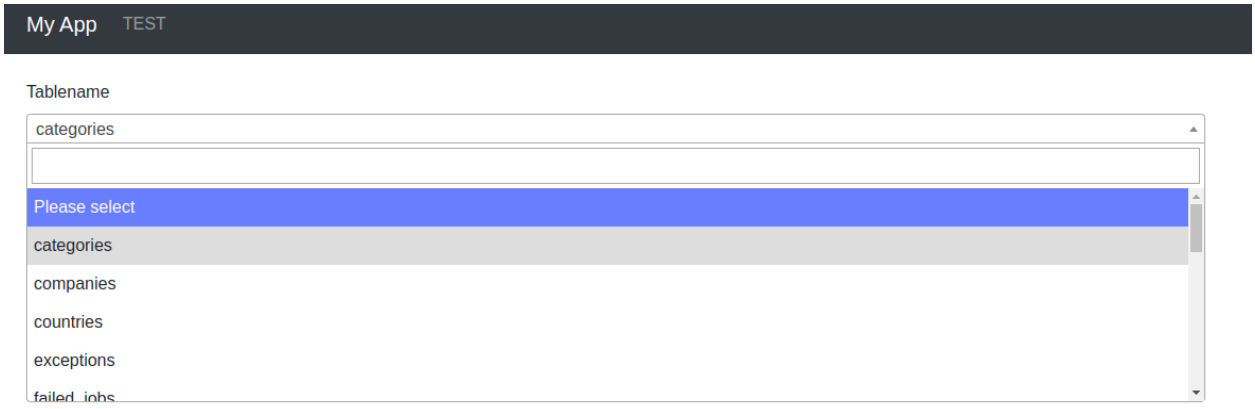

Рисунок 7 — *Вигляд представлення генератора та вибору таблиці*

Розробимо метод контролера, який підвантажить список таблиць із бази даних, і відобразить їх.

Лістинг 17 *Реалізація методу вибору таблиці*

```
public function user_interface()
{
    $db = db_connect();
    $tables = $db->listTables();
    return view("user_interface", compact('tables'));
}
```

Цей код представляє дію (action) з контролера або модуля веб-додатку. Функція user_interface() викликається для відображення інтерфейсу користувача (user interface) для перегляду списку таблиць в базі даних.

Розглянемо функцію крок за кроком:

Функція user_interface() не приймає жодних параметрів.

У першому рядку функції виконується підключення до бази даних за допомогою функції db_connect(). Здатність підключитися до бази даних залежить від налаштувань підключення, які містяться у конфігураційному файлі додатку. За допомогою $db->listTables() отримуються імена всіх таблиць в базі даних. Отримані імена таблиць передаються у змінну $tables.

Функція повертає результат виклику view("user_interface", compact('tables')). Цей код вказує на те, що функція використовує перегляд user_interface, який приймає змінну $tables для відображення списку таблиць на сторінці інтерфейсу користувача.

У перегляді user_interface можна використовувати змінну $tables для відображення списку таблиць бази даних на сторінці інтерфейсу користувача.

Цей код передбачає використання деякого фреймворка або системи керування базами даних, яка дозволяє підключатися до бази даних та отримувати інформацію про її структуру (наприклад, імена таблиць). Після цього інформацію можна відобразити на сторінці інтерфейсу користувача.

3.1.1 Розробка шаблону методу index

Шаблон методу index повинен зберігати незмінні частини коду, які незалежні від назви таблиці, сутності:

Лістинг *18 Реалізація методу index*

```
/**
 * @return string
 */
public function index(): string
{
    $model = new table_nameModel();

    $post = $this->request->getVar();
    $r = $model->_addFilters($post);

    $data['TABLE_NAME'] = $model->paginate($this->paginate_pages);
    $data['pager'] = $model->pager;
    $data['_item'] = $post;
    $data['r'] = $r;
    return view('TABLE_NAME/index', $data);
}
```

Цей код є методом index у контролері CodeIgniter 4 і виконує деякі дії при обробці запиту на головну сторінку (індекс) для ресурсу, представленого моделлю з ім'ям table_nameModel.

Розберемо кожен рядок цього методу:

$model = new table_nameModel();: Створення екземпляру моделі table_nameModel. Тут "table_name" вказує на те, що використовується модель для роботи з таблицею бази даних певного ресурсу. У генераторі воно замінює "table_name" на реальне ім'я таблиці, якою ви керуєте.

$post = $this->request->getVar();: Отримання всіх змінних запиту. Цей код отримує дані, які були відправлені на сервер, наприклад, з форми. Однак без аргументів функція getVar повертає масив усіх даних запиту.

$r = $model->_addFilters($post);: Виклик методу _addFilters моделі з переданими змінними запиту. Цей метод відповідає за додавання фільтрів до запиту для обмеження вибірки даних, також він виключає пусті фільтри з перемметрів пошуку.

$data['TABLE_NAME'] = $model->paginate($this->paginate_pages);: Виклик методу paginate для моделі з переданою кількістю сторінок на сторінці (визначено конфігураційним параметром $this->paginate_pages). Результатом є пагінована вибірка даних з таблиці. TABLE_NAME замінюється генератором на назву сутності.

$data['pager'] = $model->pager;: Передача об'єкта пагінації в масив даних.

$data['_item'] = $post;: Передача змінних запиту в масив даних. _item - це, змінна, яка заповнює форму пошуку, даними, які були надіслані браузером після пошуку.

$data['r'] = $r;: Передача заповнених параметрів пошуку до представлення для формування параметрів списку посилань посторінкової навігації.

return view('TABLE_NAME/index', $data);: Повернення представлення з ім'ям "TABLE_NAME/index" та передача масиву даних. Тут "TABLE_NAME" вказує на папку з представленнями, яка відповідає імені таблиці. TABLE_NAME замінюється генератором на назву сутності.

Цей метод призначений для відображення списку ресурсів у додатку, де дані пагінуються та можуть бути відфільтровані залежно від даних, що надходять через змінні запиту.

3.1.2 Розробка шаблону методу edit

Шаблон методу edit повинен зберігати незмінні частини коду, які незалежні від назви таблиці, сутності, а також передачена можливість перевірки правильности даних введених користувачем :

Лістинг 19 *Реалізація методу edit*

```
/**
 * saves existing entity
 * @param string|int|null $id
 * @return RedirectResponse|RedirectException|string
 * @throws ReflectionException|RedirectException
 */
```

```
public function edit(string|int|null $id = null):
string|RedirectResponse|RedirectException
{
    $model = new table_nameModel();
    $data = [];
    if ($id) {
        $data = $model->gettable_nameById($id);
        if (!$data) {
            return redirect()->to('/table_name');
        }
    }
    if ($this->request->getVar()) {
        $data = $this->request->getVar();
        $result = $id ?
            $model->updatetable_name($id, $data) :
            $model->addtable_name($data);
        if ($result) {
            return redirect()->to('/table_name');
        }
    }
    $parentEntities = [];
    $model->initFkModels();
    foreach ($model->fkModels as $key => $field) {
        //$parentEntities[$key] = $field->findAll();
    }
    return view('TABLE_NAME/create', [
        'validation' => $model->validation,
        'TABLE_NAME' => $data,
        'parentEntities' => $parentEntities]
    );
}
```

Цей код представляє метод контролера у CodeIgniter 4, який відповідає за редагування існуючого запису для певного ресурсу, представленого моделлю з ім'ям table_nameModel.

Розглянемо кожний рядок цього методу:

$model = new table_nameModel();: Створення екземпляру моделі table_nameModel. Тут "table_name" вказує на те, що використовується модель для роботи з таблицею бази даних певного ресурсу, "table_name" замінюється генератором на назву сутності.

$data = [];: Ініціалізація порожнього масиву $data, який буде використовуватися для передачі даних в представлення.

if ($id) { ... }: Умова перевірки, чи існує ідентифікатор ($id) для редагування. Якщо ідентифікатор передано, тоді виконуються наступні дії:

$data = $model->gettable_nameById($id);: Виклик методу моделі для отримання даних про існуючий запис за його ідентифікатором.

if (!$data) { return redirect()->to('/table_name'); }: Якщо запис не знайдено, відбувається перенаправлення користувача на сторінку списку ресурсів.

if ($this->request->getVar()) { ... }: Умова перевірки, чи є дані, відправлені з форми редагування. Якщо так, тоді виконуються наступні дії:

$data = $this->request->getVar();: Отримання змінних запиту.

$result = $id ? $model->updatetable_name($id, $data) : $model->addtable_name($data);: Виклик методу моделі для оновлення або додавання запису, залежно від того, чи є ідентифікатор. Результат збереження даних вказує на успіх або невдачу.

if ($result) { return redirect()->to('/table_name'); }: Якщо збереження пройшло успішно, відбувається перенаправлення користувача на сторінку списку ресурсів.

$parentEntities = [];: Ініціалізація порожнього масиву $parentEntities, який буде використовуватися для передачі даних батьківських сутностей в представлення.

$model->initFkModels();: Виклик методу моделі для ініціалізації зовнішніх ключів (FK), які можуть бути використані для відображення батьківських сутностей.

foreach ($model->fkModels as $key => $field) { ... }: Цикл, який може бути використаний для отримання списку батьківських сутностей.

return view('TABLE_NAME/create', ['validation' => $model->validation, 'TABLE_NAME' => $data, 'parentEntities' => $parentEntities]);: Повернення представлення з ім'ям "TABLE_NAME/create" та передача масиву даних, який містить дані для відображення у формі редагування, об'єкт валідації та список батьківських сутностей. TABLE_NAME замінюється генератором на назву сутності.

Цей метод призначений для відображення форми редагування для існуючого запису та обробки збереження введених даних.

3.1.3 Розробка шаблону методу delete

Шаблон методу delete повинен зберігати незмінні частини коду, які незалежні від назви таблиці, сутності:

Лістинг *20 Реалізація методу delete*

```
public function delete($id)
{
    $model = new table_nameModel();
    $model->deletetable_name($id);
    return redirect()->to('/table_lower');
}
```

Основні кроки, які виконуються в даному коді:

Створення об'єкту моделі: Спочатку створюється об'єкт моделі table_nameModel, де table_name є назвою таблиці, з якою взаємодіємо.

Виклик методу видалення: Далі викликається метод deletetable_name($id) на об'єкті моделі. Передається параметр $id, що представляє ідентифікатор запису, який необхідно видалити з таблиці.

Перенаправлення користувача: Після видалення запису виконується перенаправлення користувача на /table_lower. table_lower це шлях до сторінки, на яку перенаправляється користувач після успішного видалення запису.

У підсумку, код вищче видаляє запис з таблиці бази даних за допомогою моделі та перенаправляє користувача на іншу сторінку після видалення. Варто відзначити, що код не містить перевірок безпеки, тому слід впевнитися, що переданий ідентифікатор $id відповідає допустимим значенням та виконуються відповідні перевірки доступу перед видаленням запису.

3.1.5 Розробка шаблону методу view

Шаблон методу view повинен зберігати незмінні частини коду, які незалежні від назви таблиці, сутності, а також передачена можливість перевірки правильности даних введених користувачем

Лістинг *21 Реалізація шаблону методу view*

```
/**
 * @param string|int $id
 * @return string
 */
public function view(string|int $id): string
{
    $model = new table_nameModel();
    $data['TABLE_NAME'] = $model->gettable_nameById($id);
    return view('TABLE_NAME/view', $data);
}
```

Цей код представляє метод контролера CodeIgniter 4, який відповідає за відображення деталей конкретного запису (за вказаним ідентифікатором) для певного ресурсу, представленого моделлю з ім'ям table_nameModel.

Розглянемо кожний рядок цього методу:

$model = new table_nameModel();: Створення екземпляру моделі table_nameModel. Тут "table_name" вказує на те, що використовується модель для роботи з таблицею бази даних певного ресурсу. "table_name" буде замінен на реальне ім'я таблиці.

$data['TABLE_NAME'] = $model->gettable_nameById($id);: Виклик методу моделі для отримання даних конкретного запису за його ідентифікатором. Отримані дані зберігаються в асоціативному масиві з ключем "TABLE_NAME".

return view('TABLE_NAME/view', $data);: Повернення представлення з ім'ям "TABLE_NAME/view" та передача масиву даних для відображення у представленні. Тут "TABLE_NAME" вказує на папку з представленнями, яка відповідає імені таблиці. TABLE_NAME замінюється генератором на назву сутності.

Отже, цей метод призначений для відображення сторінки з деталями конкретного запису на основі переданого ідентифікатора.

3.2 Розробка шаблону моделі

Процес розробки шаблону типової моделі для генерації моделі на основі таблиці з бази даних включає наступні кроки:

Аналіз таблиці: Спочатку потрібно провести аналіз таблиці в базі даних, з якої необхідно згенерувати модель - структуру таблиці, поля, зв'язки, обмеження і будь-які особливості даних, які необхідно використовувати в моделі.

Визначення полів моделі: На основі вищезазначеного аналізу визначаємо поля моделі, які будуть відповідати полям таблиці. Врахувати типи даних, розміри, обмеження і будь-які інші властивості полів.

Визначення відношень: Якщо таблиця має відношення з іншими таблицями, визначаємо ці відношення. Наприклад, зв'язки один до одного, один до багатьох або багато до багатьох між необхідною таблицею і іншими таблицями.

Створення шаблону моделі: На основі вищезазначених аналізу і визначення полів і відношень розроблено шаблон моделі. Шаблон може містити клас моделі з відповідними полями і методами для доступу до даних, збереження, оновлення та видалення.

Реалізація методів доступу до даних: Це може включає методи для створення нових записів, читання даних з бази даних, оновлення записів і видалення записів, валідацію даних та обробку помилок, які можуть виникати під час доступу до даних.

Інтеграція з іншими компонентами: Модель може бути частиною більшої архітектури програмного забезпечення. Її можна інтегрувати з іншими компонентами, такими як контролери, представлення і сервіси, для створення повноцінного додатка.

Це загальний опис процесу розробки шаблону типової моделі для генерації моделі на основі таблиці з бази даних. Реалізація може варіюватися в залежності від обраного фреймворку або мови програмування.

Лістинг 22 *Реалізація методу model*

```
public function model($table = null): void
{
    $table = $table ?: $this->request->getVar('table');
    $info = CRUDTool::tableInfo($table);
    $fields_fk = CRUDTool::modelFieldsFK($table);
    $validation = CRUDTool::validation($info);
    $this->writeFile(APPPATH . 'Models/' . $info['Table'] . 'Model.php', (new
ModelRenderer())->render([
        implode("', '", $info['COLUMN_NAME']),
        implode("', '", $info['COLUMN_NAME_PARTIAL_MATCH']),
        implode("', '", $info['COLUMN_NAME_EXACT_MATCH']),
        $info['Table'], $info['pk'], $info['table'], $validation, $fields_fk,
    ]), 'Model');
    $this->controller($info);
}
```

Цей код виглядає як метод, що відноситься до генерації файлу моделі для CodeIgniter 4:

$table = $table ?: $this->request->getVar('table');: Цей рядок призначений для отримання назви таблиці з параметра $table, або, якщо він порожній, то з параметра запиту з іменем 'table'. Виглядає як динамічна можливість вказати таблицю.

$info = CRUDTool::tableInfo($table);: Отримання інформації про таблицю за допомогою статичного методу tableInfo класу CRUDTool. Ця інформація включає ім'я колонки, типи, ключі, тощо.

$fields_fk = CRUDTool::modelFieldsFK($table);: Отримання інформації про зовнішні ключі (foreign keys) за допомогою статичного методу modelFieldsFK класу CRUDTool.

$validation = CRUDTool::validation($info);: Отримання правил валідації для моделі за допомогою статичного методу validation класу CRUDTool. Вертає масив правил валідації для використання в моделі.

$this->writeFile(APPPATH . 'Models/' . $info['Table'] . 'Model.php', ...);: Виклик методу writeFile для запису результату у файл моделі. Виглядає як генерація PHP-файлу моделі.

(new ModelRenderer())->render([...]);: Створення екземпляру класу ModelRenderer і виклик його методу render з передачею деяких параметрів. Відповідає за генерацію тексту PHP-коду моделі на основі переданих даних.

$this->controller($info);: Виклик методу контролера (який відсутній в даному фрагменті коду). Це частина більшого процесу генерації контролера.

Загалом, цей код частина процесу автоматизованої генерації файлу моделі для CodeIgniter 4 на основі інформації про таблицю та зовнішні ключі.

3.3 Розробка шаблону представлення

Для роботи CRUD MVC необхідні представлення до кожного методу у CRUD контролері, отже розробимо шаблони до генерації представлення методів CRUD контролера.

3.3.1 Розробка шаблону представлення редагування — створення

Лістинг *23 Реалізація шаблону представлення редагування - створення*

```
<?= $this->extend('layouts/main') ?>
<?= $this->section('content') ?>
<?= form_open(isset($table_lower) ? "table_lower/update/{$table_lower['id']}"
: 'table_lower/store') ?>
<div class="container mb-5">
    update_fields

    <button type="submit" class="btn btn-primary"><?= isset($table_lower) ?
'Update' : 'Create' ?></button>
    <a href="<?= site_url('table_lower') ?>" class="btn btn-link">Cancel</a>
</div>
<?= form_close() ?>
<?= $this->endSection() ?>
```

Цей код використовує шаблонний рушій або фреймворк для створення сторінок веб-додатку.

Розглянемо кожну частину коду:

<?= $this->extend('layouts/main') ?>: Цей рядок розширює (extends) головний шаблон з іменем main, що містить загальний лейаут, який використовується на всіх сторінках.

<?= $this->section('content') ?>: Цей рядок починає розділ content в шаблоні. Всі елементи, які входять до цього розділу, будуть вставлені в місце зазначене як <?= $this->section('content') ?> у головному шаблоні main.

<?= form_open($table_lower ? "table_lower/update/{$table_lower['id']}" : 'table_lower/store') ?>: Цей рядок створює HTML-форму з допомогою функції form_open(). Шлях дії форми буде визначено в залежності від того, чи існує змінна $table_lower. Якщо змінна $table_lower існує, то форма буде вказувати шлях 'table_lower/update/{$table_lower['id']}', інакше - 'table_lower/store'.

: Цей блок відповідає за контейнер сторінки з класом container та стилізацією mb-5 (margin-bottom: 5).

update_fields: Цей текст update_fields може представляти список полів для оновлення або редагування.

<button type="submit" class="btn btn-primary"><?= isset($table_lower) ? 'Update' : 'Create' ?></button>: Цей тег <button> представляє кнопку для відправки форми. Текст кнопки буде 'Update', якщо змінна $table_lower існує, і 'Create', якщо змінна $table_lower не існує.

<a href="<?= site_url('table_lower') ?>" class="btn btn-link">Cancel: Цей тег <a> представляє посилання на сторінку 'table_lower'. Клас btn btn-link стилізує його як посилання.

<?= form_close() ?>: Закриваючий тег форми.

<?= $this->endSection() ?>: Цей рядок закриває розділ content в шаблоні.

В цілому, цей код створює сторінку, яка залежно від наявності змінної $table_lower відображатиме форму для редагування або створення даних. Форма використовується для відправки даних на сервер для збереження. Також сторінка має посилання "Cancel", яке перенаправить користувача на сторінку 'table_lower'. Весь контент сторінки вставлятиметься в головний шаблон main, що дозволяє застосувати єдиний макет на всіх сторінках.

3.3.2 Розробка шаблону представлення списку

Шаблон представлення повинен мати теги ‘table_lower’ ‘table_name’ щоб генератор їх замінив на відповідні значення, та вставила списки полів у HTML:

Лістинг *24 Реалізація шаблону представлення списку*

```
<?= $this->extend('layouts/main') ?>
<?= $this->section('content') ?>
<div class="my-4">
    <div class="row">
        <div class="p-5 col-12">
            <h1 class="mb-4">table_name List</h1>
            <a href="<?= site_url('table_lower/create') ?>" class="btn btn-
primary">Add table_name</a>

            <?php if (!empty($table_lower)){ ?>
                <div class="table-responsive">
                    <table class="table_lower table table-striped">
                        <thead>
                        <tr>column_th
                        </tr>
                        </thead>
                        <tbody>
                        <?php foreach ($table_lower as $item) { ?>
                            <tr>column_td
                                <td>
                                    <a href="<?=
site_url('table_lower/edit/'.$item['id']) ?>" class="btn btn-
primary">Edit</a>
                                    <form action="<?=
site_url('table_lower/delete/'.$item['id']) ?>" method="post"
                                          style="display: inline-block;">
                                        <?= csrf_field() ?>
                                        <input type="hidden" name="_method"
value="DELETE">
                                        <button type="submit" class="btn btn-
danger">Delete</button>
                                    </form>
                                </td>
                            </tr>
                        <?php } ?>
                        </tbody>
                    </table>
                </div>
                <div class="d-flex justify-content-between mb-3">
                    <div class="align-self-center">
                        <?= $pager->links() ?>
                    </div>
                </div>

            <?php }else{ ?>
                <p>No table_lower found.</p>
            <?php } ?>
```

```
        </div>
    </div>
</div>
<?= $this->endSection() ?>
```

Цей код містить шаблон перегляду, який використовується для відображення списку даних таблиці "table_name". Детальний опис розділів шаблону:

<?= $this->extend('layouts/main') ?>: Цей рядок вказує, що шаблон перегляду "table_lower" розширює (використовує) головний шаблон з назвою "main".

<?= $this->section('content') ?>: Цей рядок вказує початок секції "content", яка визначає основний вміст сторінки.

: Зовнішній контейнер сторінки, який містить відступ від верхнього краю сторінки.

: Рядок для розміщення вмісту сторінки.

: Контейнер, який містить вміст списку таблиці "table_lower".

<h1 class="mb-4">table_name List</h1>: Заголовок сторінки, що відображає назву таблиці.

<a href="<?= site_url('table_lower/create') ?>" class="btn btn-primary">Add table_name: Посилання для додавання нового запису до таблиці.

<?php if (!empty($table_lower)){ ?>: Перевірка наявності записів у таблиці. Якщо записи є, виконується наступний блок коду, в якому виводиться таблиця з даними.

: Контейнер, який дозволяє забезпечити адаптивний дизайн для таблиці, якщо вміст таблиці не поміщається на екрані.

<table class="table_lower table table-striped">: Таблиця, яка містить клас "table_lower" для здійснення налаштувань CSS та "table-striped" для відображення смугастого вигляду таблиці.

<thead>...</thead>: Частина таблиці, де розміщаються заголовки стовпців.

<tbody>...</tbody>: Частина таблиці, де розміщаються дані з бази даних, витягнуті з змінної "$table_lower".

<a href="<?= site_url('table_lower/edit/'.$item['id']) ?>" class="btn btn-primary">Edit: Посилання "Edit" для редагування окремого запису таблиці.

<form action="<?= site_url('table_lower/delete/'.$item['id']) ?>" method="post" style="display: inline-block;">: Форма для видалення окремого запису таблиці.

<?= csrf_field() ?>: Поле для захисту від CSRF-атак, яке генерує токен CSRF.

<input type="hidden" name="_method" value="DELETE">: Приховане поле з значенням "DELETE", яке вказує на метод HTTP DELETE для виконання видалення.

<button type="submit" class="btn btn-danger">Delete</button>: Кнопка для виконання видалення запису.

<?php }else{ ?>: Блок коду, який виконується, якщо в таблиці "table_lower" немає жодного запису. Виводиться повідомлення "No table_lower found.".

<?= $pager->links() ?>: Виведення посилань для пагінації результатів таблиці, яка дозволяє розділити результати на сторінки.

<?= $this->endSection() ?>: Кінець секції "content", яка визначає основний вміст сторінки.

3.4 Висновки з розділу 3

1. При розробці CRUD генератора для CodeIgniter 4 було проведено ретельний аналіз вимог і вимог до функціональності, необхідної для задоволення потреб користувачів. Були ідентифіковані ключові можливості, які повинні

бути реалізовані для створення ефективного інструменту для автоматизованого створення операцій CRUD.

2. В ході розробки генератора було використано сучасні технології та найкращі практики програмування для забезпечення ефективної роботи і високої якості коду. Використання CodeIgniter 4 як основи дозволило забезпечити швидку і надійну розробку.

3. За допомогою розробленого генератора, розробники можуть значно зекономити час і зусилля при створенні базового CRUD функціоналу своїх додатків. Це дозволяє зосередитися на більш важливих аспектах розробки, таких як бізнес-логіка та функціональність.

4. У подальшому, розроблений генератор може бути розширений та доповнений новими функціями і можливостями, що дозволить зробити його більш гнучким та універсальним для різних типів проектів та завдань.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ГЕНЕРАЦІЇ CRUD КОНТРОЛЛЕРА МОДЕЛІ ПРЕДСТАВЛЕННЯ

4.1 Результати генерації CRUD MVC

Після генерації відповідні файли записуються до папок Controllers, Models, Views/table_name. Розглянемо результати генерації моделей, представлень, контролерів.

4.1.1 Результати генерації контроллера

Генерація представлення змінила теги ‘table_lower’ ‘table_name’ на відповідні значення, та вставила списки полів у php код:

Лістинг 25 *Реалізація контроллера*

```
<?php

namespace App\Controllers;

use App\Models\CategoriesModel;
use CodeIgniter\Controller;
use CodeIgniter\HTTP\RedirectResponse;
use CodeIgniter\Router\Exceptions\RedirectException;
use ReflectionException;

class Categories extends Controller
{
    private int $paginate_pages = 5;

    /**
     * @return string
     */
    public function index(): string
    {
        $model = new CategoriesModel();

        $post = $this->request->getVar();
        $r = $model->_addFilters($post);

        $data['categories'] = $model->paginate($this->paginate_pages);
        $data['pager'] = $model->pager;
        $data['_item'] = $post;
        $data['r'] = $r;
        return view('categories/index', $data);
    }

    /**
```

```
     * saves existing entity
     * @param string|int|null $id
     * @return RedirectResponse|RedirectException|string
     * @throws ReflectionException|RedirectException
     */
    public function edit(string|int|null $id = null):
string|RedirectResponse|RedirectException
    {
        $model = new CategoriesModel();
        $data = [];
        if ($id) {
            $data = $model->getCategoriesById($id);
            if (!$data) {
                return redirect()->to('/Categories');
            }
        }
        if ($this->request->getVar()) {
            $data = $this->request->getVar();
            $result = $id ? $model->updateCategories($id, $data) : $model-
>addCategories($data);
            if ($result) {
                return redirect()->to('/Categories');
            }
        }
        $parentEntities = [];
        $model->initFkModels();
        foreach ($model->fkModels as $key => $field) {
            //$parentEntities[$key] = $field->findAll();
        }
        return view('categories/create', [
            'validation' => $model->validation,
            'categories' => $data,
            'parentEntities' => $parentEntities
        ]);
    }

    /**
     * @param string|int $id
     * @return string
     */
    public function view(string|int $id): string
    {
        $model = new CategoriesModel();
        $data['categories'] = $model->getCategoriesById($id);
        return view('categories/view', $data);
    }

    /**
     * @param string|int $id
     * @return RedirectResponse
     */
    public function delete(string|int $id): RedirectResponse
    {
        $model = new CategoriesModel();
        $model->deleteCategories($id);
        return redirect()->to('/Categories');
    }
}
```

Цей код представляє контролер Categories для фреймворку CodeIgniter Контролер відповідає за обробку запитів, пов'язаних з категоріями, такі

як перегляд, створення, редагування та видалення категорій. Контролер використовує модель CategoriesModel для здійснення дій з базою даних.

Розглянемо кожний метод контролера:

index(): Метод index() викликається для відображення списку категорій. Він отримує дані з бази даних використовуючи модель CategoriesModel, використовує пагінацію для розділення результатів на сторінки та передає дані до відображення categories/index.

edit($id): Метод edit($id) викликається для редагування категорії з вказаним ідентифікатором $id. Він отримує дані категорії з бази даних за допомогою моделі CategoriesModel і передає їх до відображення categories/create для редагування.

update(): Метод update() викликається при надсиланні форми редагування та створення категорії. Він отримує дані з POST-запиту, валідує їх за допомогою моделі CategoriesModel і оновлює категорію з вказаним ідентифікатором. Якщо оновлення вдалося, користувач перенаправляється на сторінку зі списком категорій (/categories). У протилежному випадку, якщо виникла помилка при валідації або оновленні, користувач знову відображається на сторінці редагування з повідомленням про помилки.

delete($id): Метод delete($id) викликається для видалення категорії з вказаним ідентифікатором $id. Він видаляє категорію з бази даних за допомогою моделі CategoriesModel і перенаправляє користувача на сторінку зі списком категорій (/categories).

Загалом, цей контролер забезпечує базовий CRUD функціонал для роботи з категоріями на стороні сервера. Він демонструє використання моделі для доступу до даних, роботу з пагінацією, валідацію даних та перенаправлення користувача на різні сторінки залежно від результатів операцій CRUD.

4.1.2 Результати генерації моделі

Згенерована модель містить назву таблиці конвертовану до назви класу, назву таблиці, список полів, а також правила валідації, сформовані на основі властивостей полів таблиці.

Лістинг 26 *Результат згенерованої моделі на основі таблиці categories бази даних crm*

```
<?php

namespace App\Models;

use CodeIgniter\Database\BaseResult;
use CodeIgniter\Database\ConnectionInterface;
use CodeIgniter\Model;
use CodeIgniter\Validation\ValidationInterface;
use ReflectionException;

class CategoriesModel extends Model
{
    protected $table = 'categories';
    protected $primaryKey = 'id';
    protected $allowedFields = ['id', 'name', 'parent_id', 'created_at',
'updated_at'];
    protected $validationRules = [
        'id' =>
'permit_empty|integer|greater_than_equal_to[0]|less_than_equal_to[18446744073
709551615]',
        'name' => 'max_length[255]',
        'parent_id' =>
'permit_empty|integer|greater_than_equal_to[0]|less_than_equal_to[18446744073
709551615]',
        'created_at' => 'permit_empty|valid_date',
        'updated_at' => 'permit_empty|valid_date',

    ];

    protected array $partialMatchFields = ['name'];
    protected array $exactMatchFields = ['id', 'parent_id'];

    protected $beforeInsert = ['convertEmptyStringsToNull'];
    protected $beforeUpdate = ['convertEmptyStringsToNull'];

    public array $fkModels = [
        'parent_id' => 'App\Models\CategoriesModel',

    ];

    public function initFkModels()
    {
        foreach ($this->fkModels as $key => $fkModel) {
            $this->fkModels[$key] = new $this->fkModels[$key];
        }
    }


    protected function convertEmptyStringsToNull(array $data): array
    {
        foreach ($data['data'] as $key => &$value) {
```

```
            if ($value === '') {
                unset($data['data'][$key]);
            }
        }
        return $data;
    }

    public function getCategoriesById(string|int $id): object|array|null
    {
        return $this->find($id);
    }

    public function _addFilters(array $data): array
    {
        $r = [];
        foreach ($data as $key => $item) {
            if (!empty($item) && $key !== 'page') {
                if (in_array($key, $this->partialMatchFields)) {
                    $this->like($key, $item, 'after');
                } else if (in_array($key, $this->exactMatchFields)) {
                    $this->where($key, $item);
                } else {
                    $this->like($key, $item); //todo use preferred filtering
option
                }
                $r[$key] = $item;
            }
        }
        return $r;
    }

    public function addCategories(array $data): object|bool|int|string
    {
        if (!$this->validate($data)) {
            return false;
        }
        return $this->insert($data);
    }

    public function updateCategories(string|int $id, array $data): bool
    {
        if (!$this->validate($data)) {
            return false;
        }
        return $this->update($id, $data);
    }

    public function deleteCategories(string|int $id): bool|BaseResult
    {
        return $this->delete($id);
    }
}
```

Цей код представляє модель CategoriesModel для фреймворку CodeIgniter 4. Модель використовується для взаємодії з таблицею "categories" в базі даних та виконання операцій з даними пов'язаними з категоріями.

Розглянемо кожний метод моделі:

getCategories(): Метод getCategories() повертає всі категорії з таблиці "categories". Він використовує findAll() для отримання всіх записів.

getCategoriesById($id): Метод getCategoriesById($id) повертає категорію з таблиці "categories" з вказаним ідентифікатором $id. Він використовує find($id) для отримання запису з вказаним ключем.

addCategories($data): Метод addCategories($data) додає нову категорію до таблиці "categories" з даними, які передані у змінній $data. Перед додаванням, він перевіряє валідацію за допомогою методу validate().

updateCategories($id, $data): Метод updateCategories($id, $data) оновлює категорію з вказаним ідентифікатором $id в таблиці "categories" з новими даними, переданими у змінній $data. Перед оновленням, він перевіряє валідацію за допомогою методу validate().

deleteCategories($id): Метод deleteCategories($id) видаляє категорію з таблиці "categories" з вказаним ідентифікатором $id.

joinCategories($id = null): Метод joinCategories($id = null) здійснює запит до бази даних, щоб отримати зв'язані категорії, використовуючи з'єднання (join) між таблицями "categories" та "categories". Якщо вказано ідентифікатор $id, то виконується фільтрація результатів за цим ідентифікатором.

Модель також містить декілька властивостей:

$table: Назва таблиці, з якою пов'язана модель ("categories").
$primaryKey: Поле, яке виступає в якості первинного ключа таблиці ("id").
$allowedFields: Масив із переліком дозволених полів для масового присвоєння, які використовуються в методах insert() та update().

Загалом, ця модель забезпечує інтерфейс для отримання, додавання, оновлення та видалення даних категорій з бази даних. Вона також надає можливість отримати з'єднані категорії, що є корисною функціональністю для роботи з ієрархічними даними категорій.

4.1.3 Результати генерації представлення

Генерація представлення змінила теги ‘table_lower’ ‘table_name’ на відповідні значення, та вставила списки полів у HTML:

Лістинг *27 Результат згенерованого представлення index основі таблиці categories*

```
<?= $this->extend('layouts/main') ?>
<?= $this->section('content') ?>
<div class="my-4">
    <div class="row">
        <div class="p-5 col-12">
            <h1 class="mb-4">Categories List</h1>
            <a href="<?= site_url('categories/create') ?>" class="btn btn-
primary">Add Categories</a>

            <?php if (!empty($categories)){ ?>
                <div class="table-responsive">
                    <table class="categories table table-striped">
                        <thead>
                        <tr>
                        <th> id </th>
                        <th> name </th>
                        <th> parent_id </th>
                        <th> created_at </th>
                        <th> updated_at </th>
                        </tr>
                        </thead>
                        <tbody>
                        <?php foreach ($categories as $item) { ?>
                            <tr>
                            <td><?= $item['id'] ?></td>
                            <td><?= $item['name'] ?></td>
                            <td><?= $item['parent_id'] ?></td>
                            <td><?= $item['created_at'] ?></td>
                            <td><?= $item['updated_at'] ?></td>
                                <td>
                                    <a href="<?=
site_url('categories/edit/'.$item['id']) ?>" class="btn btn-primary">Edit</a>
                                    <form action="<?=
site_url('categories/delete/'.$item['id']) ?>" method="post"
                                          style="display: inline-block;">
                                        <?= csrf_field() ?>
                                        <input type="hidden" name="_method"
value="DELETE">
                                        <button type="submit" class="btn btn-
danger">Delete</button>
                                    </form>
                                </td>
                            </tr>
                        <?php } ?>
                        </tbody>
                    </table>
                </div>
                <div class="d-flex justify-content-between mb-3">
                    <div class="align-self-center">
                        <?= $pager->links() ?>
                    </div>
                </div>
```

```
            <?php }else{ ?>
                <p>No categories found.</p>
            <?php } ?>
        </div>
    </div>
</div>
<?= $this->endSection() ?>
```

Цей код містить шаблон перегляду для відображення списку даних таблиці "categories". Детальний опис розділів шаблону:

<?= $this->extend('layouts/main') ?>: Цей рядок вказує, що шаблон перегляду "categories" розширює (використовує) головний шаблон з назвою "main".

<?= $this->section('content') ?>: Цей рядок вказує початок секції "content", яка визначає основний вміст сторінки.

: Зовнішній контейнер сторінки, який містить відступ від верхнього краю сторінки.

: Рядок для розміщення вмісту сторінки.

: Контейнер, який містить вміст списку даних таблиці "categories".

<h1 class="mb-4">Categories List</h1>: Заголовок сторінки, що відображає назву таблиці "Categories".

<a href="<?= site_url('categories/create') ?>" class="btn btn-primary">Add Categories: Посилання для додавання нового запису до таблиці "categories".

<?php if (!empty($categories)){ ?>: Перевірка наявності записів у таблиці "categories". Якщо записи є, виконується наступний блок коду, в якому виводиться таблиця з даними.

: Контейнер, який дозволяє забезпечити адаптивний дизайн для таблиці, якщо вміст таблиці не поміщається на екрані.

<table class="categories table table-striped">: Таблиця, яка містить клас "categories" для здійснення налаштувань CSS та "table-striped" для відображення смугастого вигляду таблиці.

<thead>...</thead>: Частина таблиці, де розміщаються заголовки стовпців.

<tbody>...</tbody>: Частина таблиці, де розміщаються дані з бази даних, витягнуті з змінної "$categories".

<a href="<?= site_url('categories/edit/'.$item['id']) ?>" class="btn btn-primary">Edit: Посилання "Edit" для редагування окремого запису таблиці "categories".

<form action="<?= site_url('categories/delete/'.$item['id']) ?>" method="post" style="display: inline-block;">: Форма для видалення окремого запису таблиці "categories".

<?= csrf_field() ?>: Поле для захисту від CSRF-атак, яке генерує токен CSRF.

<input type="hidden" name="_method" value="DELETE">: Приховане поле з значенням "DELETE", яке вказує на метод HTTP DELETE для виконання видалення.

<button type="submit" class="btn btn-danger">Delete</button>: Кнопка для виконання видалення запису.

<?php }else{ ?>: Блок коду, який виконується, якщо в таблиці "categories" немає жодного запису. Виводиться повідомлення "No categories found.".

<?= $pager->links() ?>: Виведення посилань для пагінації результатів таблиці, яка дозволяє розділити результати на сторінки.

<?= $this->endSection() ?>: Кінець секції "content", яка визначає основний вміст сторінки.

Лістинг *28 Результат нагенерованого представлення index основі таблиці categories*

```
<?= $this->extend('layouts/main') ?>
<?= $this->section('content') ?>
<?= form_open(isset($categories) ? "categories/update/{$categories['id']}" : 'categories/store') ?>
<div class="container mb-5">
    <div class="form-group">
        <label for="id">id</label>
        <input type="text" name="id" id="id" class="form-control" value="<?= isset($categories) ? $categories['id'] : '' ?>">
    </div>
    <div class="form-group">
        <label for="name">name</label>
```

```
        <input   type="text"   name="name"   id="name"   class="form-control"
value="<?= isset($categories) ? $categories['name'] : '' ?>">
    </div>
    <div class="form-group">
        <label for="parent_id">parent_id</label>
        <input   type="text"   name="parent_id"   id="parent_id"   class="form-
control" value="<?= isset($categories) ? $categories['parent_id'] : '' ?>">
    </div>
    <div class="form-group">
        <label for="created_at">created_at</label>
        <input  type="text"  name="created_at"  id="created_at"  class="form-
control" value="<?= isset($categories) ? $categories['created_at'] : '' ?>">
    </div>
    <div class="form-group">
        <label for="updated_at">updated_at</label>
        <input  type="text"  name="updated_at"  id="updated_at"  class="form-
control" value="<?= isset($categories) ? $categories['updated_at'] : '' ?>">
    </div>
    <button  type="submit"  class="btn  btn-primary"><?=  isset($categories)  ?
'Update' : 'Create' ?></button>
    <a href="<?= site_url('categories') ?>" class="btn btn-link">Cancel</a>
</div>
<?= form_close() ?>
<?= $this->endSection() ?>
```

Цей код містить шаблон перегляду для створення або редагування даних таблиці "categories". Детальний опис розділів шаблону:

<?= $this->extend('layouts/main') ?>: Цей рядок вказує, що шаблон перегляду "categories" розширює (використовує) головний шаблон з назвою "main".

<?= $this->section('content') ?>: Цей рядок вказує початок секції "content", яка визначає основний вміст сторінки.

<?= form_open(isset($categories) ? "categories/update/{$categories['id']}" : 'categories/store') ?>: Форма для відправки даних до контролера для оновлення або створення запису в таблиці "categories". Якщо змінна $categories містить дані, то форма використовується для оновлення запису, інакше - для створення нового запису.

: Контейнер сторінки з відступом від нижнього краю сторінки.

... : Контейнер для кожного поля форми.

<label for="id">id</label>: Мітка для поля "id".

<input type="text" name="id" id="id" class="form-control" value="<?= isset($categories) ? $categories['id'] : '' ?>">: Поле вводу для значення "id" з можливим значенням за замовчуванням, яке буде використовуватись для оновлення існуючого запису.

<button type="submit" class="btn btn-primary"><?= isset($categories) ? 'Update' : 'Create' ?></button>: Кнопка для відправки форми для оновлення або створення запису. Текст кнопки змінюється на "Update" або "Create" залежно від того, чи наявні дані в змінній $categories.

<a href="<?= site_url('categories') ?>" class="btn btn-link">Cancel: Посилання для скасування дії і повернення до списку категорій.

<?= form_close() ?>: Кінець форми.

<?= $this->endSection() ?>: Кінець секції "content", яка визначає основний вміст сторінки.

Лістинг *29 Результат згенерованого представлення index основі таблиці categories*

```
$routes->get('/categories','Categories::index');
$routes->get('/categories/edit/(:num)','Categories::edit/$1');
$routes->get('/categories/delete/(:num)','Categories::delete/$1');
```

Цей код визначає маршрути для роутера (routes) фреймворку CodeIgniter 4. Він встановлює, які контролери та методи контролерів будуть викликатись для певних URL-адрес.

$routes->get('/categories','Categories::index');: Цей маршрут відповідає за сторінку зі списком категорій. Він викликає метод index контролера Categories при HTTP GET запиті на URL-адресу '/categories'.

$routes->get('/categories/edit/(:num)','Categories::edit/$1');: Цей маршрут відповідає за сторінку редагування категорії з вказаним ідентифікатором. Він викликає метод edit контролера Categories, передаючи ідентифікатор категорії у якості аргументу, при HTTP GET запиті на URL-адресу вигляду '/categories/edit/1', де '1' - ідентифікатор категорії.

$routes->get('/categories/delete/(:num)','Categories::delete/$1');: Цей маршрут відповідає за видалення категорії з вказаним ідентифікатором. Він викликає метод delete контролера Categories, передаючи ідентифікатор категорії у якості аргументу, при HTTP GET запиті на URL-адресу вигляду '/categories/delete/1', де '1' - ідентифікатор категорії.

Вищчезазначені маршрути визначають, які методи контролера Categories будуть викликатись для різних типів запитів і URL-адрес. Вони дозволяють створити структуру для роутінгу веб-додатку на платформі CodeIgniter 4.

4.2 Вигляд згенерованих CRUD контроллерів

Посторінкова навігація містить 5 сутностей на сторінці, кнопка додавання сутності, панель посторінкової навігації посилань, форма пошуку, кнопка пошуку (рисунок 8).

Список полей для пошуку та параметри пошуку формується на основі типу полів. Якщо тип поля число — тоді пошук по цьому полю йде по точному співпадінню, якщо тип поля рядок — тоді пошук по цьому полю буде за допомогою LIKE. Приклад: name LIKE ‘abc%’.

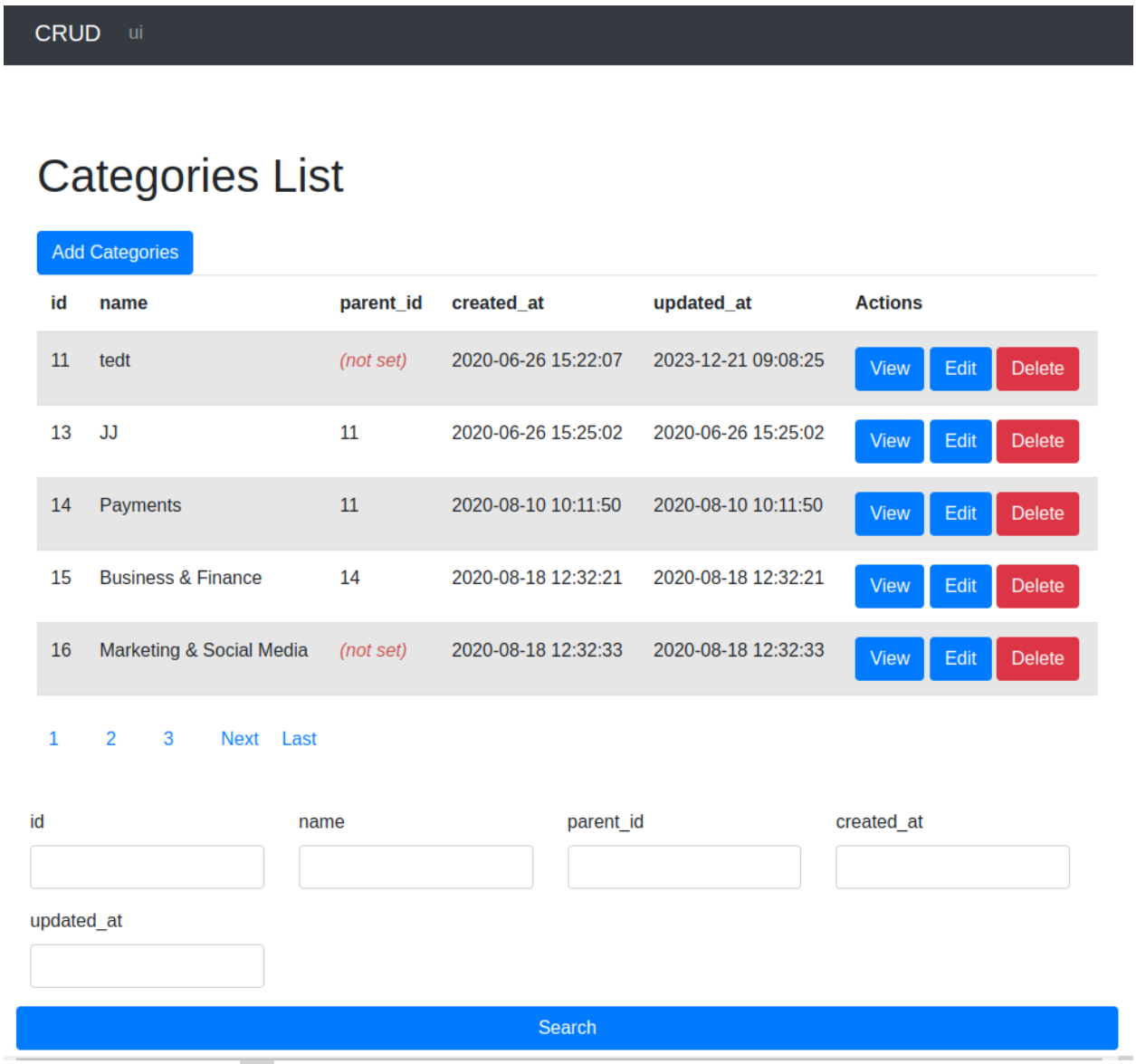

Рисунок 8 — *Вигляд списку категорій згенерованого контролера*

Згенероване представлення редагування сутності має відповідні поля, які дублюють поля таблиці, а також їх тип та властивості.

Текстові типи даних (CHAR, VARCHAR) відображаються як input type text. Числові типи даних (INT, BIGINT, SMALLINT, MEDIUMINT, SMALLINT, FLOAT, DOUBLE) відображаються як input type number. Тип даних ENUM відображається як HTML-тег select. Типи TEXT, SMALLTEXT, MEDIUMTEXT, LONGTEXT відображається як HTML-тег textarea. Якщо поле являє собою вторинний ключ — то відображається як HTML-тег select зі списком батьківської сутностей, які підвантажуються через пошук по первинному полю через AJAX запит.

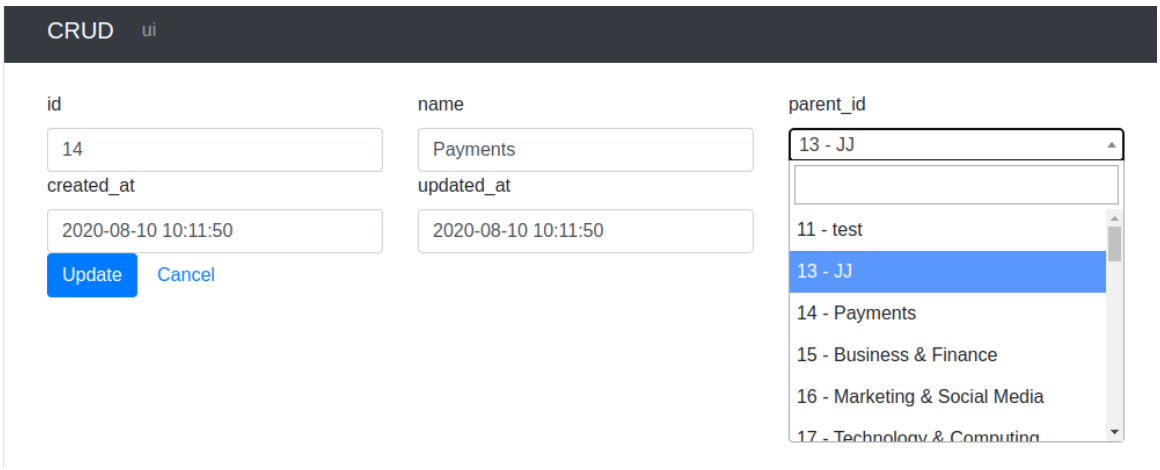

Рисунок 9 — *Вигляд створення категорії*

Представлення перегляду сутності формується таблицею де перша колонка — назви колонок таблиці, друга колонка — значення (рисунок 10).

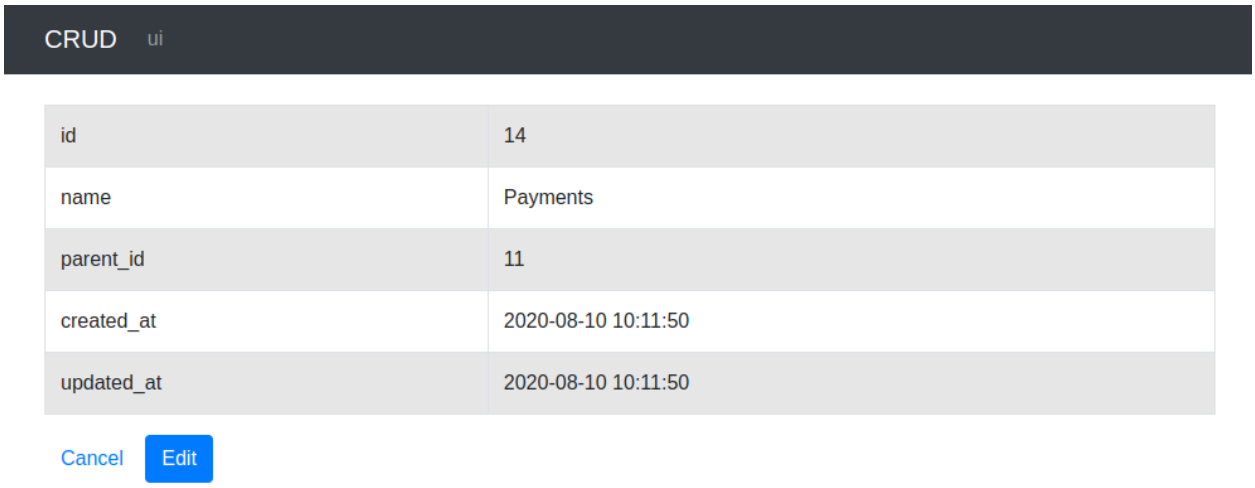

Рисунок 10 — *Вигляд представлення категорії*

У порівнянні з існуючим рішенням (рис 11) код розробленого рішення можливо адаптувати під потреби бізнес процесу

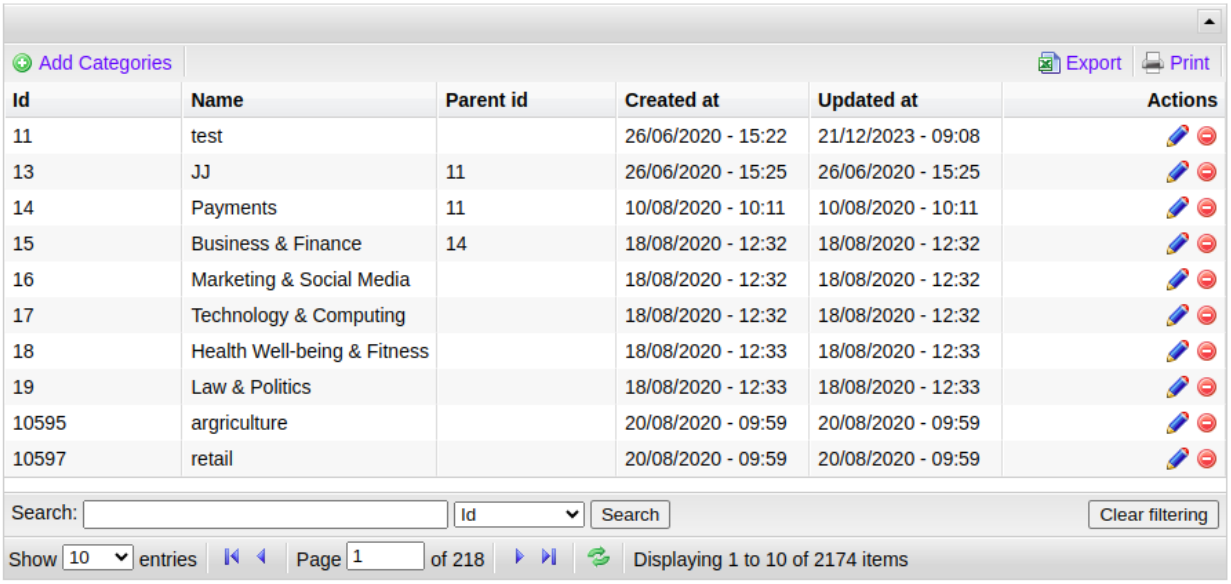

Рисунок 11 — Вигляд представлення категорії за допомогою існуюсого рішення — Codeigniter4 Grocery CRUD

4.3 Висновки з розділу 4

1. Автоматична генерація CRUD контролерів значно збільшила продуктивність розробки. Замість того, щоб створювати кожен контролер вручну, вдалося автоматизувати цей процес за допомогою кодогенерації. Це дозволило економити час та зусилля, особливо при створенні простих CRUD операцій.

2. Моделі, згенеровані автоматично, допомогли створити структуровану та ефективну базу даних. Застосування правильних відношень між моделями дозволило зберігати дані у структурованому форматі, що спрощує подальші операції з базою даних.

3. Створення автоматичних представлень значно спростив процес візуалізації даних. Кодогенерація представлень дозволила швидко створювати сторінки з переглядом, редагуванням та видаленням даних без необхідності писати багато однотипного коду.

4. Завдяки генерації CRUD коду, розробникам, використовуючим генератор, буде можливо покращити стандартизацію та однорідність у розробці проектів. Створення уніфікованого коду дозволяє легше зрозуміти та підтримувати проект в команді розробників.

5. Дослідження результатів генерації CRUD контролерів, моделей та представлень підтвердили їхню важливість у процесі розробки веб-додатків з використанням фреймворку. Вони створюють основні складові проекту та забезпечують швидке впровадження базового функціоналу.

6. У порівнянні з вже існуючим рішенням - Codeigniter GroceryCRUD розроблене рішення генерує код контроллеру, моделі, представлення, який надалі можливо допрацювати під потреби розробників та клієнтів, а існуюче рішення тільки відображає HTML представлення списку сутностей.

7. У порівнянні з вже існуючим рішенням - Laravel Nayjest GRID розроблене рішення генерує код контроллеру, моделі, представлення, який надалі можливо допрацювати під потреби розробників та клієнтів, а існуюче рішення тільки дає можливість відобразити посторінкове відображення списку сутностей з пошуком, редагування та видалення, видалення однієї сутності необхідно розробляти додатково.

ВИСНОВКИ

1. Досліджена проблема генерації стандартного коду посторінкової навігації сутностей.

2. Досліджені засоби для вирішення поставленої проблеми — за допомогою існуючих бібліотек генерації у фреймворці Yii2 Gii, та бібліотекою Nayjest Grids у фреймворці Laravel.

3. Розроблена готова до використання бібліотека для генерації CRUD контролерів на основі структури таблиць бази даних на PHP фреймворці Codeigniter4

4. Досліджені результати нагенерованих контроллерів

У порівнянні з вже існуючими рішеннями - Laravel бібліотеки Nayjest GRID, Codeigniter4 Grocery CRUD - розроблене рішення генерує код контроллеру, моделі, представлення, який надалі можливо допрацювати під потреби розробників та клієнтів, Laravel бібліотеки Nayjest GRID - тільки дає можливість відобразити посторінкове відображення списку сутностей з пошуком, редагування та видалення, видалення однієї сутності необхідно розробляти додатково, Codeigniter4 Grocery CRUD - тільки відображає HTML представлення списку сутностей, редагування та видалення сутностей.

5. Досліджені результати роботи розробленої комп’ютерної системи:

Результатом генерації є контроллер, з назвою, яка сформована за назвою таблиці у базі даних (наприклад Category.php), модель, з назвою сформованою за назвою таблиці у базі даних (наприклад CategoryModel.php), представлення, які зберігаються у теці, з назвою, сформованою за назвою таблиці (наприкла category/), представлення редагування та утворення з назвою create.php, представленя посторінкової навігації з назвою index.php, представлення форми пошуку за полями search.php, представлення перегляду однієї сутності, з назвою view.php

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications. Transactions on Machine Learning and Artificial Intelligence. 2017. Vol. 5. P. 259–266.

2. Norhaidah A. H., Nurdatillah H. PHP Frameworks Usability in Web Application Development. International Journal of Recent Technology and Engineering (IJRTE). 2019. Vol. 8, no. 3S. P. 109–116.

3. Prokofyevaa N., Boltunovaa V. Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems. Procedia Computer Science : Conference, Riga, 15 December 2017. Riga, 2017. P. 51–56.

4. Tungadi E. Generator Formulir Memanfaatkan MySQL Metadata. Seminar Nasional Komunikasi dan Informatika 2015 : Conference, Makassar, 11 June 2005.

5. Ali J. Mastering PHP Design Patterns. Packt Publishing - ebooks Account, 2016. 270 p.

6. Livraghi M. AUTOMATIC GENERATION OF WEB CRUD APPLICATIONS : Магістерська. Мілан, 2015. 113 p. URL: https://www.politesi.polimi.it/handle/10589/125742 (date of access: 04.02.2024).

7. Hewage D. Service oriented code generator for fast prototyping:based on requirement definition schema : Магістерська. Moratuwa, 2017. 79 p. URL: http://dl.lib.uom.lk/handle/123/12932 (date of access: 04.02.2024).

8. Ruben M. PHP Development Frameworks THE QUEST FOR THE HOLY GRAIL : магістерська. Barcarena, 2020. 50 p. URL: https://repositorio-cientifico.uatlantica.pt/handle/10884/1566 (date of access: 05.02.2024).

9. Jone S. Comparing Performance of Plain PHP and Four of Its Popular Frameworks : магістерська. 2015. 33 p. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-45691 (date of access: 05.02.2024).

10. Asyraf W. Revisiting Web Application Development with Integrated Records Management Important Aspect using Re-CRUD : Магістерська. Kuala

Lumpur, 2022. 54 p. URL: https://ir.uitm.edu.my/id/eprint/65774/ (date of access: 05.02.2024).

11. Gutmans A. file_put_contents – Write data to a file. php documentation. URL: https://www.php.net/manual/en/function.file-put-contents.php (дата доступу: 08.08.2023).

12. Gutmans A. file_get_contents – Reads entire file into a string. php documentaion. URL: https://www.php.net/manual/en/function.file-get-contents.php (date of access: 08.08.2023).

13. Ellis R. Database Metadata. Codeigniter Documentation. URL: https://codeigniter.com/user_guide/database/metadata.html?highlight=getforeignkeydata (date of access: 08.08.2023).

14. Полякова Н. П., Мельник Д. О. Створення власного CRUD - генератора на PHP - фреймворку Codeigniter на основі аналізу бібліотек генерації коду фреймворків Yii2 та Laravel. МАТЕРІАЛИ III ВСЕУКРАЇНСЬКОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ ЗА УЧАСТЮ МОЛОДИХ НАУКОВЦІВ «АКТУАЛЬНІ ПИТАННЯ СТАЛОГО НАУКОВО-ТЕХНІЧНОГО ТА СОЦІАЛЬНО-ЕКОНОМІЧНОГО РОЗВИТКУ РЕГІОНІВ УКРАЇНИ» Запоріжжя: ЗНУ, 2023. С. 139-