

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ PHP ФРЕЙМВОРКУ
ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ВЕБЗАСТОСУНКІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF PHP
FRAMEWORK TO INCREASE THE EFFICIENCY OF WEB APPLICATION
DEVELOPMENT

Виконав: студент 2 курсу, групи КНТ-120м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Захряпа Р.А.

(прізвище та ініціали)

Керівник Зеленьова І.Я.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Захряпи Романа Артуровича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація PHP фреймворку для підвищення ефективності розробки вебзастосунків. Research and Software Implementation of PHP framework to Increase the Efficiency of Web Application Development

керівник проєкту (роботи) Зеленьова Ірина Яківна, к.т.н., доцент,

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

затверджені наказом закладу вищої освіти від “05” листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) 1 грудня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка Архітектури фреймворку. 3. Особливості тестування та експлуатації системи. 4. Керівництво програміста. Додатки

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Зеленьова І.Я., к.т.н., доцент		
Нормоконтроль	Бєлова А.В., асистент		

7. Дата видачі завдання “ 04 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання
2	Опис предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури фреймворку	4-5 тиждень	Розділ 2
4	Розробка та тестування фреймворку	6-9 тижні	Розділ 3
6	Створення керівництва програміста	10 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї	11 тиждень	ПЗ, Додатки
8	Нормоконтроль та рецензування	12 тиждень	
9	Захист роботи	13 тиждень	

Студент

(підпис)

Захряпа Р.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Зеленьова І.Я.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 132 с., 42 рис., 32 табл., 2 дод., 66 джерел.

ФРЕЙМВОРК, ПЛАТФОРМА, ПРОДУКТИВНІСТЬ, ЕФЕКТИВНІСТЬ, ВЕБЗАСТОСУНОК, BOOTSTRAP, CSS, HTML, JS, PHP, MYSQL

Об'єкт дослідження – засоби підвищення ефективності розробки вебзастосунків.

Предмет дослідження – фреймворки з використанням останніх технологій та мови програмування PHP.

Метою виконання дипломної кваліфікаційної роботи магістра є пришвидшення процесу розробки вебзастосунків шляхом використання шаблонізатора.

Для реалізації програмного продукту використовувалися наступні програмні засоби: об'єктно-орієнтована мова програмування PHP, PHP версії 7.4, середовище розробки PhpStorm та веббраузер Chrome. Для прискорення та полегшення розробки була використана розподілена система керування версіями Git.

Результати. Створено фреймворк з використанням мови програмування PHP, який завдяки шаблонізатору, значно пришвидшує створення виглядів, а також великої кількості компонентів, що надають методи по обробці вхідних даних.

Галузь використання – розробка вебзастосунків, фреймворк для створення вебзастосунків.

ABSTRACT

Explanatory note for master's graduation qualifying work: 132 pages, 42 figures, 32 tables, 2 appendixes, 66 sources.

FRAMEWORK, PLATRFORM, PRODUCTIVITY, EFFICIENCY, WEB-APPLICATION, BOOTSTRAP, CSS, HTML, JS, PHP, MYSQL

The target of the study is tools to improve the efficiency of web application development.

The subject of research is frameworks using the latest technology and the programming language PHP.

The purpose of the master's thesis is to accelerate the development of web applications through the use of a templating tool.

The purpose of the master's thesis is to accelerate the development of web applications through the use of a templating tool.oriented programming language PHP, PHP ver. 7.4, development environment PhpStorm and web-browser Chrome. To accelerate and facilitate the development of a distributed version control system Git.

The results. The framework was created with the PHP programming language, and which greatly accelerate the creation of views due to the templating engine, as well as a large number of components that provide methods for processing input data.

The scope of use. Web application development, frameworks for creating web applications.

ЗМІСТ

С.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	11
1.1 Основні поняття	11
1.2 Огляд та порівняння існуючих фреймворків	18
1.3 Обґрунтування вибору мови програмування та середовища розробки	29
1.4 Аналіз основних інструментів розробки	36
1.5 Визначення завдання роботи	44
1.6 Висновки за розділом 1	46
2 РОЗРОБКА АРХІТЕКТУРИ ФРЕЙМВОРКУ	47
2.1 Структура та архітектура проєкту.....	47
2.2 Класи системи	52
2.2.1 Основні класи системи	52
2.2.2 Допоміжні класи системи	64
2.3 Основні розроблені алгоритми	72
2.4 Висновки за розділом 2	76
3 ОСОБЛИВОСТІ ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ	77
3.1 Система контролю версій.....	77
3.2 Особливості інтерфейсу системи	79
3.3 Експлуатаційні вимоги	80
3.4 Тестування системи та аналіз результатів.....	80
3.5 Висновки за розділом 3	87
4 КЕРІВНИЦТВО ПРОГРАМІСТА.....	88
4.1 Системні вимоги до персонального комп'ютера для підтримки чиподальшої розробки програмного продукту	88

4.2	Налаштування системи перед початком її роботи.....	88
4.3	Звертання до програми	90
4.4	Висновки за розділом 4	98
ВИСНОВКИ		99
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ		101
ДОДАТОК А Текст програми		107
ДОДАТОК Б Слайди презентації.....		122

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API	– Application Program Interface;
CLI	– command-line interface;
CSS	– Cascading Style Sheets;
DSL	– Domain Specific Language;
IDE	– Integrated development environment;
JS	– JavaScript;
MVC	– model-view-controller;
ORM	– object relational mapping
PHP	– Hypertext Preprocessor, Personal Home Page;
SQL	– Structured Query Language;
БД	– база даних;
ООП	– об'єктно-орієнтоване програмування;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер;
ПП	– програмний продукт.

ВСТУП

В наш час комп'ютерні технології, а зокрема вебтехнології, розвиваються надзвичайно швидкими темпами. Комп'ютери набули широкого використання у багатьох сферах діяльності людини. Їх використовують як для роботи (ведення бізнесу), так і для приємного дозвілля.

Через обумовлену пандемією COVID-19, майже у всьому світі немає нагоди вийти на вулицю та придбати собі якийсь товар так, як карантин введений майже у кожній країні світу. Потрібність онлайн магазинів, де можна продивитися та замовити будь-який товар з не виходячи з дому, установ що потребують запису для відвідування зростає неймовірними темпами, як зросла і популярність розробки вебзастосунків.

Саме через те, що на сьогодні PHP (Hypertext Preprocessor) має долю у 78,8% (відсотків) усіх вебзастосунків інтернету, саме її було обрано у якості мови програмування для створення фреймворку. В даний час PHP, як і раніше, займає лідируючі позиції у світі веброзробки. Значна частина всіх вебзастосунків працює на цій мові програмування. Проста у використанні CMS (Content Management System) Wordpress продовжує набирати популярність. CMS Wordpress дозволяє користувачам створювати й розгортати свої власні вебзастосунки на безплатний або платних хостингових платформах без додаткових зусиль. Ось чому найближче майбутнє мови PHP здається яскравим, оскільки PHP продовжує розвиватися і широко використовуватися [1]-[3].

Потреба написання власного фреймворка з використання ООП мови PHP виражалася в тому, що необхідно було не тільки мінімізувати вихідний код на користь швидкості його виконання, але й значно пришвидшити створення виглядів завдяки шаблонізатору, а також реалізувати компоненти та їх методи з обробки вхідних даних. Так само була мета створити тільки необхідні базові компоненти для того, щоб зменшити час завантаження й обробки запитів. На даний момент PHP-фреймворки мають великий попит на сучасному ринку

інформаційних технологій, а значить існує і «попит» на розробників, які їх створюють. Отже, створення фреймворку є актуальною проблемою не тільки в описаній ситуації, яка виникла, але й загалом.

Для ефективної розробки вебзастосунків використовують фреймворки, що дозволяють зменшити час розробки та покращити якість коду. Фреймворк, який буде розроблений за допомогою мови програмування РНР та призначений для створення вебзастосунків, дозволить розробникам почати використовувати усі можливості фреймворка, що надасть більшої привабливості, ніж інші аналоги, та демонструватиме хороші показники як у простоті використання, так і у швидкості виконання написаного коду.

Перший розділ описує предметну область.

Другий розділ призначений для обґрунтування архітектури фреймворку.

Третій розділ описує основні рішення з реалізації системи та її функціонування з описаними технологіями.

Четвертий розділ надає інформацію по взаємодії програміста з розробленим фреймворком для подальшого використання.

Метою виконання дипломної кваліфікаційної роботи магістра є пришвидшення процесу розробки вебзастосунків шляхом зменшення кількості рядків коду та швидкості виконання коду завдяки наданню великої кількості компонентів з методами по обробці вхідних даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Вебзастосунок (іноді називають вебдодаток) – розподілений застосунок, в якому клієнтом виступає браузер, а сервером – вебсервер. Браузер може бути реалізацією так званих тонких клієнтів – логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у зображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому вебзастосунки є міжплатформовими сервісами. Внаслідок цієї універсальності й відносної простоти розробки вебзастосунки стали широко популярними в кінці 1990-х – початку 2000-х років [4].

Фреймворк (від англ. Framework, каркас, платформа, структура, інфраструктура) – це платформа для розробки програмних додатків. Вона забезпечує основу, на якій розробники програмного забезпечення можуть створювати програми для конкретної платформи. Наприклад, фреймворк може включати зумовлені класи і функції, які можуть використовуватися для обробки введення, управління апаратними пристроями і взаємодії з системним програмним забезпеченням. Це спрощує процес розробки, оскільки програмістам не потрібно винаходити колесо кожен раз, коли вони розробляють новий додаток. Фреймворк може також включати бібліотеки коду, компілятор і інші програми, які використовуються в процесі розробки програмного забезпечення. Якщо звернутися до історії самого слова «фреймворк», то цей неологізм з'явився в мові відносно недавно, приблизно на початку XXI століття [5]-[6].

Розробники програмних фреймворків прагнуть полегшити розробку програмного забезпечення, дозволяючи дизайнерам і програмістам присвятити свій час задоволенню вимог програмного забезпечення (ПЗ) замість того, щоб займатися більш стандартними низькорівневими деталями створення робочої

системи, тим самим скорочуючи загальний час розробки. Наприклад, команда, яка використовує вебфреймворк для розробки банківського вебзастосунку, може зосередитися на написанні коду, специфічного для банківської діяльності, а не на механіці обробки запитів і питаннях управління [7].

Фреймворк відрізняється від поняття бібліотеки тим, що бібліотека може бути використана в програмному продукті просто як набір підпрограм близькою функціональності, не впливаючи на архітектуру програмного продукту і не накладаючи на неї ніяких обмежень. У той час як фреймворк диктує правила побудови архітектури додатку, задаючи на початковому етапі розробки поведінку за замовчуванням – фреймворк, який потрібно буде розширювати й змінювати відповідно до зазначених вимог. Приклад програмного фреймворка – С.М.Ф. (Content Management Framework), а приклад бібліотеки – модуль електронної пошти. Також, на відміну від бібліотеки, яка об'єднує в собі набір близької функціональності, фреймворк може містити в собі велику кількість різних за тематикою бібліотек [8].

Каркасний підхід – це конфігурація програми, що будується з двох частин [9]:

- постійна частина – каркас (фреймворк), не змінний від конфігурації до конфігурації й несе в собі гнізда (місця під змінні модулі), в яких розміщується друга змінна частина;
- змінні модулі (точки розширення) можуть містити гнізда. В цьому випадку, поряд з каркасом програми в цілому, можна говорити й про каркас змінного модуля [9].

CSRF (від анг. cross-site request forgery) – підробка міжсайтових запитів, також відома як атака в один клік або «session riding» і скорочено CSRF або XSRF – це тип шкідливого використання вебзастосунку, при якому несанкціоновані команди подаються від користувача, якому вебзастосунок довіряє [10].

Composer – менеджер керування залежностями. Composer запускається з командного рядка і встановлює залежності (наприклад, бібліотеки) для

програми. Він також дозволяє користувачам встановлювати PHP-додатки, доступні в «Packagist», який є його основним репозиторієм, що містить доступні пакети. Він також надає можливості автозавантаження для бібліотек, в яких вказана інформація про автозавантаження, щоб полегшити використання стороннього коду.

Архітектурний шаблон проєктування MVC (Model View Controller) – це шаблон в проєктуванні програмного забезпечення, що часто використовується для реалізації користувацьких інтерфейсів, даних і керуючої логіки. Програма складається з моделі даних, контролера та подання. Цей шаблон вимагає, щоб кожна частина була розділена на різні об'єкти [11].

ORM (від англ. Object-Relational Mapping, Об'єктно-реляційна проєкція) – це метод, який дозволяє запитувати дані з бази даних і управляти ними з використанням об'єктно-орієнтованої парадигми [12].

В інженерії програмного забезпечення міграція схем (також міграція бази даних, управління змінами бази даних) – це керування інкрементними, зворотніми змінами схеми реляційної бази даних. Міграція схеми виконується в базі даних, коли виникає необхідність додавання нових таблиць і полів у БД або ж повернення до попередньої версії БД (користувач може вибрати на який момент часу необхідно повернути схему БД) [13].

Аутентифікація – це процес встановлення того, що хтось дійсно є тим, ким він себе видає, шляхом перевірки пароля та логіна користувача у системі [14].

Авторизація – належати до правил, які визначають, кому що дозволено, чи може користувач переглядати, редагувати, видаляти чи створювати записи [14].

PDO – це аббревіатура для об'єктів даних PHP (PHP Data Objects). PDO – простий і послідовний спосіб доступу до баз даних. Це означає, що розробники можуть писати зрозумілий код для багаторазового використання. PDO – це не рівень абстракції, а скоріш за все уніфікований API (Application Programming Interface) [15].

Прикладний програмний інтерфейс (від англ. Application Programming Interface, API) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. API є типом програмного інтерфейсу, що пропонує послуги іншим програмам. Документ або стандарт, що описує, як створити з'єднання або інтерфейс, називається специфікацією API. Вважається, що комп'ютерна система, що відповідає цьому стандарту, реалізує або надає API. Термін API може посилатись на реалізацію або на специфікацію [16].

Предметно-орієнтована мова програмування (англ. Domain-specific language, DSL) призначена для вирішення задач конкретної предметної галузі, яка дає особливе уявлення про проблему або специфічні методи її вирішення [17].

Пакет – це незалежний сторонній або вбудований модуль (частина, що додає нові функції).

HMVC (англ. Hierarchical model–view–controller) – ієрархічні модель-вигляд-контролер, одне з розширень архітектурного шаблон MVC, що дозволяє вирішити деякі проблеми масштабованості програм, що мають класичну MVC-архітектуру. Контролер має певну систему нагляду, так він спочатку вибирає модель, а потім вид, реалізуючи механізм затвердження контролером. Модель запобігає прямому доступу вигляда до даних моделі. Найбільшою практичною перевагою використання архітектури HMVC є віджет структур вмісту. Прикладом можуть бути коментарі, оцінки, відображення RSS-каналів у Twitter, або блозі, або відображення вмісту кошика для покупок у вебзастосунку електронної комерції. По суті, це частина вмісту, яка повинна відображатися на кількох сторінках, а можливо, навіть у різних місцях [18].

Одна з речей, з якими ви будете боротися в MVC – це те, що ви не можете викликати контролер з іншого контролера (якщо ви не використовуєте для цього власну бібліотеку), але у HMVC ви можете це зробити.

Переваги HMVC над MVC [11], [18]:

- повторно використовувати свій код в інших проєктах;
- спростити підтримку вашого проєкту;
- масштабувати свій проєкт;
- організувати код;
- модульна структура.

HTML (англ. HyperText Markup Language – мова розмітки гіпертексту) – це мова тегів, засобами якої здійснюється розмічання вебсторінок для мережі Інтернет. Браузери отримують HTML-документи з вебсервера, або з локальної пам'яті і передають документи в мультимедійні вебсторінки [19].

Браузер, також переглядач, вебпереглядач, вебоглядач, вебнавігатор (англ. Browser) – програмне забезпечення для комп'ютера або іншого електронного пристрою, як правило, під'єданого до Інтернету, що дає можливість користувачеві взаємодіяти з текстом, малюнками або іншою інформацією на гіпертекстовій вебсторінці. Тексти та малюнки можуть містити посилання на інші вебсторінки, розташовані на тому ж вебсайті або на інших вебсайтах. Вебпереглядач з допомогою гіперпосилань дозволяє користувачеві швидко та просто отримувати інформацію, розміщену на багатьох вебсторінках [20].

Вебсервер (англ. Web Server) – це сервер, що приймає HTTP-запити від клієнтів, зазвичай веббраузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потокм або іншими даними [21]. Вебсервер – одна з основ Всесвітньої павутини.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від HyperText Transfer Protocol, протокол передачі гіпертекстових документів [22].

Гіпертекст (англ. Hypertext) – текст для перегляду на комп'ютері, який містить зв'язки з іншими документами («гіперзв'язки» чи «гіперпосилання»); читач має змогу перейти до пов'язаних документів безпосередньо з вихідного (первинного) тексту, активувавши посилання. Найпопулярнішим зразком

гіпертексту є World Wide Web, у якому вебоглядач переміщує користувача з одного документу на інший, щойно той «натисне» на гіперпосилання [23].

CSS (англ. Cascading Style Sheets, укр. Каскадні таблиці стилів) – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних [24].

У більшості корпоративних кіл слова «продуктивність» і «ефективність» використовуються як синоніми. І хоча це може бути правильним в розмовній мові, це неправильно з технічної точки зору.

Продуктивність – це досягнення найкращого результату в будь-якій ситуації. Компанії не стають більш продуктивними через зростання їх бюджетів або термінів. Компанія стає більш продуктивною, тому що вони зосереджуються на тому, щоб робити більше, використовуючи свої поточні ресурси. Незалежно від того, великий бюджет або малий, люди, які зосереджені на продуктивності, більше дбають про те, щоб зробити все, що вони можуть, в порівнянні з тим, що у них є на поточний момент.

Припустимо, ваша компанія може обробляти 40 документів на годину. Компанія, яка орієнтована на продуктивність, спробує збільшити кількість документів, які вони можуть підготувати за цей час.

Отже, ваші співробітники багато працювали і тепер можуть обробляти в цілому 50 документів на годину, що робить їх продуктивніше на 25% (відсотків).

Незалежно від того, чи у співробітників компанії є шість годин, або одна година, співробітники компанії, орієнтовані на продуктивність, можуть працювати в тому ж темпі. Співробітники все ще виробляють стільки, скільки можуть, навіть якщо їх загальна кількість зменшується через скорочення.

Ефективність більше залежить від виживання. Або, принаймні, ефективність – це спроба виконати таку саму роботу з меншими ресурсами.

Скажімо, ваша компанія, яка тепер може обробляти 50 документів на годину, скорочує бюджет. Тепер компанія намагається виконати такий же обсяг роботи (50 документів обробляється за годину) з меншою кількістю ресурсів

для їх обробки. Це якраз і була б ця ситуація, коли необхідно сконцентруватися на ефективності.

В ідеальному світі компанії намагаються бути ефективними та продуктивними. Компанії завжди повинні шукати способи зробити більше з тим, що мають, одночасно виконуючи таку ж роботу з меншою кількістю зайвих витрат (час, ресурси, робота та матеріали).

При цьому більшість успішних компаній зосереджують свою увагу на продуктивності, а не на ефективності.

Продуктивність допомагає компаніям робити більше, тоді як ефективність допомагає компаніям робити те ж саме з меншими витратами.

Компанії дійсно успішні тільки тоді, коли вони зростають, що також відомо як отримання прибутку.

І це зростання, або збільшення прибутку, вимагає більшої продуктивності.

Якщо компанія збирається зосередитися на тому, щоб зробити щонебудь більш ефективним, то для цього необхідно буде виконати багато нудної роботи, але насправді це не допоможе компанії зрости.

Компанія буде ефективна, якщо здатна досягти бажаних результатів з невеликими витратами ресурсів. Якщо ви в змозі виконати однаковий обсяг роботи з меншим часом і ресурсами, ви будете більш ефективними. Якщо ви витрачаєте більше часу та ресурсів, але виконуєте той самий обсяг роботи, що і раніше, ваша продуктивність буде менш ефективною. Тому ефективність – це просто якість та ступінь ККД (коефіцієнт корисної дії), за якими компанія працює.

Ті, хто стурбовані продуктивністю, намагаються отримати максимальний результат за допомогою наявних у них ресурсів. Наприклад, якщо у них великий бюджет, вони намагаються створити максимально можливий обсяг роботи. Якщо у них невеликий бюджет, вони все ще намагаються створити максимально можливий обсяг роботи.

Розробка програмного забезпечення – трудомісткий процес. Значна частина його успіху залежить від продуктивності та ефективності розробників. Вимірювання правильних показників допомагає оцінити продуктивність і завчасно виявити червоні прапори, тож ви зможете підтримувати високу продуктивність та ефективність для розробки продуктів.

Наприклад, використовуючи шаблонізатор, що скорочує кількість строк коду для перевірки певної умови підвищує як ефективність, так і продуктивність. Ефективність – було покращено завдяки зменшенню затрат (у даному прикладі – часу). Продуктивність – завдяки використанню шаблонізатора, з'являється більше часу для написання коду більшого за об'ємом.

Git – розподілена система керування версіями файлів та спільної роботи.

Консоль (англ. console) – пристрій, який забезпечує взаємодію оператора комп'ютера з операційною системою, або будь-яким застосунком.

Процес перевірки називають валідацією, а дані, що відповідають вимогам – валідними.

xDebug – це розширення у браузері Chrome, що дозволяє використовувати точки зупину (від англ. breakpoint).

1.2 Огляд та порівняння існуючих фреймворків

У даному розділі розглянуті чотири найбільш популярні фреймворки, які написані на мові програмування PHP:

- Yii – фреймворк, назва якого розшифровується як «Yes, it is! », вже існує як 13 років і постійно оновлюється. У нього широкі можливості: одна з найвищих швидкість роботи (в порівнянні з іншими фреймворками), кешування, обробка помилок, міграція БД (баз даних);

– Laravel – цей фреймворк часто лідирує в різноманітних опитуваннях, що стосуються PHP-фреймворків. Наприклад, в 2013 році Laravel був названий самим перспективним проєктом 2014 року, а в 2015 році зайняв перші місця в категоріях «Фреймворк корпоративного рівня» і «Фреймворк для особистих проєктів». Також Laravel простий в освоєнні;

– Symfony – цей фреймворк часто рекомендують використовувати для створення великих порталів, бо його можна назвати одним з найбільш стабільних PHP-фреймворків. Це гнучкий і масштабований фреймворк зі значним функціоналом. Symfony містить корисні багаторазові компоненти, що стосуються безпеки, шаблонів, перекладу, налаштуванні форм і багато чого іншого;

– CodeIgniter – один з найстаріших фреймворків, перший публічний реліз якого відбувся у 2006 році. Цей фреймворк має масу переваг: швидка установка, хороша документація, мала вага, з ним ви можете легко реалізувати задуманий проєкт.

Для того, щоб обрати найбільш придатні частини з кожного з фреймворків, вони будуть детально проаналізовані.

Фреймворк «Yii» – це простий, але високопродуктивний універсальний фреймворк, заснований на компонентній структурі, також фреймворк використовує шаблон проєктування MVC (Model View Controller). Він відомий завдяки високій продуктивності та своєю простотою. Історія Yii розпочалася 1 січня 2008 як проєкт щодо виправлення деяких вад у фреймворку PRADO (PHP Rapid Application Development Object-Oriented), що став в 2004 переможцем «Zend PHP 5 coding contest». Автором фреймворку є Цян Сун. На рисунку 1.1 зображено зовнішній вигляд фреймворку «Yii» та Gii меню (що дозволяє створювати контролери, моделі та вигляду) [25].

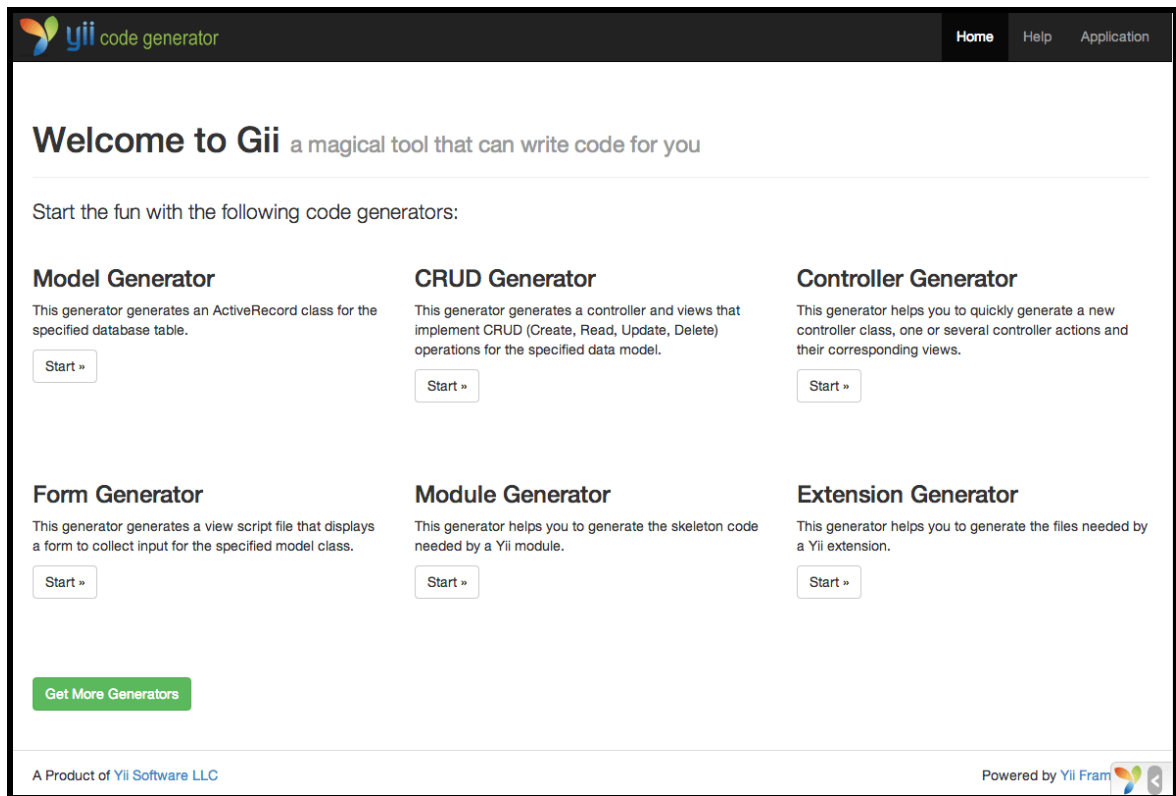


Рисунок 1.1 – Фреймворк «Yii» та Gii меню

Yii дозволяє використовувати код сторонніх розробників, а наявний в ньому генератор коду Gii дозволяє швидко створювати базові структури, на основі яких можна будувати власні рішення. Розробники цього фреймворку дуже серйозно підійшли до питань безпеки. Зокрема, в нього вбудовані механізми хешування паролів, заснованих на bcrypt, та засоби шифрування.

Переваги Yii:

- є платформою з відкритим вихідним кодом, що надає розробникам найшвидший варіант для створення великомасштабних вебзастосунків;
- MVC архітектура дозволяє спростити розробку, є дуже розповсюдженою та робить код більш зрозумілим для подальшого розширення;
- використовує «Composer» для управління залежностями;
- пропонує набір високоінтуїтивних і розширюючих інструментів для швидкої генерації коду за допомогою такої функції як CRUD ;
- надає декілька варіантів кешування відображень (у вайлах та у базі даних, Memcache);

- має дуже зручний обробник помилок (виводить рядки коду та рядок, де було допущено помилку), також може запропонувати знайти помилку у мережі інтернет;
- налагодження є дуже інформативним (відображаються усі запити до БД, використані моделі, поточну конфігурацію, використані бібліотеки, час виконання запитів, кількість логів у системі, поточну сторінку навігації, версію фреймворку, версію PHP, кількість використаних CSS та JS файлів);
- надає можливості завантаження документації у декількох форматах (pdf, html);
- Yii також супроводжується широким спектром заходів безпеки, які здатні захистити ваші сайти від таких атак, як SQL (Structured Query Language) ін'єкції, підробка cookie, підробка міжсайтових запитів (CSRF) і міжсайтовий скриптинг (CSS).

Недоліки Yii:

- через архітектуру MVC трохи збільшується час виконання;
- вимагає попередніх знань про сторонні модулі та їх використання, встановлення;
- останнім часом дуже рідко отримує нові версії, через що злоумисники можуть скористатися цим та знайти вразливості;
- функції Ajax недостатньо добре опрацьовані та іноді вимагають додаткового написання JavaScript коду.

У Yii2 використовується пакетний менеджер Composer для управління залежностями. Завдяки лінивому завантаженню Yii2 вважається одним із найшвидших PHP-фреймворків.

Laravel – це безплатний PHP-фреймворк з відкритим кодом, який був створений Тейлором Отвелом у 2011 році. Відтоді він зміг стати тим фреймворком, який найчастіше вибирають веброзробники. Одна з головних причин цього полягає в тому, що Laravel дозволяє розробляти складні

вебзастосунки простіше, швидше і безпечніше, ніж будь-які інші фреймворки [26].

Вітальна сторінка фреймворку Laravel (рис. 1.2).

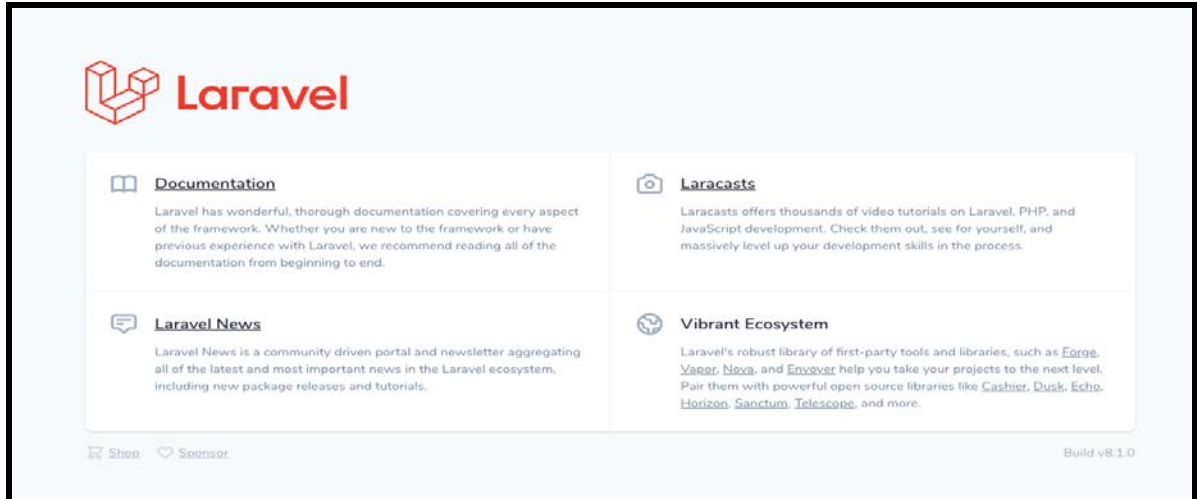


Рисунок 1.2 – Вітальна сторінка фреймворку «Laravel»

Крім того, Laravel полегшує вирішення багатьох великих завдань, що веде до того, що розробникам доводиться писати, у відповідних ситуаціях, менше коду. Серед таких завдань – маршрутизація, кешування, авторизація, аутентифікація. Завдяки використанню цього фреймворка спрощуються такі завдання, як перевірка електронних адрес користувачів, хешування і скидання паролів. Laravel вибирають, в основному, розробники B2B (business to business) рішень. А після появи Homestead – заздалегідь підготовленого контейнера Vagrant – установка Laravel стала надзвичайно простою [27]-[28].

У цьому фреймворку застосовується шаблонізатор Blade. Він дозволяє використовувати код PHP в уявленнях. Blade практично не впливає на швидкість роботи додатків, тому що уявлення зберігаються в окремих файлах з розширенням «.blade.php». Код уявлень перетворюється в нативний PHP.

Переваги Laravel:

- одна з найзручніших систем управління та створення маршрутизації (навігації), що надає можливості давати назви певним сторінкам та посилатися на них з інших сторінок, використовуючи ім'я маршруту, яке дає користувач (розробник системи);
- Laravel Artisan – спеціальний інструмент Laravel, що поставляється з власним інтегрованим інтерфейсом командного рядка (дозволяє створювати власні команди, що будуть виконуватися з командного рядка та виконувати ті дії, на які запрограмує розробник; також дозволяє створювати моделі та контролери);
- MVC архітектура дозволяє спростити розробку, є дуже розповсюдженою та робить код більш зрозумілим для подальшого розширення;
- керування чергами значно пришвидшує відповідь від сервера до клієнта завдяки створенню черг з поставлених задач/команд;
- детальна документація;
- вбудована система аутентифікації та авторизації;
- легкість додавання сторонніх модулів;
- Eloquent ORM (Object–relational mapping) дозволяє створювати моделі з існуючих таблиць БД.

Недоліки Laravel:

- через архітектуру MVC трохи збільшується час виконання;
- відсутній модуль CRUD генерації;
- деякі оновлення можуть бути проблематичними;
- через використання шаблонізатора «blade.php» збільшується час створення вигляду;
- вимагає більше конфігурації та коду клею, ніж деякі інші фреймворки.

Для управління залежностями використовується раніше згаданий Composer.

Laravel Artisan поставляється з власним інтегрованим інтерфейсом командного рядка. Artisan дозволяє розробникам створювати міграції баз даних, `tinker` (взаємодія з базою даних). Він також контролює систему баз даних, тому розробникам не потрібно виконувати рутинні завдання програмування.

Symfony – це фреймворк, написаний за допомогою мови PHP, пропонує 50 багаторазових компонентів для створення вебзастосунків [29]. Він був опублікований як безплатне програмне забезпечення 18 жовтня 2005 року. Symfony прагне прискорити створення та обслуговування вебзастосунків та повторюваність коду. Він також спрямований на створення надійних програм у контексті підприємства та має на меті надати розробникам повний контроль над конфігурацією: від структури каталогів до сторонніх бібліотек, майже все можна налаштувати. Symfony комплектується додатковими інструментами, які допомагають розробникам тестувати, налагоджувати та документувати проекти. Вітальна сторінка фреймворку Symfony (рис. 1.3).

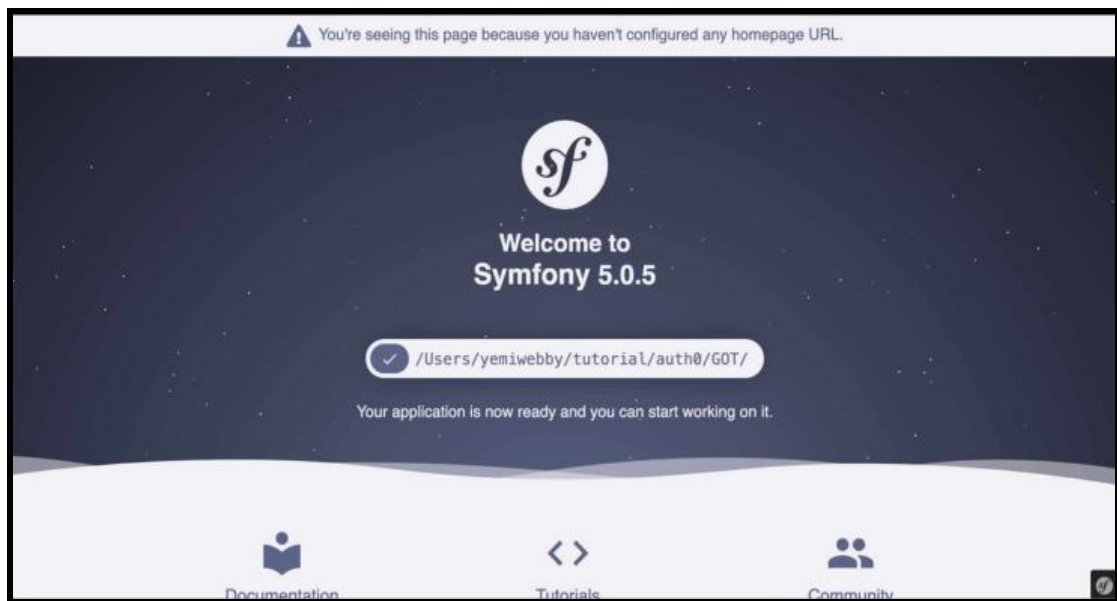


Рисунок 1.3 – Фреймворк «Symfony» та його вітальна сторінка

Doctrine ORM – це набір бібліотек PHP. Об'єктивно-реляційний проєктор (ORM) для PHP 5.3.0+, який базується на шарах абстрактних дій доступу до БД DBAL (Doctrine Database Abstraction Layer). В останній доктрині з ключовими можливостями доктрина робить записи до БД у власному

об'єктно-орієнтованому діалоговому вікні SQL з наявним DQL (Doctrine Query Language) і базується на ідеальних HQL (Hibernate Query Language) [30].

DBAL (Doctrine Database Abstraction Layer) – це рівень абстракції, який розташований поверх PDO та пропонує інтуїтивно зрозумілий та гнучкий API для спілкування з найпопулярнішими реляційними базами даних [30]. Profiler – інструмент для відладки та моніторингу швидкості роботи системи.

Переваги Twig:

- швидка копія шаблонів до простого оптимізованого коду PHP;
- безпечний, тому що має режим пісочниці для оцінки ненадійного коду шаблону;
- дозволяє створювати власні конструкції розробнику, визначаючи власні спеціальні теги та фільтри, та створювати власний DSL.

Переваги Symfony:

- вбудована можливість документувати проєкт (не кожен фреймворк може надати таку можливість, на рівні корпорації – це дуже важлива функція);
- зручний інструмент відладки Profiler;
- підтримка черг за допомогою вбудованого пакета Messenger;
- легко почати писати код;
- Symfony Flex;
- додаткові накладні витрати на компіляцію шаблонів Twig;
- пакети Symfony – це бібліотеки, які дуже легко завантажити та розпочати за лічені хвилини;
- структура каталогів за замовчуванням дуже абстрагована, не пов'язана і готова до складних проєктів, які вимагають великої гнучкості;
- використовує Doctrine ORM, яка може похвалитися неймовірною популярністю серед PHP- розробників.

Недоліки Symfony:

- продуктивність може бути проблемою, особливо у старих версіях;
- Doctrine ORM недостатньо добре задокументована;

- поставляється зі значною кількістю зовнішніх залежностей;
- через використання шаблонізатора «twig» збільшується час створення вигляду [31]-[33].

Twig – це сучасний шаблонізатор з розширенням «twig», що використовується фреймворком Symfony за замовчуванням [32].

Symfony Flex – додатковий інструмент, що допомагає в інтеграції сторонніх пакетів шляхом часткового налаштування.

CodeIgniter – це вебфреймворк для швидкої розробки програмного забезпечення з відкритим кодом для використання при створенні динамічних вебзастосунків за допомогою PHP [34].

Базується на популярній моделі розробки модель-вигляд-контролер (MVC). Хоча класи контролерів є необхідною частиною розробки в CodeIgniter, моделі та вигляди є необов’язковими. CodeIgniter також можна модифікувати для використання ієрархічної модель-вигляд-контролер (HMVC, Hierarchical model-view-controller), що дозволяє розробникам підтримувати модульне групування контролерів, моделей та подання у форматі підкаталога.

Вітальна сторінка фреймворку CodeIgniter (рис. 1.4).

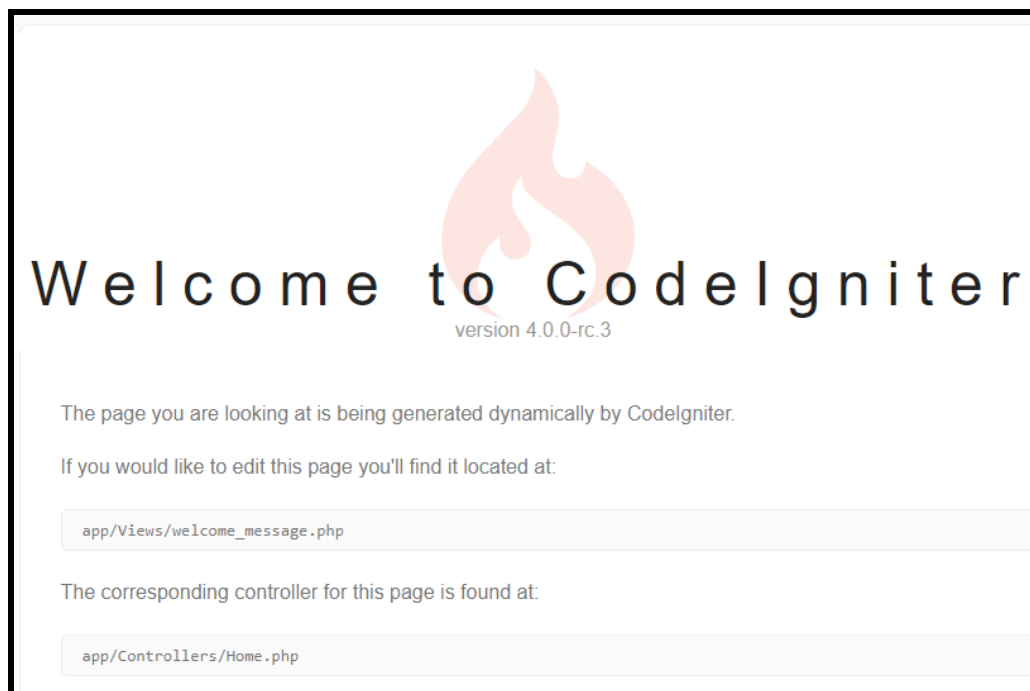


Рисунок 1.4 – Фреймворк «CodeIgniter» та його вітальна сторінка

Переваги CodeIgniter:

- можливість вибору між архітектурою MVC та HMVC;
- інформативне налагодження (що відображає усі запити до БД, використані моделі, поточну конфігурацію, використані бібліотеки, час виконання запитів);
- надає дуже зручний варіант по створенню валідації, що значно пришвидшує розробку та час на її створення.

Недоліки CodeIgniter:

- не підтримує модульне розділення коду за замовчуванням, що може ускладнити процес як розробки, так і підтримки;
- має менш варіантів кешування;
- відсутній модуль CRUD - генерації;
- відсутній модуль авторизації та аутентифікації за замовчуванням;
- має менш модулів функцій, ніж деякі інші фреймворки, через що може не підійти для великих проєктів;
- дещо складний у вивченні через не зовсім інтуїтивні назви методів та документацію;
- були випадки нестабільних версій у минулому [35].

Розроблений EllisLab, а також Ріком Еллісом (Rick Ellis) та Полом Бурдіком (Paul Burdick). 6 червня 2013 року EllisLab передав CodeIgniter технологічному інституту Британської Колумбії (Канада).

Використовуючи CodeIgniter, ви заощадите багато часу, якщо розробляєте вебзастосунок з самого початку. CodeIgniter також є безпечним, оскільки він здатний запобігати різноманітним атакам.

У цьому розділі було наведено 4 приклади систем управління конфігурацією серверів, але усі вони, окрім переваг, мають і недоліки.

Порівняння найважливіших характеристик різних фреймворків наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння PHP-фреймворків

Параметри порівняння	Yii 2.31	Laravel 8.41.0	Symfony 5.2.1	Codeigniter 4.0.4
Підтримуючі версії PHP	>=5.1 – 8.0	>=7.3-8.0	>=7.2.5 – 8.0	>=7.3
Документація і приклади	Детально описана та надає вичерпну кількість прикладів; є можливості завантаження документації	Надає велику кількість прикладів з детальним описом	Велика кількість прикладів, але вони не дуже добре описані	Присутня, але не достатньо детальна, мало прикладів; Часто сайт з документацією не працює
Жорстка структура каталогів	+	+	+	–
Офіційна підтримка інтернаціоналізації	+	+	+	+
Вбудований CRUD генератор	+	–	–	–
Складність встановлення	Складно	Легко	Помірна	Легко
Складність налаштування	Досить довго, через потребу в налаштуванні файлів .htaccess	Дуже легко, необхідно лише вказати декілька параметрів у одному файлі	Швидко та інтуїтивно	Іноді документація не надає опис усіх параметрів
Архітектура	MVC	MVC	MVC	HMVC
Тип ORM	ActiveRecord	Eloquent ORM	Doctrine ORM	На вибір розробника
Складність у використанні ORM	Не дуже зручно	Дуже зручно	Дуже зручно	–
Частота використання	9%	50%	24%	12%
Вбудований шаблонізатор	–	Blade	twig	–
Ліцензія	BSD	MIT	MIT	Apache
Фреймворк / бібліотека для стилів за замовчуванням	Модифікована версія Bootstrap 3, або стороння (на вибір розробника)	Модифікована версія Bootstrap 5 або стороння (на вибір розробника)	Надає можливості встановлення сторонніх стилів	Надає можливості встановлення сторонніх стилів

Після порівняння фреймворків було прийняте рішення, що необхідно створити шаблонізатор, що буде схожий на шаблонізатор фреймворку Laravel «.blade», також буде реалізована більш проста взаємодія з БД за допомогою ORM, та шаблон проектування ActiveRecord для взаємодії з БД. У якості основного архітектурного шаблону проектування буде обрано архітектурний шаблон MVC.

1.3 Обґрунтування вибору мови програмування та середовища розробки

Для написання програмного продукту було обрано об'єктно-орієнтовану мову програмування PHP. PHP Hypertext Preprocessor (спочатку Personal Home Page Tools, «Інструменти для створення персональних вебсторінок») – одна з провідних мов сучасної веброзробки [36].

Третя версія PHP увійшла в історію, саме вона визначила сучасний стиль і вигляд PHP. Перероблена ізраїльськими програмістами Енді Гутманс і Зеєвом Сураські версія PHP 3.0 офіційно вийшла влітку 1998 року.

Основна перевага PHP 3.0 – можливість підключення до ядра додаткових модулів для розширення функціоналу. Мова PHP набирала все більше прихильників. PHP підтримувала роботу з усіма популярними базами даних. В PHP 3.0 була можлива підтримка багатьох протоколів і різних API. Зростання популярності мови PHP сприяла її стрімкому розвитку. PHP стала розшифровуватись «hypertext Preprocessor» – гіпертекстовий препроцесор.

В кінці 1998 року ізраїльські програмісти почали розробляти новий каркас (фреймворк). Вони зробили акцент на збільшення продуктивності та вдосконалення модульності базису PHP-коду. Завдяки праці розробників в 1999 році народився каркас під назвою Zend Engine, який був на голову вище движка попередньої версії PHP. На основі Zend Engine до 2000 року офіційно була випущена мова PHP 4.0. У новій версії значно зросла продуктивність, присутній ряд корисних нововведень. У PHP 4.0 була включена підтримка сесій. З'явилася можливість буферизації відображень. Мова стала значно безпечнішою.

В середині липня 2004 року побачила світ п'ята версія PHP. PHP 5 працює на ядрі Zend Engine 2, що значно ефективніше і продуктивніше попередніх версій. Значним моментом в історії розвитку PHP було впровадження повноцінного об'єктно-орієнтованого програмування в PHP 5, що в рази спростило і зробило полегшало розробку масштабних проєктів на мові PHP.

Головна характеристика PHP – можливість інтерпретації. На відміну від Java, яка компілюється, а потім запускається в роботу, PHP створюється під час звернення до неї. Людина відкриває сайт, на сервер надсилається запит, і в цей час компілюється код. Кожен скрипт компілюється в реальному часі, а потім виконується.

PHP є однією з найпоширеніших мов, що використовуються у сфері веброзробок (разом із Java, .NET, Perl, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів [36].

Зараз остання версія інтерпретатора PHP – версія 8.1.

Об'єктно-орієнтоване програмування (ООП) – це програмування, орієнтоване на використання об'єктів в коді програми [37].

Основні парадигми ООП включають у себе спадкування, поліморфізм (багато форм) та інкапсуляцію (приховування даних).

Модифікатори доступу – це ключові слова, які задають оголошений рівень доступності для кожного члена класу, будь то метод чи поле.

Модифікатори доступу, який доступ та кому вони його надають:

- public (відкритий) – доступ відкритий всім;
- protected (захищений) – доступ відкритий для похідних класів, тобто лише похідні класи отримують вільний доступ до таких властивостей або методів;
- private (закритий) – доступ відкритий самому класу, тобто методи та поля даного класу, однак усі похідні та інші класи не отримують доступу до цих даних [38].

Порівняння швидкості роботи різноманітних версій PHP за допомогою «PHPBench» наведено у рисунку 1.5 [39].

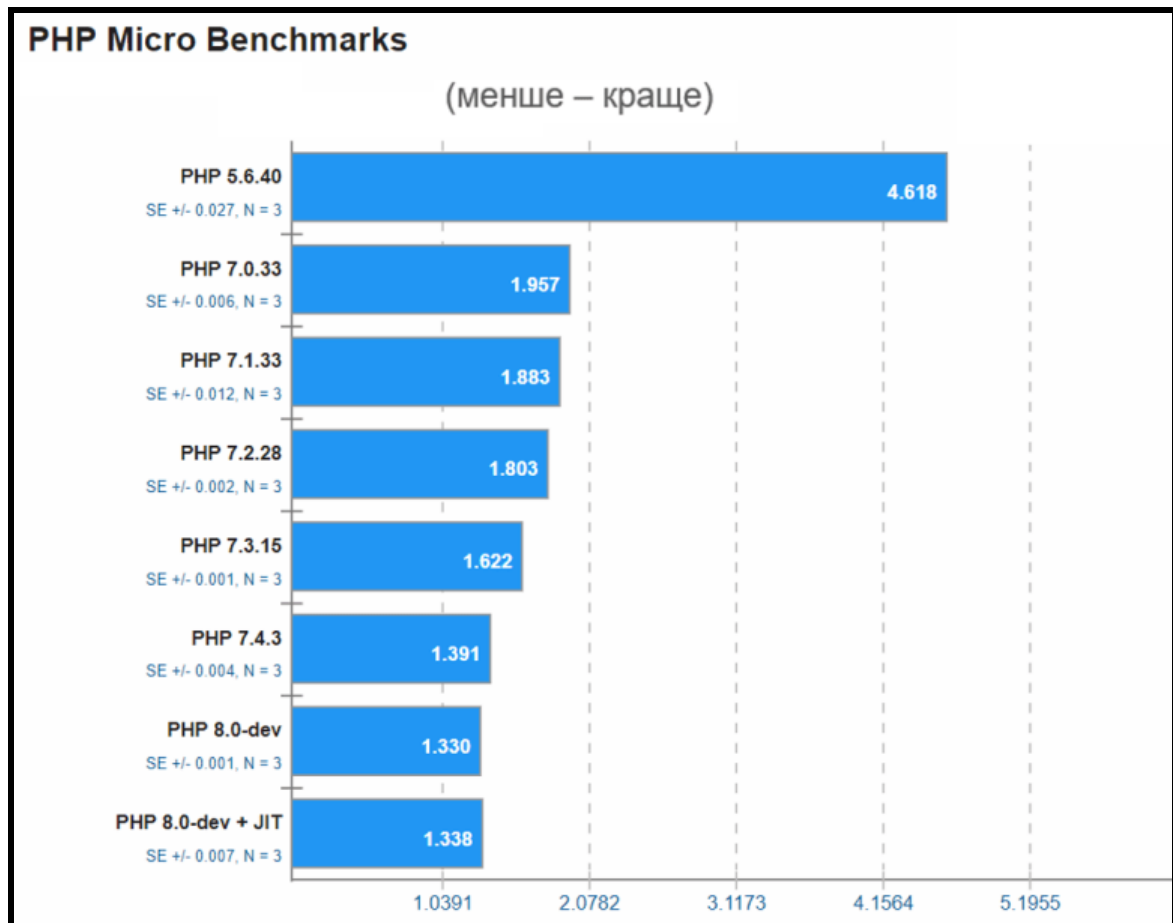


Рисунок 1.5 – Порівняння швидкості роботи різноманітних версій PHP за допомогою «PHPBench»

Об'єктно-орієнтоване програмування надає наступні переваги при розробці програмного забезпечення:

- зменшена схильність до помилок: об'єкт контролює доступ до своїх власних даних;
- об'єкт блокують помилкові спроби доступу до приватних полів;
- просте повторне використання: об'єкти підтримують себе, тому можуть використовуватися як будівні блоки для інших програм [37].

Для створення програмного продукту було прийнято рішення використовувати середовища розробки JetBrains PhpStorm – комерційне багатоплатформове інтегроване середовище розробки для PHP, яке розробляється російською компанією JetBrains на основі платформи IntelliJ IDEA.

IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безплатної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проєктів мають можливість отримати безплатну ліцензію [40].

Перша версія IntelliJ IDEA з'явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати сирцевий код (англ. source code, вихідний код) програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій [40].

PhpStorm – це інтегроване середовище розробки (анг. IDE, integrated development environment), що призначене для мови PHP та насправді «розуміє» ваш код. PhpStorm підтримує PHP 5.3/5.4/5.5/5.6/7.0/7.1/7.2/7.4/8.0/8.1, забезпечує запобігання помилкам на ходу, найкраще автозаповнення та рефакторинг коду, налагодження нульової конфігурації та розширений редактор HTML, CSS та JavaScript. Вперше середа розробки PhpStorm була випущена у 2009 році та написана за допомогою мови програмування Java. Остання версія на поточний момент 2021 року є 2021.2.3 [41].

Інтегроване середовище розробки (анг. IDE, integrated development environment) – це програмне забезпечення, яке надає комплексні можливості програмістам для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та налагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду [42].

Автодоповнення коду – це метод аналізу вже існуючого коду та надання користувачеві підказок щодо можливих варіантів написання коду та автоматичного додання вбудованих функцій та методів у саму мову PHP.

PhpStorm надає багатий і інтелектуальний редактор коду для PHP з підсвіткою синтаксису, розширеною конфігурацією форматування коду та перевіркою коду на наявність помилок у реальному часі.

Переваги PhpStorm 2021:

- підвищує продуктивність розробників, допомагаючи знайти помилки, виділяє некоректний або небажаний код;
- багатофункціональне середовище розробки як для пошуку, так і для перегляду файлів;
- велика кількість режимів перегляду;
- велика кількість налаштувань – від налаштування кольорів кожної конструкції до створення власних скорочень для пришвидшення написання коду;
- працює на будь-якій сучасній операційній системі;
- зберігає історію редагованих файлів (більше, ніж за пів року);
- додаткові інструменти для налагодження;
- вбудований командний рядок (анг. CLI – command-line interface);
- перевірка коду під час його написання;
- підтримка більшості існуючих фреймворків, написаних на мові PHP (Symfony, Yii, Laravel, CodeIgniter, Phalcon, CakePHP, Drupal);
- специфікація PSR1-12 призначена для стандартизації написаного коду;
- інтеграція зі сторонніми програмами з налагодження кода;
- багатоплатформність (Windows, macOS, Linux);
- індексація файлів для пришвидшення пошуку цих файлів [43].

Недоліки PhpStorm 2021:

- вартість ліцензії;
- завантаження проєкта може зайняти деякий час (у разі якщо проєкт має багато файлів);
- може потребувати багато оперативної пам'яті (ОП);

- не найкращим чином надається можливість працювання з декількома проєктами;
- для новачка може бути складним в опануванні [43]-[44].

Середовище розробки PhpStorm надає багато можливостей по створюванню та підтримці проєктів будь – якого розміру. Середовище надає розробнику дуже багато опцій з налаштуванням під себе, зміну оформлення, запуск PHPUnit тестів, аналіз коду, швидка навігація по методам, класам та файлам.

Середовище надає наступні можливості:

- перегляд файлів різного формату (від зображень до файлів у форматі json);
- детектор дубльованого коду;
- підтримка Vagrant, Composer, вбудований REST клієнт, CLI, SSH консоль;
- IDE забезпечує автодоповнення коду, підсвічування синтаксису, розширену конфігурацію форматування коду, перевірку помилок на ходу, згортання блоків коду. Автоматизований рефакторинг, який дбайливо ставляється до вашого коду, допомагаючи легко та безпечно вносити глобальні налаштування проєкту;
- перевірка правопису з можливістю виправлення;
- дуже зручний пошук файлів та каталогів проєкту;
- перейменування змінних по всьому проєкту або одному файлу;
- надає можливість підключення до БД та виконання запитів;
- багатий функціонал для пошуку коду (за допомогою регулярних виразів, перевірка регістра, повний збіг слів);
- перевірка залежностей в реальному часі і відображення вікна з підказкою про можливість установки нового пакета;
- надає дуже багато різноманітних режимів для перегляду, написання коду (режим презентації; режим мінімалістичний, повноекранний; режим дзен);

- при використанні контролю версій відображаються нові, змінені файли та рядки коду, що були видалені чи додані до кожного файлу;
- використовує широкий функціонал, додаткові гарячі комбінації, нові компоненти, що підтримуються та виправляються;
- перегляд декількох файлів одночасно, завдяки декільком робочих областей;
- підказка до кожного класу, методу, вбудованого у PHP, та до властивостей (у разі якщо вони були описані);
- аналіз коду, що може підказати яку частину коду можна прибрати чи замінити, застосовуючи менше дій;
- можливість змінити тему оформлення (від світлої до темної) [44].

Для комфортної роботи програми необхідно мати персональний комп'ютер з такими характеристиками:

- процесор Intel або AMD від 4 ядер;
- сучасна відеокарта, що підтримує роздільну здатність у 1920x1080 та відповідний монітор;
- 8 ГБ ОП;
- пристрої вводу, такі як миша та клавіатура;
- підтримуючі операційні системи (Microsoft Windows 8 або новіше, macOS 10.13 та новіше, будь-який дистрибутив Linux, що підтримує Gnome, KDE або Unity DE);
- як мінімум 5 ГБ вільного місця на твердотілому накопичувачі [45].

Все це робить середовище розробки PhpStorm найкращим рішенням для вирішення поставленої задачі.

1.4 Аналіз основних інструментів розробки

Composer – це менеджер залежностей. Він завантажує їх в локальний каталог (vendor), знаходить всі залежності у репозиторії і завантажує їх, а потім робить весь цей код доступним для вашого проєкту через автозавантажувач [46]-[47].

Більшість залежностей знаходиться у репозиторії «Packagist», який є його основним сховищем, де містяться доступні пакети. Він також реалізує автозавантажувач класів для встановлених бібліотек. Це полегшує використання коду як від сторонніх розробників, так і для розробника, що створює власний проєкт.

Composer був розроблений Нілом Адерманом і Хорді Боггіано, які й досі супроводжують проєкт. Вони почали розробку у квітні 2011 року і вперше випустили його 1 березня 2012 року [46].

Composer сильно натхненний «npm» з Node.js і «bundler» Ruby. Алгоритм рішення залежностей проєкту почався як порт на основі PHP satsolver libzypp openSUSE [46].

HTML (англ. HyperText Markup Language – мова розмітки гіпертексту) – це мова тегів, засобами якої здійснюється розмічання вебсторінок для мережі Інтернет. Браузери отримують HTML-документи з вебсервера, або з локальної пам'яті, і передають документи в мультимедійні вебсторінки [19].

Елементи HTML є будівельними блоками сторінок HTML. За допомогою конструкцій HTML зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у візуалізовану сторінку. HTML надає засоби для створення структурованих документів, позначаючи структурну семантику тексту, наприклад, заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML окреслені тегами, написаними з використанням кутових дужок. Теги на кшталт «» чи «<input />» безпосередньо виводять вміст на сторінку. Інші теги, такі як «<p>», оточують текст і надають інформацію про нього, а також можуть включати інші теги як піделементи.

Браузери не показують теги HTML, але використовують їх для інтерпретації вмісту сторінки [19].

В HTML можна вбудовувати програми, написані на скриптових мовах, наприклад JavaScript, які впливають на поведінку та вміст вебсторінок. Включення CSS визначає вигляд і компонування вмісту. World Wide Web Consortium (анг. W3C, Консорціум Всесвітнього павутиння), який супроводжує стандарти HTML та CSS, заохочує використання CSS над явним презентаційним HTML з 1997 року [19], [24].

HTML впроваджує засоби для:

- створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати;
- отримання інформації зі Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео та інших об'єктів до тексту.

Розмітка в HTML складається з чотирьох основних компонентів: елементів (та їхніх атрибутів), базових типів даних, символічних мнемонік та декларації типу документа [19].

Консорціум Всесвітнього павутиння (мережі) – є головною міжнародною організацією зі стандартизації для Всесвітнього павутиння. Заснований у 1994 році та наразі під керівництвом Тіма Бернерса-Лі, консорціум складається з організацій-членів, які мають штатний персонал, який працює разом над розробкою стандартів для Всесвітньої павутини. Станом на 21 жовтня 2019 року W3C налічувала 443 учасники. W3C також бере участь в освітній та просвітницькій діяльності, розробляє програмне забезпечення та слугує відкритим форумом для обговорення Інтернету [48].

CSS (англ. Cascading Style Sheets, укр. Каскадні таблиці стилів) – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних [24].

CSS є основною технологією всесвітньої павутини, поряд із HTML та JavaScript. Найчастіше CSS використовують для візуальної презентації

сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів [24].

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці вебсторінок. Головна перевага блочної верстки – розділення змісту сторінки (даних) та їхньої візуальної презентації [24].

CSS використовується авторами та відвідувачами вебсторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг – можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS) [24].

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерів) [24].

Мова структурованих запитів (SQL) використовується для маніпулювання, пошуку, вставлення, оновлення та видалення даних у системі управління реляційними базами даних (СУБД) [49].

Для спрощення, SQL повідомляє базі даних, що робити і що саме їй потрібно.

SQL – це стандартна мова для роботи з реляційною базою даних, яку використовують усі системи СУБД, такі як MySQL, MS Access, MS SQL, Oracle, Postgres та інші [50].

База даних (БД) – це структурована сукупність відомостей про об'єкти чи сутності конкретної предметної області та окремих стандартизованих засобів для автоматизації їх обробки [51].

Таблиця – це сукупність пов'язаних даних, що зберігаються у форматі таблиці в базі даних. Вона складається зі стовпців і рядків [51].

У реляційної БД – рядок також називають кортежем, який представляє з себе всього один непрямо структурований елемент даних, який знаходиться в таблиці. Таблиця БД складається з рядків та стовпців. Кожен рядок у таблиці це набір пов'язаних даних, і кожен рядок у таблиці має ту саму структуру, а стовпець зберігає у собі назву однієї з характеристик структури.

У реляційній базі даних стовпець (також атрибут) – це значення даних певного типу, для кожного рядка БД стовпець зберігає одне значення. Стовпець може містити: цифри, дати, текстові значення та булеві значення.

Модель реляційної БД стверджує, що дані зберігаються в одній або кількох таблицях. У цій моделі також зазначено, що кожную таблицю можна розглядати як двовимірну матрицю, що складається з рядків і стовпців.

На практиці рядки та стовпці таблиці реляційних БД часто згадуються більш традиційними термінами, записами та полями. Насправді, деякі пакети програмного забезпечення використовують один набір термінів, деякі – інші, а деякі – комбінацію.

Загалом, кожна таблиця моделюється за реальною сутністю, такою як постачальник або рахунок-фактура. Потім стовпці таблиці представляють такі атрибути сутності, як ім'я, адреса та номер телефону. І кожен рядок таблиці представляє один екземпляр сутності. Значення зберігається на перетині кожного рядка та стовпця, іноді називається коміркою.

Клієнт-сервер представляє з себе три апаратні системи [2]:

- клієнти – зазвичай це ПК, які вже є на робочих столах у всій компанії (клієнтами можуть бути мобільні пристрої, такі як ноутбуки, планшети та смартфони);
- мережа – це кабелі, лінії зв'язку, карти мережевого інтерфейсу, концентратори, маршрутизатори та інші компоненти, які з'єднують клієнтів та сервер;

– сервер – зазвичай називають сервером баз даних, – це комп'ютер, який має достатню швидкість процесора, внутрішню пам'ять (ОЗУ), дискову пам'ять для зберігання файлів, БД системи.

Коли ви використовуєте SQL для доступу до бази даних MySQL, ця система часто представляє з себе клієнт-сервер (рис. 1.6).

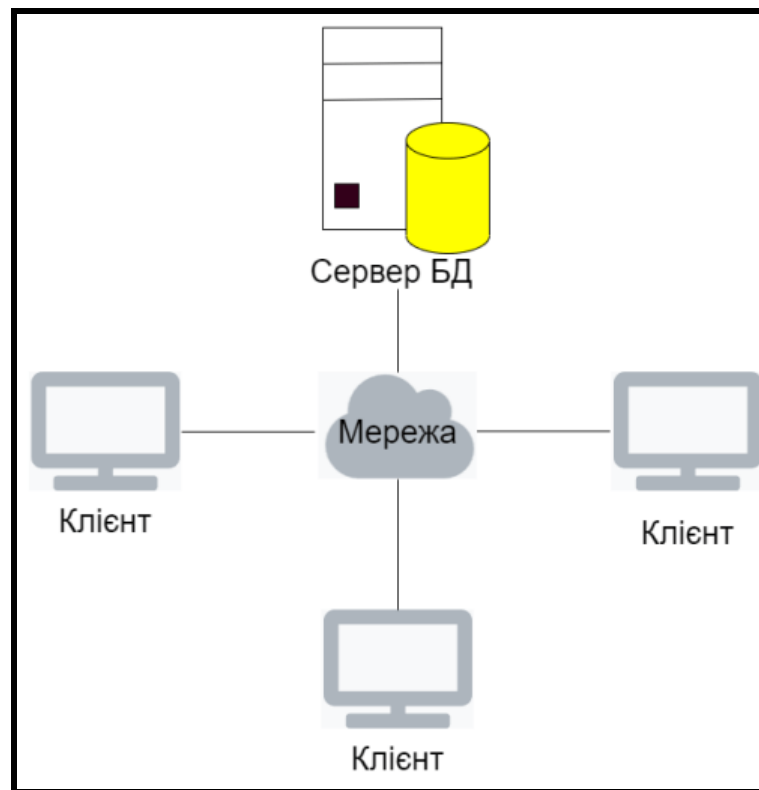


Рисунок 1.6 – Приклад клієнт-серверної моделі

У найпростішому вигляді система клієнт-сервер складається з одного БД і одного або кількох клієнтів. Проте сьогодні багато систем клієнт-сервер включають додаткові сервери. Рисунок 1.7 показує дві системи клієнт-сервер, які включають додатковий сервер між клієнтами та сервером БД.

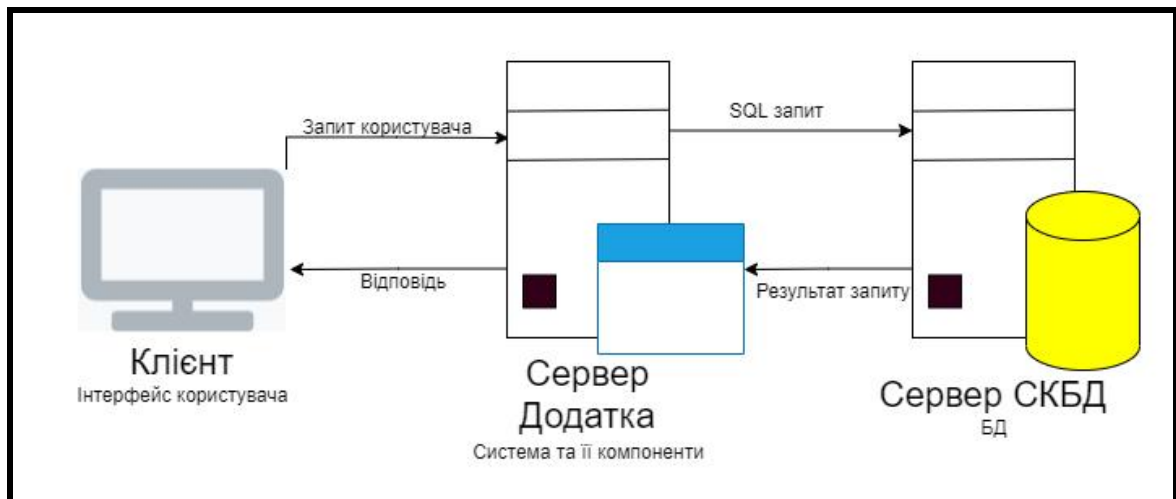


Рисунок 1.7 – Додатковий сервер між клієнтами та сервером БД

Система клієнт-сервер зображена на рисунку 1.8.



Рисунок 1.8 – Проста вебсистема клієнт-сервер

Компонент бізнесу – це логічне представлення однієї або кількох таблиць. Бізнес-компоненти є основою для контролю за тим, як дані відбираються, вставляються та оновлюються в базових таблицях [52].

Окрім сервера БД та клієнтів, система клієнт-сервер може включати додаткові сервери, такі як сервери додатків та вебсервери.

Сервери додатків, зазвичай, використовуються для зберігання бізнес-компоненти, які виконують частину обробки даних. Зокрема, ці компоненти використовуються для обробки запитів до БД з інтерфейсу користувача, запущеного на клієнті за допомогою браузера.

Вебсервери, зазвичай, використовуються для зберігання вебпрограм та вебслужб. Вебпрограми – це програми, призначені для роботи на вебсервері. Вебслужби подібні до бізнес-компонентів, за винятком того, що, як і вебпрограми, вони призначені для роботи на вебсервері.

У вебсистемі веббраузер, запущений на клієнті, надсилає запит вебсерверу через Інтернет. Потім вебсервер обробляє запит і передає будь-які запити щодо даних на сервер БД.

Більш складна архітектура системи може включати два або більше серверів додатків, вебсервери та сервери БД [4], [51].

Хоча прикладне програмне забезпечення працює на клієнті, воно використовує дані, які зберігаються на сервері. Для цього сервер використовує API доступ до даних. Метод створення запитів, що ви використовуєте для роботи з API, залежить від мови програмування та API. Після того, як клієнт надсилає запит до СКБД, ця СКБД інтерпретує запит і надсилає результати назад до клієнта [53]-[54].

Система управління базами даних (СУБД) – це програмне забезпечення, яке взаємодіє з кінцевими користувачами, додатками та самою базою даних для збирання та аналізу даних. Програмне забезпечення СУБД додатково включає основні засоби, що надаються для адміністрування бази даних. Сукупність бази даних, СУБД та пов'язаних із нею додатків можна назвати системою баз даних. Часто термін БД також використовується у вільному розумінні для позначення будь-якої з СУБД, системи баз даних або додатку, пов'язаного з базою даних [55]. Порівняння можливостей СУБД наведено у таблиці 1.2.

Bootstrap 5 був офіційно випущений 5 травня 2021 року. На поточний момент останньою версією є версія Bootstrap 5.1.0.

Таблиця 1.2 – Порівняння СКБД

Критерії	СКБД			
	MariaDB	MySQL	Oracle	MSSQL Server
Структура даних	SQL	SQL	Мульти-модельна, SQL	T-SQL
Ліцензія	Відкрита	Відкрита	Закрита	Закрита
Якість документації	Висока	Середня	Висока	Висока
Масштабованість	Вертикальна	Вертикальна зі складною структурою	Вертикальна	Вертикальна зі складною структурою
Підтримка інших структур даних	Document store; Graph DBMS; Spatial DBMS	Document store; Spatial DBMS	Document store; Graph DBMS; RDF store; Spatial DBMS	Document store; Graph DBMS; Spatial DBMS
Складність для освоєння	Середня	Середня	Висока	Висока
Створена за допомогою мови програмування	C, C++	C and C++	C and C++	C++
Рік створення	2009	1995	1980	1989
Розробник	MariaDB Corporation	Oracle	Oracle	Microsoft
Рівень популярності	Середній	Дуже високий	Дуже високий	Дуже високий

Для його роботи необхідно підключити два основних файли:

- «bootstrap.min.css»;
- «bootstrap.bundle.min.js» або «bootstrap.min.js».

Переваги Bootstrap 5:

- налаштованість елементів, завдяки класам, які можна видалити або додати, щоб елемент відповідав будь-яким вимоги;
- велика кількість класів для стилізації майже кожного елемента HTML;
- зменшує час розробки адаптивного дизайну вебсторінок;
- логічність, яка присутня в кожному класі і для тих користувачів хто знайомий з CSS, це буде величезним плюсом як в освоєння інструменту в цілому, так і для знаходження нових можливостей;
- підтримка – Bootstrap має величезну підтримку спільноти, тому ви зазвичай можете отримати допомогу, коли у вас виникнуть проблеми;
- кросбраузерність;

- не великий розрір файлів, які необхідно підключати;
- надзвичайно зручна та зрозуміла документація з неймовірною кількістю прикладів як на самому сайті, так і додаткових прикладів для кожного елемента, що можна завантажити;
- можливість підтримки інших мов завдяки заготовленим стилям для тих мов де читаю з право наліво, а не навпаки;
- відкритий вихідний код;
- безліч безплатних та професійних шаблонів, тем та плагінів.

Недоліки Bootstrap 5:

- може потребувати багато часу на розробку та оформлення вебзастосунку, якщо дизайн має тенденцію відхилятися від звичного дизайну, що використовується у Bootstrap;
- у деяких випадках елементи можуть мати велику кількість класів, що ускладнити їх використання;
- під час створення дизайну вам доведеться докласти додаткових зусиль, інакше всі вебзастосунки будуть виглядати однаково, якщо ви не зробите серйозних налаштувань;
- відмова від прямої залежності з jQuery, тому Bootstrap 5 став ще набагато швидше ніж був раніше, та цього чекали дуже багато розробників зі всього світу [56].

Станом на жовтень 2021 року Bootstrap – одинадцятий за кількістю зірок проєкт на GitHub, що має понад 154000 зірок (що є надзвичайно гарним показником).

1.5 Визначення завдання роботи

Фреймворк, що розроблюється, буде складатися з наступних частин:

- маршрутизатор – призначений для створення навігації до кожної сторінки везастосунка;

- клас для взаємодії з базами даних (MySQL);
- шаблонізатор – призначений для полегшення та зменшення частин коду для написання відображень;
- валідатор – містить у собі набір функцій, класів, що перевіряють відповідність вказаних даних, застосовуючи один з існуючих валідаторів системи, або шляхом розширення класу BaseRule створити власний валідатор та використовувати його;
- ORM з використанням шаблону проєктування ActiveRecord;
- MigrateCommand клас, який буде надавати базовий інтерфейс для обробки міграції системи;
- контролер для авторизація користувача у системі;
- клас для взаємодії з сесіями;
- клас для створення форми та її елементів;
- клас для взаємодії з файловою системою;
- користувацький інтерфейс, що відповідає за відображення даних;
- реалізація архітектурного шаблону MVC;
- надання файлу для конфігурації базових класів та параметрів;
- у разі виникнення критичних помилок, або тих, які не вдалося обробити, фреймворк повинен надати повідомлення, назву файла, рядок, у якому допущена помилка, тільки у тому разі, якщо встановлений режим розробки (на реальному сервері будь-які помилки, пов'язані з кодом, повинні бути приховані);
- використовувати бібліотеки Bootstrap 5 для створення елементів інтерфейсу;
- розробити засоби захисту інформації;
- використовувати бібліотеку для відображення помилок при розробці та взаємодії з консоллю.

На відміну від вже існуючих фреймворків, у даному фреймворку повинні бути враховані недоліки інших аналогів для спрощення написання

коду, зменшення витрат часу на розробку вебзастосунок. Фреймворк повинен надавати лише необхідні функції для пришвидшення обробки запитів та надання відповіді для клієнтської частини.

1.6 Висновки за розділом 1

В першому розділі проаналізовані особливості предметної області, проведено аналіз існуючих PHP-фреймворків і виявлено їх переваги та недоліки.

Також, було доведено актуальність розробки власного PHP-фреймворку та сформульовано основні вимоги до системи, що розроблюється.

Обрано технології, які будуть використовуватися у фреймворку, а саме – мова програмування PHP та середовища розробки PhpStorm, за допомогою якого буде проводитись розробка програмного забезпечення. У якості інструментів роботи із базами даних було обрано реляційну модель БД та СКБД MySQL. Для створення виглядів на клієнтській стороні буде використовуватися бібліотека Bootstrap 5. Сформульовані вимоги як для функцій системи, так і для інтерфейсу. Розглянута мова тегів та розмітки стилів, що допомагає створювати власні стилізовані елементи у системі, та буде використовуватися для створення виглядів. У якості менеджера залежностей був обраний Composer, що використовує репозиторій Packagist, у якому знаходиться більшість пакетів, написаних за допомогою мови програмування PHP.

2 РОЗРОБКА АРХІТЕКТУРИ ФРЕЙМВОРКУ

Даний розділ присвячений опису та обґрунтуванню програмно-технічних рішень, які були прийняті під час проєктування програмного забезпечення (ПЗ).

Для легкості та зручності було прийняте рішення використовувати шаблон проєктування MVC, ORM (анг. Object Relational Mapping, Об'єктно-реляційна проєкція), що значно покращує роботу з об'єктами, спрощуючи зберігання нових записів і оновлюючи вже існуючі.

2.1 Структура та архітектура проєкту

Використання архітектури MVC та розроблювального фреймворка потребує розміщення файлів з вихідним кодом певним чином. Основні каталоги проєкта описані у таблиці 2.1.

Таблиця 2.1 – Опис каталогів проєкту

Назва каталогу	Опис
app	Головний каталог проєкту. Цей каталог містить у собі моделі, контролери, маршрутизацію та команди
app/routes	Містить файл «web.php», що описує усі можливі шляхи для навігації та їх параметри
app/Models	Повинна зберігати усі моделі системи, що взаємодіють з PDO API, яке у свою чергу виконує запити до БД та повертає результат, якщо дані були знайдені/додані/видалені/оновлені
app/Http/Controllers	Повинна зберігати усі контролери системи, які приймають запити від клієнта, та передають оброблений запит (дані) до моделі, яка у свою чергу виконує обробку отриманих даних
app/Console/Commands	Зберігає у собі класи, які повинні розширювати функціонал базового класу Command, який дозволяє взаємодіяти з ПЗ, використовуючи командний рядок
config	Містить у собі два файли: «config.php», «providers.php». Перший відповідає за базову конфігурацію проєкту, наприклад, інформація щодо БД, назва вебзастосунку, шлях до каталогу з виглядами. Другий, у свою чергу повертає масив необхідних провайдерів, які можна додавати та видаляти. Ці додані файли використовуються у якості додаткового функціонала, зокрема, провайдер, клас валідації даних та виглядів
database/migrations	Містить у собі класи для створення таблиці у базі даних
framework	Головний каталог фреймворку, який містить у собі базові ключові класи та компоненти системи, які дозволяють реалізувати взаємодію між собою

Продовження таблиці 2.1

Назва каталогу	Опис
framework/authentication	Містить основні класи, які відповідають за авторизацію та аутентифікацію користувача у системі
framework/db	Містить основні класи, які взаємодіють з БД
framework/exceptions	Містить класи, які відповідають за оброблення та відображення помилок
framework/helpers	Містить допоміжні класи, які взаємодіють зі строками, числами, масивами, зображенням та HTML елементами
framework/Files	Містить допоміжні класи, які допомагають оброблювати файли
framework/middlewares	Містить класи, які використовуються для обмеження дій користувачів, наприклад, обмеження доступу до певних частин проєкту
framework/notification	Містить у собі клас Notification, що дозволяє створювати та показувати повідомлення для користувачів
framework/Provider	Містить у собі класи, які описують додаткові допоміжні методи по обробці даних (ValidationProvider, ViewProvider)
framework/routing	Містить у собі класи (Route, RouteHandle, Router), які обробляють запит користувача з метою створення відповіді на основі вибраного навігаційного шляху, ґрунтуючись на назвах навігації
framework/validation	Містить менеджер класів, призначених для зберігання в собі інших класів, які валідують певні типи даних
framework/validation/rules	Містить класи, які відповідають за перевірку певних типів даних
framework/View	Містить у собі клас Manager відображень, що надає обраний шаблонізатор, для створення відображення
framework/View/Engine	Містить у собі класи, які обробляють різноманітні шаблони
framework/View/form	Містить клас Form, що надає HTML елементи форми
framework/View/form/elements	Містить у собі класи для відображення елементів форми (Input, Textarea)
public	Містить у собі файл точки входу «index.php» та «.htaccess». Також може містити іконку вебзастосунку.
resources	Зберігає різноманітні ресурси проєкту (файли формату – css, js, зображення, шрифти)
resources/js	Містить у собі усі файли формату «.js» (окрім тих, що підключаються з інших джерел), які використовуються у вебзастосунку
resources/styles	Містить у собі усі файли формату «.css» (окрім тих, що підключаються з інших джерел), які використовуються у вебзастосунку
resources/views	Містить у собі усі файли – відображення, які належать проєкту
resources/views/layouts	Містить у собі різноманітні макети, які змінюють розташування головних елементів системи (верхнього та нижнього меню)
runtime	Зберігає у собі тимчасові файли проєкту, які можуть генеруватися під час роботи системи
storage	Зберігає у собі зображення та файли, які будуть створюватися під час реалізації однієї чи іншої функції або метода
storage/framework/views	Зберігає у собі файли виглядів, які були згенеровані за допомогою шаблонізатора

Кожен каталог має своє призначення та структуру, яку необхідно дотримуватися для того, щоб будь-який розробник міг легко орієнтуватися в

існуючому проєкті. Розроблена структура каталогів робить фреймворк схожим на інші аналоги тому, що при розробці архітектури було вирішено використовувати шаблон проєктування MVC.

Також система містить зразковий файл під назвою «.env.example», який зберігає у собі змінні середовища, такі як:

- назва проєкту;
- тип середовища (розробка чи справжній сервер);
- інформацію про з'єднання з БД (типа з'єднання, IP адресу сервера, порт, назву БД, користувача та його пароль);
- налаштування, які використовуються для відправки пошти.

Перед створенням головного об'єкта вебзастосунку класу Application, цей файл конфігурації зчитується та задає ці змінні середовища до масиву конфігурації, які потім можна використовувати впродовж роботи вебзастосунку в будь-якому місці для зчитування даних, що там зберігаються.

Model-view-controller (MVC) – це шаблон проєктування програмного забезпечення, який зазвичай використовується для розробки інтерфейсів, які поділяють пов'язану логіку програми на три взаємопов'язані елементи. Ці три елементи – це модель, вигляд та контролер, які можна модифікувати (рис 2.1).

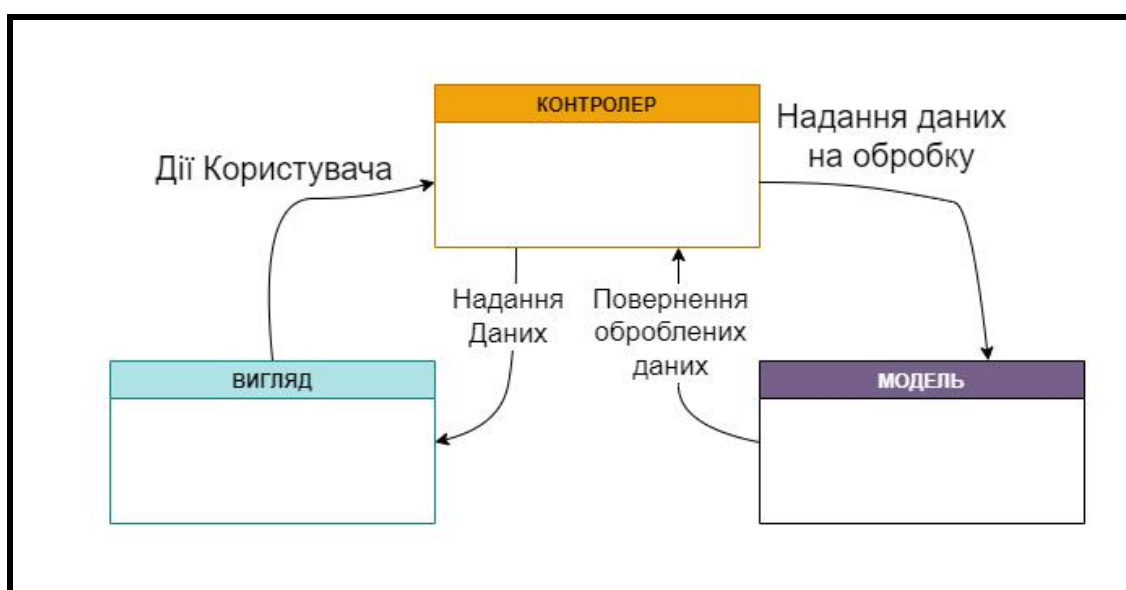


Рисунок 2.1 – Архітектурний шаблон MVC, що розділяє додаток на три основні групи компонентів

Основні складові MVC:

- модель (Model) – це динамічна структура даних програми, яка не залежить від призначеного для користувача інтерфейсу, що безпосередньо управляє даними, логікою і правилами застосування. Відповідає за управління даними додатка, отриманих з БД;
- вигляд (View) – будь-яке уявлення інформації, наприклад, графік, діаграма, таблиця або текстові дані. Можливо кілька подань однієї і тієї ж інформації, наприклад, гістограма для керівництва і табличне представлення для бухгалтерів;
- контролер (Controller) – приймає вхідні дані і перетворює їх в команди для моделі або вигляду [11].

Для функціонування вебсайту в Інтернеті потрібен не тільки комп'ютер і вихід до мережі, але й встановлене на нього спеціальне програмне забезпечення, що надає доступ до даних за протоколами HTTP та HTTPS. Таким програмним забезпеченням і є вебсервер. Існують різні типи вебсерверів, але найбільш поширений – Apache. Він використовує відкритий код, надається безплатно, постійно допрацьовується і поліпшується, підтримує більшість скриптів і працює майже на всіх платформах, у тому числі на Windows, Linux, MacOS.

Hypertext Transfer Protocol Secure (від англ. HTTPS) – захищений протокол передачі HTTP. Він використовується для безпечного зв'язку через комп'ютерну мережу в Інтернеті. HTTPS – протокол зв'язку, шифрується на транспортному рівні (від англ. TLS, Transport Layer Security), раніше відомий як рівень захищених сокетів (від англ. SSL, Secure Sockets Layer). Тому протокол також називають HTTP через TLS або HTTP через SSL, який є розширенням протоколу HTTP [57].

Безпека зв'язку використовує засоби, які запобігають доступу неавторизованих перехоплювачів до телекомунікацій, що використовуються для передачі даних, які легко може отримати лише реципієнт [58].

Фреймворк містить у собі два файли «.htaccess», які необхідні для його запуску:

- перший файл знаходиться в кореневому каталозі проєкту і служить для переадресації на ресурси проєкту (файли формату – css, js, зображення, шрифти), або ж переадресує на другий файл «.htaccess», що знаходиться у каталозі «public/», якщо це не підпадає під категорію ресурсу;
- другий файл знаходиться у каталозі «public/» та перевіряє чи існує файл або каталог, який було запитано у вебсерверу, та, якщо збігів не знайдено, відбувається переадресація на «точку входу» («public/index.php»).

Точкою входу до проєкту є файл «index.php», що знаходиться у каталозі під назвою «public». Повний шлях до файлу виглядає як «public/index.php».

У комп'ютерному програмуванні «точка входу» – це точка у програмі, додатку або вебзастосунку, де починається виконання програми, і де програма має доступом до аргументів командного рядка.

Налаштовують Apache у конфігураційних файлах, які зберігають у текстовому форматі. У цих файлах можна задати певні правила дії вебсервера. Головний конфігураційний файл має назву httpd.conf або apache.conf (в залежності від дистрибутива ОС). У багатьох випадках усі конфігурації сервера можна вказувати саме тут, і в основному такий спосіб є кращим, тому що він:

- зменшує час відповіді вебсервера при запиті (так при кожному зверненні до вебсервера Apache, який не буде переглядати всі каталоги на наявність файлу «.htaccess»);
- деякі директиви, наприклад, директиви модуля mod_rewrite, здебільшого працюють краще з головного конфігураційного файлу [59].

Файли «.htaccess» мають у собі базову конфігурацію для вебсервера Apache. Вебсервер Apache має конфігурацію за замовчуванням, але доступ до неї не завжди можливо отримати. Наприклад, на спільних хостингах доступ до налаштувань Apache вебсервера зазвичай заборонено, тому було вирішено створити власні налаштування за замовчуванням, щоб фреймворком міг користуватись будь-який розробник.

Через це було прийнято рішення створити конфігурацію для файлів «.htaccess», що дозволить налаштувати вебсервер Apache на будь-якому хостингу, що підтримує Apache вебсервер. Ті розробники, які мають доступ до налаштувань Apache вебсерверу, отримають перевагу у швидкості роботи.

2.2 Класи системи

Під час розробки програми були створені основні класи та допоміжні.

Основні класи визначають поведінку системи від контролерів, що отримують інформацію та направляють її на обробку до моделі, після чого модель з інформацією направляється до вигляду, що у свою чергу показує необхідну інформацію у вигляді HTML тегів, до застосовуються CSS класи чи стилі.

Допоміжні класи – це класи, які відіграють другорядну роль у фреймворках, вони надають додаткові можливості по взаємодії з різноманітними типами даних чи HTML тегами. Допоміжні класи корисні для надання доступу до часто використовуваних та призначені для допомоги в забезпеченні деяких функцій, що не є основною метою вебзастосунку або класу, в яких він використовується, і водночас дозволяє уникнути дублювання цього коду по всьому проєкту.

Кожен користувацький клас системи має спадкувати базовий клас (Model, Controller, BaseActiveRecord, Command, MigrateCommand, BaseRule), так як ці класи надають мінімально необхідний функціонал для того, щоб усе працювало.

2.2.1 Основні класи системи

Основні класи фреймворку зберігаються у каталозі «framework».

Клас Controller призначений для обробки даних, які будуть отримані після того, як користувач системи зробить запит до будь-якого маршруту

(шляху навігації) системи. Controller надає мінімально необхідну конфігурацію для початку робот. Поля та методи класу наведені у таблиці 2.2.

Таблиця 2.2 – Поля та методи класу Controller

Метод	Опис
redirect	Відкритий метод, що слугує для перенаправлення користувача на вказану сторінку
goHome	Відкритий метод, що слугує для перенаправлення користувача на головну сторінку
goBack	Відкритий метод, що слугує для перенаправлення користувача на попередню сторінку, якщо така була знайдена

Клас Model призначений для обробки даних, які будуть отримані після того, як контролер створить модель та задасть певні значення до полів з отриманого запиту від користувача. Клас Model розширює клас BaseActiveRecord. Поля та методи класу наведені у таблиці 2.3.

Таблиця 2.3 – Поля та методи класу Model

Поле/Метод	Опис
SCENARIO_NEW	Відкрите поле, що зберігає у собі тип сценарію, який може бути використаний у дочірньому класі
SCENARIO_EDIT	Відкрите поле, що зберігає у собі тип сценарію, який може бути використаний у дочірньому класі
SCENARIO_DELETE	Відкрите поле, що зберігає у собі тип сценарію, який може бути використаний у дочірньому класі
SINGLE_ERROR_MESSAGE	Відкрите поле, яке визначає чи є необхідність зберігати у моделі більше, ніж одну помилку, що допустив користувач системи під час заповнення форми для кожного поля
\$scenario	Відкрите поле, що зберігає у собі тип сценарію, який необхідно буде виконати під час певних дій системи
\$errors	Закрите поле, що зберігає у собі помилки, які допустив користувач системи під час заповнення форми
\$id	Закрите поле, що зберігає у собі id (ідентифікатор запису)
\$fillable	Відкритий метод, що повертає масив полів та валідаторів, які необхідно застосувати до поля
rules	Відкритий абстрактний метод, який потребує реалізації у дочірньому класі
tableName	Метод, що повертає назву таблиці БД
construct	Відкритий метод, конструктор, що створює необхідну конфігурацію для моделі під час створення об'єкту
load	Відкритий метод, що заповнює поле \$attributes значеннями
getFillable	Відкритий метод, що повертає поле \$fillable, або власний масив полів
validate	Відкритий метод, що проводить валідацію кожного значення масиву \$attributes
addError	Відкритий метод, що додає помилки до поля \$errors
errorMessages	Відкритий метод, що повертає поле \$errors
hasError	Відкритий метод, що перевіряє, чи має певне поле помилку

Продовження таблиці 2.3

Поле/Метод	Опис
hasErrors	Відкритий метод, що перевіряє, чи має будь-яке поле моделі помилки
getFirstError	Відкритий метод, що повертає першу помилку для вказаного поля, чи нульову строку, якщо помилки не знайдено
Labels	Відкритий метод, що повертає назву поля, яку можна використовувати в будь-якому місті в даному класі
getLabel	Відкритий метод, що повертає назву для вказаного поля
addDirty	Відкритий метод, що додає до масиву \$dirty поле, яке було змінено
booted	Захищений метод, що запускається під час створення об'єкту
scenarios	Захищений метод, що зберігає у собі та повертає масив сценаріїв
getActiveScenario	Захищений метод, що повертає масив з полями, з якими можна взаємодіяти
clearErrors	Захищений метод, що видаляє усі помилки моделі, або ж для вказаного поля
getValidatorAndAttributes	Захищений метод, що повертає масив полів, валідаторів, які необхідно застосувати до цих полів
isDirty	Захищений метод, що перевіряє чи змінювалося значення поля, чи ні
notifyValidationErrors	Захищений метод, що створює повідомлення про помилки, які виникли у моделі

Клас `BaseActiveRecord` має таку назву через те, що він реалізує шаблон проєктування під назвою «`ActiveRecord`».

`ActiveRecord` використовується у програмному забезпеченні, яке зберігає дані об'єкта в пам'яті в реляційних базах даних. Мартін Фаулер дав назву шаблону `ActiveRecord` у своїй книзі «`Patterns of Enterprise Application Architecture`» у 2003 році. Інтерфейс об'єкта, що відповідає цьому шаблону, включатиме такі функції, як `Insert`, `Update` та `Delete`, а також властивості, які відповідають стовпцям базової таблиці БД [60].

`ActiveRecord` – це підхід до доступу даних у БД. Екземпляр об'єкта цього класу прив'язаний до одного рядка в таблиці. Після створення об'єкта та його заповнення, він зберігається у відповідній таблиці БД, за яку відповідає. При запиті до БД, будь-який рядок зберігається у об'єкті, який можна буде змінити, чи видалити, та відповідні дії відобразяться у самій БД. Клас `ActiveRecord` описує методи або властивості доступу для кожного стовпця в таблиці. Розробник може створити похідний клас, який розширює або частково змінює деяку поведінку об'єкта [60]. Цей шаблон зазвичай використовується разом із об'єктно-реляційною проєкцією (ORM).

Клас `BaseActiveRecord` призначений для обробки даних, які будуть отримані після того, як користувач системи зробить запит до будь-якого маршруту (шляху навігації) системи, що містить у собі модель. Базовий клас надає лише найнеобхідніші поля та методи. Усі методи цього класу є відкритими «public», а поля захищеними «protected». Поля та методи класу `BaseActiveRecord` описані у таблиці 2.4.

Таблиця 2.4 – Поля та методи класу `BaseActiveRecord`

Поле/Метод	Опис
<code>\$connection</code>	Поле, яке зберігає у собі з'єднання з БД
<code>\$table</code>	Поле, яке зберігає у собі назву таблиці БД
<code>\$attributes</code>	Поле, яке зберігає у собі усі необхідні поля/атрибути (стовпці) БД для поточного об'єкта
<code>\$dirty</code>	Поле, яке зберігає у собі усі поля, які були змінені
<code>with</code>	Статичний метод, що створює іншу модель, яка пов'язана моделлю, у якій цей метод використовується
<code>query</code>	Статичний метод, що повертає об'єкт з'єднання з БД
<code>__callStatic</code>	Статичний метод, що намагається обробити статичний виклик методу, який був використаний як об'єкт з'єднання з БД
<code>find</code>	Статичний метод, що приймає у якості параметра <code>id</code> рядку БД, та повертає його, якщо такий рядок був знайдений у БД
<code>__get</code>	Магічний метод моделі, що намагається отримати значення поля, які не було знайдено, використовуючи певний алгоритм для знаходження цих полів серед вже існуючих полів чи методів, та повертає значення цього поля, якщо воно було знайдено
<code>__set</code>	Магічний метод моделі, що намагається задати значення полю, яке не було знайдено або до якого немає доступу. Використовуючи існуючі методи, об'єкт намагається задати значення полю з яким взаємодіє цей метод. Якщо назва поля знайдена у масиві <code>\$attributes</code> , то вона буде задана у цьому масиві, а також буде додано назву поля, яке буде зберігатися у масиві <code>\$dirty</code> . У разі, якщо поле не знайдено, у масиві <code>\$attributes</code> , використовується певний алгоритм для знаходження цих полів серед вже існуючих методів
<code>setConnection</code>	Метод, що задає значення полю <code>\$connection</code>
<code>getConnection</code>	Метод, що повертає значення поля <code>\$connection</code>
<code>setTable</code>	Метод, що задає значення полю <code>\$table</code>
<code>getTable</code>	Метод, що повертає поле <code>\$table</code>
<code>getAttributes</code>	Метод, що повертає усі поля/атрибути об'єкту (тобто поля таблиці БД)
<code>save</code>	Метод, що зберігає запис до БД
<code>update</code>	Метод, що оновлює запис у БД
<code>delete</code>	Метод, що видаляє запис з БД
<code>getDirty</code>	Метод, що повертає масив, який містить назви полів, які були змінені. Цей метод можна використовувати у тих випадках, коли необхідно зберегти лише ті данні, які були змінені
<code>toInt</code>	Метод, що приводить тип даних вказаного атрибута до цілого числа
<code>hasOne</code>	Метод, що допомагає створювати зв'язок з іншою моделлю (таблицею), яка має зв'язок один до одного, головної таблиці до підлеглої
<code>getDb</code>	Метод, що повертає об'єкт з'єднання з БД

Продовження таблиці 2.4

Поле/Метод	Опис
hasMany	Метод, що допомагає створювати зв'язок з іншою моделлю (таблицею), яка має зв'язок один до багатьох
belongsTo	Метод, що допомагає створювати зв'язок з іншою моделлю (таблицею), яка має зв'язок один до одного, підпорядкованої таблиці до головної

Клас `AdvancedEngine`, що призначений для зберігання у собі конфігурації шаблонізатора. Клас `AdvancedEngine` містить у собі усі можливі обробники для виводу даних, такі як: екранування вихідних строк тексту, додавання нових виглядів у поточний вигляд, генерування РНР конструкцій (коду). Для перетворення шаблону до реального РНР коду, що можна запустити на виконання, використовуються регулярні вирази, які допомагають знайти необхідну послідовність символів та виконати над ними наступні операції:

- додавання іншого параметра;
- заміна формальних параметрів фактичними.

Регулярний вираз (від англ. `regex` або `regexr`), або раціональний вираз – це послідовність символів, яка визначає шаблон пошуку. Зазвичай такі шаблони використовуються алгоритмами пошуку рядків для «знаходження» або «знаходження і заміни» певних символів у рядку, або для перевірки вводу. Ця методика розроблена в теоретичній інформатиці та теорії формальної мови [61].

Формальна мова – множина скінчених послідовностей символів, які описуються правилами певного виду, і називаються граматиною, або синтаксисом мови [62].

Поля та методи класу наведені у таблиці 2.5.

Таблиця 2.5 – Поля та методи класу `AdvancedEngine`

Поле/Метод	Опис
<code>\$layouts</code>	Захищене поле, що зберігає назви шаблонів, які вже були створені, та до яких можна отримати доступ у поточному вигляді
<code>compile</code>	Захищений метод, який використовує регулярні вирази, що отримують параметри, які пізніше будуть передані до певного метода або конструкції на виконання
<code>include</code>	Відкритий метод, що додає один вигляд до іншого

Продовження таблиці 2.5

Поле/Метод	Опис
render	Відкритий метод, що перевіряє, чи існує вже такий файл у поточному шаблонізаторі, та чи не є цей файл застарілим. Якщо файл буде визнаний як застарілий, то новий буде створений автоматично
extends	Відкритий метод, що розширює вже існуючий вигляд
__call	Відкритий метод, що використовує менеджер макросів (алгоритми обробки даних), який зберігає у собі більшість методів з обробки вихідних даних, які будуть відображені у вигляді

Router – клас, що відповідає за маршрутизацію усього фреймворку. Router направляє користувача на ту сторінку, на яку користувач запитував. Також Router відповідає за створення та передачу параметрів до контролера, а зокрема це об'єкти наступних класів: Request, Response. Додатковим, але не обов'язковим є використання проміжного програмного забезпечення (middleware), що може обмежувати доступ до деяких частин вебзастосунку. Для обмеження доступу до деяких частин вебзастосунку, розробнику необхідно реалізувати клас BaseMiddleware, який дозволяє змінити поведінку існуючих маршрутів системи (вибірково).

Проміжне програмне забезпечення (від англ. middleware, ППЗ) – шар програмного забезпечення, що є посередником між різними компонентами великого застосунку.

Поля та методи класу наведені у таблиці 2.6.

Таблиця 2.6 – Поля та методи класу Router

Поле/Метод	Опис
\$request	Відкрите поле, що зберігає об'єкт класу Request
\$response	Відкрите поле, що зберігає об'єкт класу Response
\$activeRoute	Захищене поле, що зберігає об'єкт класу Route
\$routes	Захищене поле, що зберігає у собі усі можливі маршрути системи у вигляді масиву
\$rules	Захищене поле, що зберігає у собі набір правил, які може представляти об'єкт типу BaseMiddleware. Правила зберігаються у вигляді масиву
getInstance	Відкритий метод, що повертає єдиний екземпляр цього ж класу
get	Відкритий метод, що призначений для створення нових маршрутів, до яких можна отримати доступ, використовуючи запит «GET»
post	Відкритий метод, що призначений для створення нових маршрутів, до яких можна буде отримати доступ, використовуючи запит «POST»

Продовження таблиці 2.6

Поле/Метод	Опис
addSystem	Відкритий метод, що призначений для створення нових системних маршрутів, таких як сторінка «404» – «сторінку не знайдено»
addNew	Відкритий метод, що призначений для додавання маршрутів до масиву \$rules
group	Відкритий метод, що призначений для об'єднання деяких схожих правил, для того, щоб уникнути повторного коду
middleware	Відкритий метод, що призначений для обмеження деяких можливостей маршруту, наприклад, тільки авторизовані користувачі можуть переглядати сторінку
route	Відкритий метод, що призначений для знаходження маршруту, який був запитаний користувачем. Для пошуку маршруту використовуються регулярні вирази, які можна змінювати в залежності від цілей
getActiveRoute	Відкритий метод, що призначений для отримання поточного об'єкту типу Route
resolve	Відкритий метод, що призначений для задання поточного маршруту, шляхом створення обраного контролера, який буде оброблювати поточний запит
calculateAccessRules	Закритий метод, що призначений для виконання певних правил, які були задані для вказаного маршруту
checkIfRouteAlreadyAdded	Закритий метод, що перевіряє чи існує такий маршрут
routeExistIn	Закритий метод, що перевіряє чи існує маршрут в одному з підмасивів «POST» або «GET»
setCurrentActiveRoute	Закритий метод, який задає активний маршрут
resolveController	Закритий метод, який намагається знайти клас системи, ідентифікувати чи є він дочірнім класом Controller, і повернути цей клас, якщо він був знайдений
match	Закритий метод, який шукає заданий маршрут та повертає об'єкт Route, який відповідає за поточний маршрут

Клас Route призначений для зберігання і створення конкретного маршруту системи та зберігає у собі назву метода та класу контролера, який необхідно створити задля того, щоб Route міг повернути поточний маршрут та направити користувача на обрану сторінку. Поля та методи класу наведені у таблиці 2.7.

Таблиця 2.7 – Поля та методи класу Route

Поле/Метод	Опис
\$parameters	Захищене поле, що зберігає у собі параметри маршруту «GET» або «POST»
\$method	Захищене поле, що зберігає у собі поточний метод «GET» або «POST»
\$path	Захищене поле, що зберігає у собі шлях, за який відповідає обраний об'єкт типу Route
\$name	Захищене поле, що зберігає у собі назву маршруту
\$handler	Захищене поле, що зберігає у собі клас і метод, який буде викликано у разі навігації до цієї сторінки
name	Відкритий метод, що призначений для отримання поля \$name

Продовження таблиці 2.7

Поле/Метод	Опис
matches	Відкритий метод, що призначений для заміни формальних параметрів типу «{name}» на фактичні, які було отримано під час запиту до сервера
className	Відкритий метод, що призначений для отримання ім'я класу поточного маршруту
methodName	Відкритий метод, що призначений для отримання ім'я метода поточного маршруту
getActiveController	Відкритий метод, що призначений для повернення щойноствореного контролера, який відповідає за поточний маршрут
parameters	Відкритий метод, що призначений для повернення масиву параметрів \$parameters

Клас Request призначений для зберігання у собі інформації щодо поточного шляху (URL), його параметрів та методу, який був використаний для отримання даних з ресурсу.

URL (від англ. Uniform Resource Locator) – це посилання на вебресурс, який визначає його розташування в комп'ютерній мережі та механізм його отримання [63].

Поля та методи класу наведені у таблиці 2.8.

Таблиця 2.8 – Поля та методи класу Request

Поле/Метод	Опис
\$uri	Відкрите поле, рядок, що зберігає у собі повний шлях маршруту разом із параметрами
\$urlParts	Відкрите поле, масив, що зберігає у собі елементи path та query. Path зберігає у собі лише шлях маршруту, а query зберігає у собі параметри, якщо вони були задані
\$requestPath	Відкрите поле, що зберігає у собі шлях маршруту
\$hasParams	Відкрите поле, що зберігає у собі булеве значення, чи має поточний маршрут «GET» параметри
\$method	Закрите поле, що зберігає у собі поточний тип HTTP запиту
\$referer	Закрите поле, що зберігає у собі попередню адресу маршруту, якщо вона була
getPath	Відкритий метод, що повертає поточний шлях маршруту
getMethod	Відкритий метод, що повертає поточний метод запиту
isGet	Відкритий метод, що визначає чи є поточний метод запиту методом «GET»
isPost	Відкритий метод, що визначає чи є поточний метод запиту методом «POST»
getBody	Відкритий метод, що повертає тіло запиту
getUri	Відкритий метод, що повертає масив, який зберігає у собі поле \$uri
refererPage	Відкритий метод, що повертає масив, який зберігає у собі поле \$referer

QueryBuilder – базовий клас, що описує загальні методи по створенню запитів до БД. Поля та методи класу наведені у таблиці 2.9.

Таблиця 2.9 – Поля та методи класу QueryBuilder

Поле/Метод	Опис
TYPE_SELECT	Відкрите поле, що зберігає у собі тип запиту на вибірку
TYPE_INSERT	Відкрите поле, що зберігає у собі тип запиту на додавання нового запису
TYPE_UPDATE	Відкрите поле, що зберігає у собі тип запиту на оновлення
TYPE_DELETE	Відкрите поле, що зберігає у собі тип запиту на видалення запису
\$limit	Захищене поле, що використовується для зберігання максимальної кількості записів, які можна отримати одним запитом
\$offset	Захищене поле, що використовується для зберігання кількості записів, які необхідно пропустити
\$type	Захищене поле, що використовується для зберігання типу запиту (TYPE_SELECT, TYPE_INSERT, TYPE_UPDATE, TYPE_DELETE)
\$table	Захищене поле, що використовується для зберігання назви таблиці, до якої відбувається запит
\$orderBy	Захищене поле, що використовується для зберігання порядку сортування
\$values	Захищене поле, що використовується для зберігання значень, які використовуються у запиті
\$columns	Захищене поле, що використовується для зберігання назв стовпців, яким відповідає поле зі значеннями \$values
\$wheres	Захищене поле, що використовується для зберігання інформацію про умову вибірки рядків
prepare	Відкритий метод, що призначений для об'єднання кожного з рядків запиту, які відповідають одному з типів запиту (TYPE_SELECT, TYPE_INSERT, TYPE_UPDATE, TYPE_DELETE)
executeQuery	Відкритий метод, що призначений для виконання запиту, який повністю сформований у вигляді рядка
select	Відкритий метод, що призначений для завдання стовпців, які необхідно отримати та завдання типу запиту у полі \$type
from	Відкритий метод, що призначений для завдання значення полю \$table
to	Відкритий метод, що є синонімом до методу from та частіше за все використовується при додаванні нових рядків
where	Відкритий метод, що призначений для завдання значень поля \$wheres, що слугує для завдання умови, по якій будуть отримані дані
orderBy	Відкритий метод, що призначений для сортування записів
take	Відкритий метод, що призначений для завдання полів \$limit, \$offset
first	Відкритий метод, що призначений для повернення першого знаходження, яке відповідає умові виборці
all	Відкритий метод, що призначений для повернення усіх рядків таблиці
last	Відкритий метод, що призначений для отримання останнього рядка з таблиці
insert	Відкритий метод, що призначений для додавання нового запису
update	Відкритий метод, що призначений для оновлення вже існуючого рядка
delete	Відкритий метод, що призначений для видалення вже існуючого рядка
getLastInsertId	Відкритий метод, що призначений для отримання id запису, який був доданий в останньому запиті
joinColumns	Захищений метод, що призначений для повернення рядку зі стовпців, які необхідно повернути

Продовження таблиці 2.9

Поле/Метод	Опис
getWhereValues	Захищений метод, що призначений для повернення запиту умови по якій необхідно виконати запит
compileSelect	Захищений метод, що призначений для об'єднання усіх рядків, що були надані у запиті
compileLimit	Захищений метод, що призначений для об'єднання полів \$limit, \$offset
compileWheres	Захищений метод, що призначений для складання запиту умови
compileInsert	Захищений метод, що призначений для складання запиту додавання нового запису
compileUpdate	Захищений метод, що призначений для складання запиту оновлення вже існуючого запису
compileDelete	Захищений метод, що призначений для складання запиту видалення вже існуючого запису
compileOrderBy	Захищений метод, що призначений для сортування записів

MysqlBuilder – клас, що розширює функціонал класу QueryBuilder, та надає інтерфейс для взаємодії з БД MySQL. Поля та методи класу наведені у таблиці 2.10.

Таблиця 2.10 – Поля та методи класу MysqlBuilder

Поле/Метод	Опис
\$connection	Захищене поле, що використовується для зберігання об'єкту, який отримує доступ до БД
__construct	Відкритий метод, що викликається під час створення об'єкта

Абстрактний клас Connection, що надає можливості створити з'єднання з БД, та надає об'єкт через який класи, що спадкуються від класу QueryBuilder, можуть створювати запити до БД.

Поля та методи класу наведені у таблиці 2.11.

Таблиця 2.11 – Поля та методи абстрактного класу Connection

Поле/Метод	Опис
pdo	Відкритий абстрактний метод, що призначений для отримання базового об'єкта PDO, для взаємодії з БД
query	Відкритий абстрактний метод, що призначений для отримання об'єкту класу QueryBuilder, для створення запитів
createTable	Відкритий абстрактний метод, що призначений для створення таблиці БД
alterTable	Відкритий абстрактний метод, що призначений для змінення таблиці
getTables	Відкритий абстрактний метод, що призначений для повернення таблиць, які присутні у поточній БД

Продовження таблиці 2.11

Поле/Метод	Опис
hasTable	Відкритий абстрактний метод, що призначений для перевірки чи існує таблиця у БД
dropTables	Відкритий абстрактний метод, що призначений для видалення таблиці з БД

`MysqlConnection` – клас, що надає можливості створити з'єднання з БД типу MySQL та розширює абстрактний клас `Connection`. Клас `MysqlConnection`, реалізує кожен метод з класу `Connection` та надає декілько власних методів та полів, описаних у таблиці 2.12.

Таблиця 2.12 – Поля та методи класу `MysqlConnection`

Поле/Метод	Опис
<code>\$pdo</code>	Закрите поле, що зберігає об'єкт класу PDO
<code>\$database</code>	Закрите поле, що зберігає назву БД з якою взаємодіє об'єкт цього класу
<code>__construct</code>	Відкритий метод, що викликається під час створення об'єкта та створює об'єкт цього класу шляхом отримання конфігурації з системи
<code>createPdo</code>	Відкритий метод, що створює об'єкт PDO та задає його полю <code>\$pdo</code>

Абстрактний клас `DriverProvider`, що представляє з себе базовий клас, який надає базові методи, які необхідно розширити при створенні класу провайдера, який буде створювати усі можливі типи з'єднання з БД. Поля та методи класу наведені у таблиці 2.13.

Таблиця 2.13 – Поля та методи абстрактного класу `DriverProvider`

Поле/Метод	Опис
<code>bind</code>	Відкритий метод, що додає нові можливості одного з провайдерів до вебзастосунку, які пізніше можна буде використовувати
<code>name</code>	Абстрактний, захищений метод, що надає назву провайдера при його використанні
<code>factory</code>	Абстрактний, захищений метод, що повертає об'єкт фабрики, який призначений для породження об'єктів, які надаються методом <code>drivers</code>
<code>drivers</code>	Абстрактний, захищений метод, що повертає масив елементів, за які відповідає провайдер

Клас `DatabaseProvider`, що розширює базовий клас `DriverProvider`, та реалізує методи базового класу для надання можливих типів з'єднання з БД. Поля та методи класу наведені у таблиці 2.14.

Таблиця 2.14 – Поля та методи класу `DriverProvider`

Поле/Метод	Опис
<code>name</code>	Захищений метод, що надає назву провайдера при його використанні
<code>factory</code>	Захищений метод, що надає об'єкт <code>Factory</code> , який призначений для породження об'єктів, які надаються методом <code>drivers</code>
<code>drivers</code>	Захищений метод, що повертає масив елементів з існуючих у системі типів з'єднання з БД

Інтерфейс `DriverFactory`, що призначений для того, щоб розробник мав представлення які методи необхідно реалізувати у похідному класі. Поля та методи класу наведені у таблиці 2.15.

Таблиця 2.15 – Поля та методи інтерфейсу `DriverFactory`

Поле/Метод	Опис
<code>addDriver</code>	Відкритий метод, що призначений для додавання нових об'єктів, які буде зберігати новий клас провайдер, який реалізує поточний інтерфейс
<code>connect</code>	Відкритий метод, що призначений для створення з'єднання з БД за вказаною конфігурацією системи, яка надається у якості параметра у вигляді масиву налаштувань, що повинен реалізувати похідний клас

Клас `Factory`, що призначений для створення вказаного об'єкта з'єднання з БД. Поля та методи класу наведені у таблиці 2.16.

Таблиця 2.16 – Поля та методи класу `Factory`

Поле/Метод	Опис
<code>\$drivers</code>	Захищене поле, яке зберігає у собі масив з існуючих функцій, які породжують з'єднання з БД
<code>addDriver</code>	Відкритий метод, що призначений для додавання нових об'єктів, які відповідають за з'єднання з БД
<code>connect</code>	Відкритий метод, що призначений для створення з'єднання з БД та повернення цього об'єкту

Клас `Application` представляє з себе головний клас системи, який створює більшість інших об'єктів, таких як: `Router`, `Request`, `Response`, `Controller`. Поля та методи класу наведені у таблиці 2.17.

Таблиця 2.17 – Поля та методи класу `Application`

Поле/Метод	Опис
<code>\$app</code>	Відкрите поле, що зберігає у собі екземпляр цього класу
<code>\$config</code>	Відкрите поле, що зберігає у собі масив налаштувань системи
<code>\$router</code>	Відкрите поле, що зберігає у собі об'єкт типу <code>Router</code>
<code>\$request</code>	Відкрите поле, що зберігає у собі об'єкт типу <code>Request</code>
<code>\$response</code>	Відкрите поле, що зберігає у собі об'єкт типу <code>Response</code>
<code>\$controller</code>	Відкрите поле, що зберігає у собі об'єкт типу <code>Controller</code>
<code>create</code>	Відкритий статичний метод, що задає усі базові об'єкти системи
<code>app</code>	Відкритий статичний метод, що повертає екземпляр цього ж класу
<code>getDb</code>	Відкритий статичний метод, що повертає об'єкт, що може звертатися до БД
<code>run</code>	Відкритий статичний метод, що запускає фреймворк

2.2.2 Допоміжні класи системи

Для спрощення розробки вебзастосунків були розроблені допоміжні класи, які описані у таблиці 2.18.

Таблиця 2.18 – Класи та їх призначення

Клас	Опис
<code>ArrayHelper</code>	Надає методи для обробки масивів
<code>StringHelper</code>	Надає методи для обробки рядків
<code>Command</code>	Базовий клас, що надає методи для обробки команд з командного рядка. Клас <code>Command</code> був створений командою розробників під назвою «Symfony»
<code>MigrateCommand</code>	Надає методи для обробки команд з командного рядка, до якого має доступ лише розробник
<code>BaseRule</code>	Надає методи для надання перевірки параметрів та структури класу, яку необхідно дотримуватися під час створення нового правила (похідного класу)
<code>Pagination</code>	Надає методи для створення сторінкової навігації (пагінація), які отримують дані, що будуть використані для створення відповідної пагінації
<code>DD</code>	Надає методи для налагодження програми, а зокрема виведення інформації про змінні, які передаються до обраного метода за допомогою параметрів
<code>Session</code>	Надає методи для роботи з масивом сесій
<code>FileHelper</code>	Надає методи для роботи з файлами
<code>Notification</code>	Надає методи для створення оповіщень
<code>Form</code>	Надає методи для створення елементів форм, таких як: текстове поле, велике текстове поле, випадний список
<code>View</code>	Клас, який створює об'єкт, який розширює інтерфейс <code>EngineInterface</code> , та надає його можливості

Продовження таблиці 2.18

Поле/Метод	Опис
EngineInterface	Інтерфейс, який призначений для створення шаблонізатора, та відображає методи, які необхідно реалізувати у похідному класі
EngineManager	Зберігає у собі масив методів, що можна використовувати у вигляді. Створює об'єкт типу View та задає до об'єкта, який розширює інтерфейс EngineInterface його основні методи для їх подальшого використання
ViewProvider	Призначений для надання методів, які можна використовувати у вигляді та типи шаблонізаторів

Клас `ArrayHelper` призначений для обробки масивів. Методи класу наведені у таблиці 2.19.

У інформатиці масив – це структура даних, що складається з набору елементів (значень або змінних), кожен із яких ідентифікується принаймні одним індексом чи ключем масиву.

Таблиця 2.19 – Методи класу `ArrayHelper`

Метод	Опис
<code>fillPropsFromArray</code>	Заповнює поля об'єкта даними з масиву, використовуючи ключі та значення з масиву, і повертає об'єкт
<code>isAssoc</code>	Перевіряє, чи є масив асоціативним чи ні, та повертає булеве значення
<code>getEncounteredItemPos</code>	Знаходить елемент у масиві та повертає номер його позиції, ключ або ж -1, якщо елемент не був знайдений
<code>strposa</code>	Перевіряє чи існують строки у масиві, та повертає булеве значення
<code>getIdValueArray</code>	Підготовлює масив з даними для HTML елемента <code>select</code> , що у свою чергу генерує випадковий список
<code>getArrayFullDifferenceByValue</code>	Повертає повну різницю між двома масивами за значенням
<code>getArrayFullDifferenceByKey</code>	Повертає повну різницю між двома масивами за ключем

Клас `StringHelper` призначений для обробки рядків. Методи класу наведені у таблиці 2.20.

Таблиця 2.20 – Методи класу `StringHelper`

Метод	Опис
<code>extractString</code>	Залежно від третього аргументу витягує рядкове значення до або після вказаного значення. Якщо входження не було знайдено, белеве значення «false» буде повернене
<code>endsWith</code>	Перевіряє, чи закінчується рядок вказаним значенням рядка

Клас `Command` призначений для обробки команд з командного рядка, до якого має доступ лише розробник. Даний клас є базовим, через що, його необхідно розширювати для створення власних команд, які будуть виконувати ті дії, які розробник опише у ньому. Клас `Command` – це клас який був створений командою розробників під назвою «Symfony». Більше детальну інформацію, стосовно цього класу можна продивитися за посиланням [64]. Поля та методи класу наведені у таблиці 2.21.

Таблиця 2.21 – Поля та методи класу `Command`

Поле/Метод	Опис
<code>\$defaultName</code>	Захищене поле, що зберігає у собі назву команди, по якій можна буде викликати команду з командного рядка
<code>configure</code>	Закритий метод, що задає базову конфігурацію команди (коротке та повне команди, параметри які команда може або повинна приймати)
<code>execute</code>	Закритий метод, який буде виконано під час виклику команди

Клас `MigrateCommand` призначений для обробки команд з командного рядка, до якого має доступ лише розробник. Розширює базовий клас `Command`. `MigrateCommand` призначений для знаходження усіх файлів міграцій та їх виконання. Поля та методи класу наведені у таблиці 2.22.

Таблиця 2.22 – Поля та методи класу `MigrateCommand`

Поле/Метод	Опис
<code>\$defaultName</code>	Захищене поле, що зберігає у собі назву команди «migrate», по якій можна буде викликати команду з командного рядка
<code>configure</code>	Закритий метод, що задає базову конфігурацію команди з не обов'язковою опцією «--fresh», яка дозволяє видалити усі таблиці БД та заново їх створити
<code>execute</code>	Закритий метод, що знаходить усі файли у каталозі, який задано у файлі конфігурації фреймворку. Кожен файл, що був знайдений, запускається на виконання, якщо цей файл відсутній у таблиці «migrations» БД
<code>connection</code>	Закритий метод, що створює об'єкт, за допомогою якого можна буде звертатися до БД
<code>createMigrationTable</code>	Закритий метод, що створює таблицю міграцій у БД

Абстрактний клас `BaseRule` призначений для надання базової перевірки параметрів та структури класу, яку необхідно дотримуватися під час створення

нового правила (похідного класу). Поля та методи класу наведені у таблиці 2.23.

Таблиця 2.23 – Поля та методи абстрактного класу BaseRule

Поле/Метод	Опис
validate	Відкритий метод, що виконує валідацію даних моделі, які були передані до цього методу, та повертає булеве значення. «Так», якщо усе вірно. «Ні», якщо дані не пройшли валідацію (перевірку)
getMessage	Відкритий метод, що повертає повідомлення у разі помилки, яку виявив метод validate
validateParams	Відкритий метод, що перевіряє наявність усіх необхідних параметрів
fieldNameToUpperCase	Відкритий метод, що видаляє «_» з назви поля, та використовує цю назву для показу полів, які містять помилки

Клас Pagation призначений для створення сторінкової навігації (пагінація), цей клас надає методи, які отримають дані, що будуть використані для створення відповідної пагінації. Поля та методи класу наведені у таблиці 2.24.

Таблиця 2.24 – Поля та методи класу Pagation

Поле/Метод	Опис
\$valid	Відкрите поле, що зберігає у собі булеве значення. «Так» - отримані дані є коректними. «Ні» – є помилка в отриманих параметрах
\$start	Відкрите поле, що зберігає у собі початкову сторінку, яка буде відображена після генерації пагінації
\$nbrsOfPageToDisplay	Відкрите поле, що зберігає у собі кількість сторінок, які необхідно показати у вигляді
\$linksCreated	Відкрите поле, що зберігає у собі кількість сторінок, які були згенеровані
\$middlePage	Відкрите поле, що зберігає у собі номер сторінки, який необхідно розмістити посередині пагінації
\$halfPages	Відкрите поле, що зберігає у собі булеве значення. «Так» - отримані дані є коректними. «Ні» – є помилка в отриманих параметрах
\$outputtedRecords	Захищене поле, що зберігає у собі кількість виведених рядків
\$offset	Захищене поле, масив, що зберігає два значення. Перше значення – з якого запису необхідно почати запит до БД. Друге значення – кількість записів, які необхідно отримати з БД
\$totalRecords	Захищене поле, що зберігає у собі кількість записів у БД
\$recordPerPage	Захищене поле, що зберігає у собі кількість записів, які необхідно відобразити на одній сторінці
\$currPage	Захищене поле, що зберігає у собі поточну сторінку
\$pages	Захищене поле, що зберігає у собі кількість можливих сторінок для відображення
\$prev	Захищене поле, що зберігає у собі номер попередньої сторінки
\$next	Захищене поле, що зберігає у собі номер наступної сторінки

Продовження таблиці 2.24

Поле/Метод	Опис
\$noResultsText	Захищене поле, що зберігає у собі текст, який необхідно відобразити, якщо записів у БД не було знайдено
init	Відкритий метод, що запускає обчислення отриманих даних
calculateTotalPages	Відкритий метод, що підраховує загальну кількість сторінок
trackPageNumber	Відкритий метод, що знаходить поточну, наступну та попередню сторінки
getCurrentPage	Відкритий метод, що повертає поточну сторінку
getRecordPerPage	Відкритий метод, що повертає перелік кількості сторінок на одну сторінку
checkPageLimitsToDisplay	Відкритий метод, що повертає перелік кількості сторінок на які можна перейти в один клац
isItCurrentPage	Відкритий метод, що перевіряє чи є поточка сторінка активною сторінкою
isFirstPage	Відкритий метод, що визначає чи є поточна сторінка першою
isLastPage	Відкритий метод, що визначає чи є поточна сторінка останньою сторінкою
getHtml	Відкритий метод, що повертає HTML для відображення пагінації
getPageUrl	Відкритий метод, що повертає адресу ресурсу, яку було згенеровано для певної сторінки
getPreviousPage	Відкритий метод, що повертає поле \$prev
getNextPage	Відкритий метод, що повертає поле \$next
doWeNeedPagination	Відкритий метод, що визначає чи можливо і чи є необхідність створювати пагінацію
setTotalRecords	Відкритий метод, що задає значення полю \$totalRecords
setRecordsPerPage	Відкритий метод, що задає значення полю \$recordPerPage
getOffset	Відкритий метод, що задає значення полю \$offset
setOffset	Відкритий метод, що повертає поле \$offset
setOutputtedRecords	Відкритий метод, що задає значення полю \$outputtedRecords
setNoResultsText	Відкритий метод, що задає значення полю \$noResultsText
setCurrentPage	Відкритий метод, що задає значення полю \$currPage
getPagesCount	Відкритий метод, що повертає поле \$pages
getRecordsCount	Відкритий метод, що повертає поле \$totalRecords
getNoResultsText	Відкритий метод, що повертає поле \$noResultsText
getInfo	Відкритий метод, що надає інформацію про те, скільки записів було виведено та їх позиції на сторінці (з N по M)
calculateHalfPageNumber	Відкритий метод, що автоматично обчислює яку сторінку можна розташувати посередині, якщо активною є сторінка N
calculateMiddlePageNumber	Відкритий метод, що автоматично обчислює перший номер сторінки
calculateOffset	Відкритий метод, що об'єднує у масив поле \$start та \$recordPerPage
validate	Відкритий метод, що перевіряє коректність отриманих даних з контролера

Клас DD призначений для налагодження програми, а зокрема виведення інформації про змінні, які передаються до обраного метода за допомогою параметрів. Поля та методи класу наведені у таблиці 2.25

Таблиця 2.25 – Поля та методи класу DD

Поле/Метод	Опис
\$startTime	Захищене поле, що зберігає у собі початок виконання коду у мс (10^{-3} с)
\$endTime	Захищене поле, що зберігає у собі кінець виконання коду у мс
dd	Відкритий статичний метод, що відображає інформацію про вміст змінної, про яку розробник хоче дізнатися більше інформації під час виконання коду. Метод, що показує дані з найменшою кількістю стилів для кращого сприйняття. Метод приймає три параметри. Перший – дані, які необхідно показати. Другий – відображає створений HTML код у вигляді тегів, або у вигляді тексту. Третій – чи треба завершувати виконання програми
dl	Відкритий статичний метод, що відображає інформацію про вміст змінної, про яку розробник хоче дізнатися більше інформації під час виконання коду, але цей метод, на відміну від dd, не завершує виконання та не має стилізованого виведення
startTimeTracking	Відкритий статичний метод, що фіксує початок виконання коду
getEndTime	Відкритий статичний метод, що фіксує завершення виконання коду та повертає час виконання коду у мс з чотирма знаками після коми
ddAnyArgs	Відкритий статичний метод, що містить у собі стилі для відображення даних метода dd
getStyles	Відкритий статичний метод, що зберігає стилі, які використовує метод dd

Session – клас, що надає методи для роботи з масивом сесій. Поля та методи класу наведені у таблиці 2.26.

Таблиця 2.26 – Поля та методи класу Session

Поле/Метод	Опис
FLASH_KEY	Відкрите поле, константа, що використовується для зберігання назви ключа повідомлень, які додаються під час роботи системи
__construct	Відкритий метод, що викликається під час створення об'єкта
set	Відкритий метод, що призначений для створення елементів сесії по типу ключ, значення (одному ключу відповідає одне значення)
getAll	Відкритий метод, що призначений для отримання усього масиву сесії
get	Відкритий метод, що призначений для отримання значення по його ключу
remove	Відкритий метод, що призначений для видалення значення по його ключу
destroy	Відкритий метод, що призначений для створення базового масиву сесій чи його отримання (необхідно викликати на початку роботи з масивом сесії)
initIfItDoesNotExist	Відкритий метод, що призначений для перевірки чи є будь-які повідомлення за ключем FLASH_KEY
hasAnyFlash	Відкритий метод, що призначений для отримання повідомлення за його ключем
setFlash	Відкритий метод, що призначений для усіх отриманих повідомлень, які зберігаються за ключем FLASH_KEY
getFlashByKey	Відкритий метод, що призначений для видалення повідомлення з підмасиву FLASH_KEY за його ключем
getAllFlashes	Відкритий метод, що призначений для створення елементів сесії по типу ключ, значення (одному ключу відповідає одне значення)
destroyFlashesWhere	Відкритий метод, що призначений для отримання усього масиву сесії

FileHelper – клас, що надає допоміжні методи для роботи з файлами. Методи класу наведені у таблиці 2.27.

Таблиця 2.27 – Методи класу FileHelper

Метод	Опис
removeFolderAndItsFiles	Відкритий статичний метод, що видаляє директорію та її файли, підкаталоги по вказаному шляху, який передається у якості параметра
tryToRemoveFoldersAndFiles	Відкритий статичний метод, що перевіряє чи існує директорія та файли у вказаній директорії та видаляє їх, якщо такі були знайдені
normalizeSlashes	Відкритий статичний метод, що видаляє надлишкові роздільники директорії, якщо такі були знайдені
removeFile	Відкритий статичний метод, що видаляє вказаний файл за його ім'ям та розширенням
canICreateADirectory	Відкритий статичний метод, що перевіряє чи можливо створити директорію за вказаним шляхом
isDirectoryExist	Відкритий статичний метод, що перевіряє чи існує директорія за вказаним шляхом
isFileInUse	Відкритий статичний метод, що перевіряє чи використовується вказаний файл
createDir	Відкритий статичний метод, що створює директорію за вказаним шляхом

Notification – клас, що надає методи для створення оповіщень. Поля та методи класу наведені у таблиці 2.28.

Таблиця 2.28 – Поля та методи класу Notification

Поле/Метод	Опис
ADMIN_VISIBLE	Відкрите поле, константа, яка говорить про те, що це повідомлення повинно бути відображено для адміністраторів сайту
USER_VISIBLE	Відкрите поле, константа, яка говорить про те, що це повідомлення повинно бути відображено для будь-якого користувача сайту
SUCCESS	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує зелений колір
WARNING	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує жовтий колір
DANGER	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує червоний колір
INFO	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує голубий колір
PRIMARY	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує синій колір
SECONDARY	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує сірий колір
DARK	Відкрите поле, константа, яка зберігає тип повідомлення, який відповідає за колір повідомлення. Дана константа використовує темносірий колір
TYPES	Відкрите поле, константа, яка зберігає масив усіх існуючих типів повідомлень

Продовження таблиці 2.28

Поле/Метод	Опис
\$type	Відкрите поле, що зберігає у собі тип створеного повідомлення
\$xTimes	Відкрите поле, що зберігає у собі кількість повідомлень, що дублюються (у разі якщо одне і те саме повідомлення було створено більше, ніж один раз, ця кількість буде відображена у повідомленні)
\$title	Відкрите поле, що зберігає у собі заголовок повідомлення
\$desc	Відкрите поле, що зберігає у собі основний текст повідомлення
\$closable	Відкрите поле, що зберігає у собі булеве значення – чи можливо закрити це повідомлення
\$visibility	Відкрите поле, що зберігає у собі значення ADMIN_VISIBLE чи USER_VISIBLE та використовується для фільтрування повідомлень

Form – клас, що надає методи для створення елементів форм, таких як: текстове поле, велике текстове поле, випадний список. Поля та методи класу наведені у таблиці 2.29.

Таблиця 2.29 – Поля та методи класу Form

Поле/Метод	Опис
\$HTML	Відкрите статичне поле, що зберігає у собі HTML розмітку форми
\$ERRORS	Відкрите статичне поле, що зберігає у собі помилки, якщо атрибути форми були задані некоректно
begin	Відкритий статичний метод, який відкриває HTML тег форми
end	Відкритий статичний метод, який закриває HTML тег форми
validateAttributes	Відкритий статичний метод, який перевіряє (валідує) параметри форми
input	Відкритий статичний метод, який виводить поле для вводу (тексту, чисел, файла)
textarea	Відкритий статичний метод, який виводить велике поле для вводу

UML (Unified Modeling Language) діаграма зв'язків основних класів необхідних для взаємодії зображено на рисунку 2.2.

Уніфікована мова моделювання (UML) є універсальною мовою в області розробки програмного забезпечення, призначеного для надання візуалізації проєктованої системи [47].

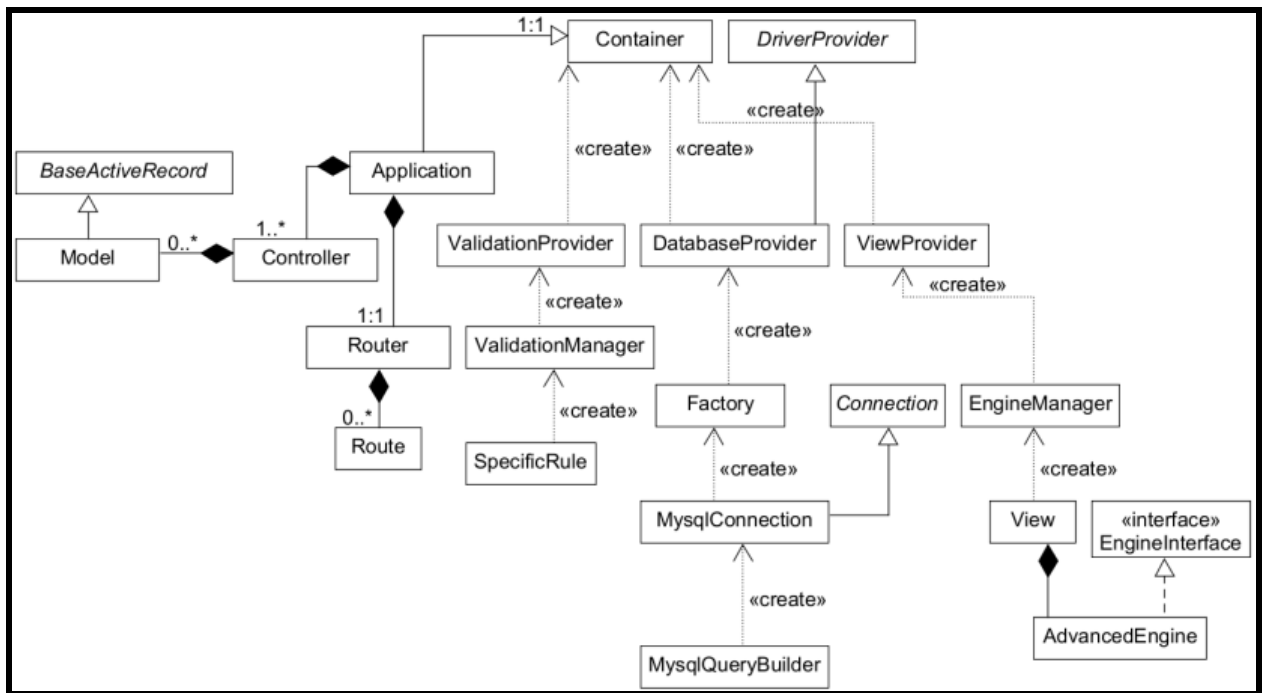


Рисунок 2.2 – UML-діаграма класів системи

Типи використаних зв'язків між класами:

- залежність – зміни в одному класі вносять зміни до іншого (зображується пунктирною лінією з стрілкою на її кінці);
- спадкування – один клас успадковує поля та методи іншого класу або перевизначають їх (зображується лінією з трикутником на її кінці);
- реалізація – визначає які методи необхідні реалізувати у класі, застосовується тільки до інтерфейсів (зображується пунктирною лінією з трикутником на її кінці);
- композиція – окремий випадок агрегації та призначений для зв'язків на частина-ціле (зображується лінією з ромбом на її кінці).

2.3 Основні розроблені алгоритми

Під час розробки фреймворку були розроблені алгоритми для обробки вбудованого у систему шаблонізатора, який надає базовий набір функцій для виведення даних, об'єднання виглядів та перетворення написаного шаблону на мову програмування РНР з використанням регулярних виразів.

На лістингу 2.1 приведений алгоритм генерації РНР файлів, що використовує шаблонізатор класу AdvancedEngine.

Лістинг 2.1 – Алгоритм генерації РНР файлів

```
public function getCachedOrCache(View $view)
{
    $hash = md5($view->path);
    $hashedFileName = "{$hash}.php";
    $folder = basePath() . '/storage/framework/views';
    if (FileHelper::canADirectoryBeCreated($folder)) {
        FileHelper::createDir($folder);
    }
    $hashedFileLocation = "{$folder}/{$hashedFileName}";
    if (!is_file($hashedFileLocation)) {
        touch($hashedFileLocation);
    }
    $cached = realpath($hashedFileLocation);
    $fileNotExistOrItsOutdated = !file_exists($hashedFileLocation) || filemtime($view->path) >
    filemtime($hashedFileLocation);
    if ($fileNotExistOrItsOutdated) {
        $content = $this->compile(file_get_contents($view->path));
        $content = "<!--$view->path-->\n" . $content;
        file_put_contents($cached, $content);
    }
    return $cached;
}
```

Алгоритм генерації РНР-файлів, що використовує шаблонізатор класу AdvancedEngine зображений на рисунку 2.3.

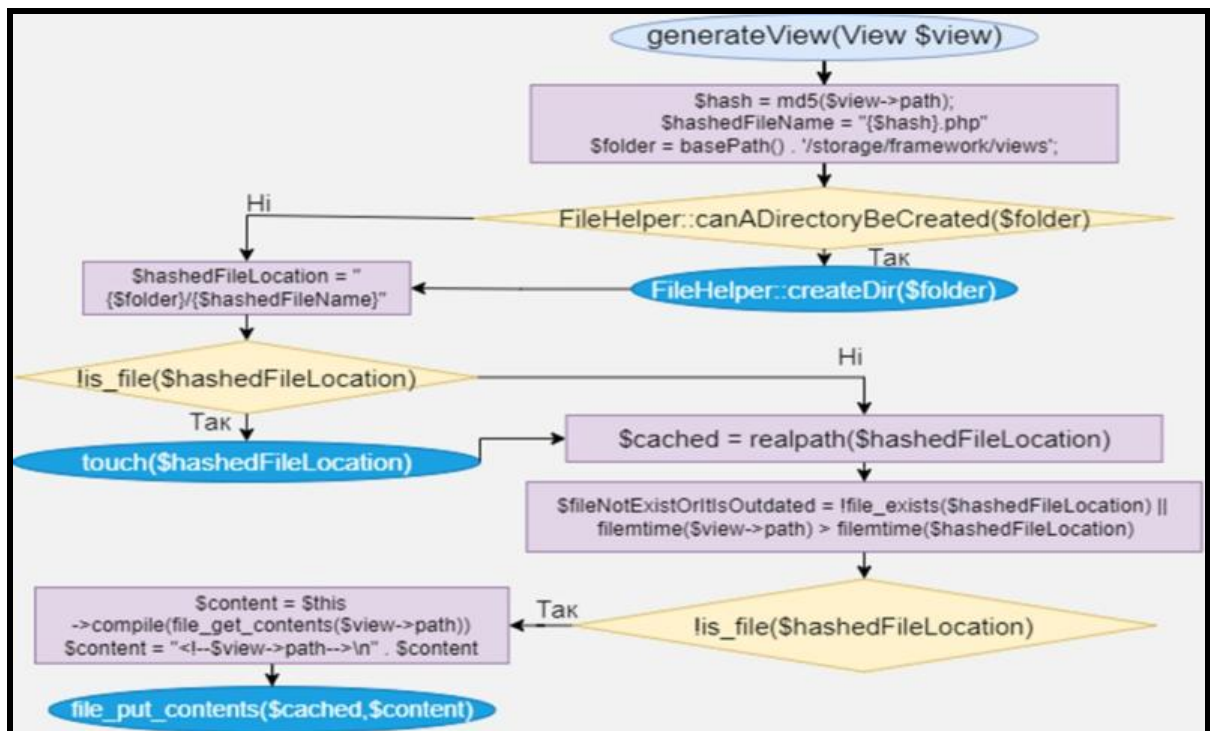


Рисунок 2.3 – Алгоритм генерації РНР-файлів класу AdvancedEngine

На рисунку 2.4 зображено порівняння коду з використанням шаблонізатора AdvancedEngine, зліва, та згенерований файл, який вже має усі необхідні команди для мови програмування PHP, справа.

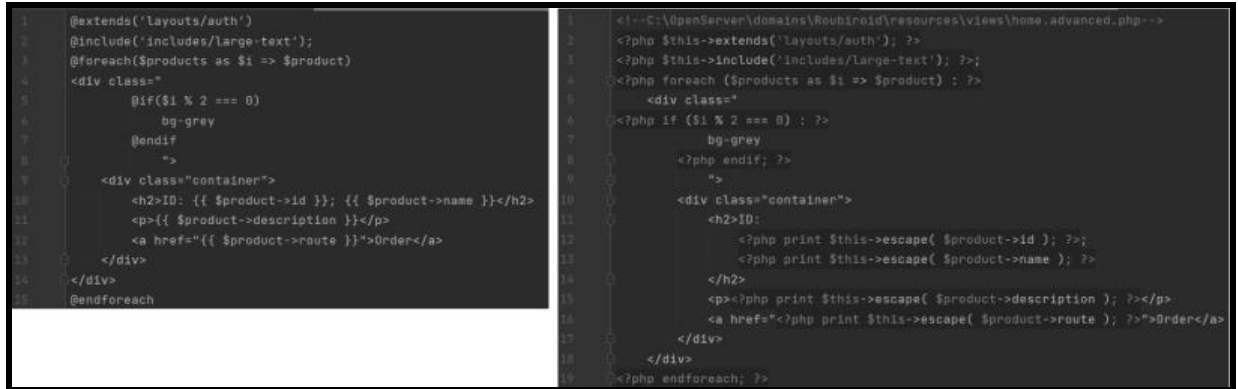


Рисунок 2.4 – Порівняння коду до та після використання шаблонізатора AdvancedEngine

Як можна побачити з рисунку 2.4, шаблонізатор AdvancedEngine значно полегшує написання коду шляхом скорочення конструкцій. Також такий варіант коду значно полегшує сприйняття написаного коду у разі, якщо ознайомитися з існуючими конструкціями.

На лістингу 2.2 приведений алгоритм заміни команд шаблонізатора AdvancedEngine на звичайні конструкції та методи мови програмування PHP.

Лістинг 2.2 – Алгоритм заміни команд

```

protected function compile(string $template): string
{
    $paramsRegExp = '\\(((?<=\\).*(?=\\)))\\';
    $template = preg_replace("#<!--.*-->#", "", $template);
    $template = preg_replace("#\\{\\{--.*--\\}\\}\\#", "", $template);
    $template = preg_replace_callback("#@extends{$paramsRegExp}#", function ($matches) {
        return PHP_EOL . '<?php $this->extends(' . $matches[1] . '); ?>';
    }, $template);
    $template = preg_replace_callback("#@if{$paramsRegExp}#", function ($matches) {
        return PHP_EOL . '<?php if (' . $matches[1] . ') : ?>';
    }, $template);
    $template = preg_replace_callback("#@endif#", function ($matches) {
        return '<?php endif; ?>';
    }, $template);
    $template = preg_replace_callback("#@else#", function ($matches) {
        return '<?php else : ?>';
    }, $template);
    $template = preg_replace_callback("#@foreach{$paramsRegExp}#", function ($matches) {
        return PHP_EOL . '<?php foreach (' . $matches[1] . ') : ?>';
    }, $template);
    $template = preg_replace_callback("#@endforeach#", function ($matches) {

```

```

        return '<?php endforeach; ?>';
    }, $template);
    $template = preg_replace_callback("#(?<=?>)[\s+\S+]@([^\(]+){$paramsRegExp}#", function ($matches)
    {
        return '<?php $this->' . $matches[1] . '(' . $matches[2] . ')'; ?>';
    }, $template);
    $template = preg_replace_callback('#\{([^\}]*?)\}#', function ($matches) {
        return '<?php print $this->escape(' . $matches[1] . ')'; ?>';
    }, $template);
    $template = preg_replace_callback('#\{!([^\}]+)!\}#', function ($matches) {
        return '<?php print ' . $matches[1] . ' '; ?>';
    }, $template);
    return $template . PHP_EOL;
}

```

На лістингу 2.3 приведений алгоритм заміни формальних параметрів фактичними, де значення «`{{key}}`» буде замінене на справжній параметр, що буде використаний у маршрутизаторі класу Router.

Лістинг 2.3 – Алгоритм заміни формальних параметрів фактичними

```

public static function route(string $name, array $parameters = []): string
{
    $methodExistIn = self::routeExistIn($name, 'post');
    $methodExistIn ??= self::routeExistIn($name, 'get');
    self::$redirectMethod = $methodExistIn;
    self::$redirectMethod ??= self::DEFAULT_REDIRECT_METHOD;
    foreach (self::$routes[self::$redirectMethod] as $route) {
        if ($route->name() !== $name) {
            continue;
        }
        $finds = [];
        $replaces = [];
        foreach ($parameters as $key => $value) {
            array_push($finds, "{{$key}}");
            array_push($replaces, $value);
            array_push($finds, "{{$key}}?");
            array_push($replaces, $value);
        }
        $path = $route->path;
        $path = str_replace($finds, $replaces, $path);
        if (strpos($path, '{') {
            throw new \Exception("Wrong view or parameters supplied to router");
        }
        $path = preg_replace("#\[^\]]+\}#", "", $path);
        return $path;
    }
    throw new \InvalidArgumentException("no route with that name " . $name . "");
}

```

2.4 Висновки за розділом 2

У другому розділі розроблена архітектура, алгоритми та класи фреймворку, які реалізують усі необхідні методи для виконання поставлених задач.

Розроблений власний клас для налагодження програми. Фреймворк створений в середовищі розробки PhpStorm 2021 з використанням об'єктно-орієнтованої мови програмування PHP версії 7.4, вебсервер Apache 2.4 та сервера БД MySQL 5.7, браузер Chrome.

Кожна з частин фреймворку повністю відповідає поставленим вимогам. Архітектурний шаблон MVC був реалізований найкращим чином завдяки використанню нових функцій PHP, що значно покращило швидкодію фреймворку.

3 ОСОБЛИВОСТІ ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

3.1 Система контролю версій

Контроль версій (Version Control) – це управління кількома версіями проєкту. Для управління версією слід відстежувати кожну зміну (додавання, редагування чи видалення) файлів у проєкті. Контроль версій записує кожну зміну, внесену у файл (або групу файлів), і надає можливості повернення видаленої інформації (коду). Для ефективного контролю версій потрібно використовувати інструменти під назвою «Git». Вони допомагають розробнику переходити між змінами та швидко повертатися до попередньої версії, коли щось невірно. Однією з найважливіших переваг використання Version Control є робота в команді. Коли кілька розробників беруть участь у розробці проєкту, відстеження змін стає доволі складним, і це значно збільшує ймовірність перезапису змін іншого розробника [65].

За допомогою контролю версій кілька людей можуть працювати над своєю копією проєкту (так званими гілками) і об'єднувати ці зміни в основний проєкт лише тоді, коли вони, або інші члени команди, задоволені роботою [66].

Команди до Git можна надсилати як з різноманітних графічних оболонок, таких як (gitk та git-gui розповсюджуються з самим Git), так і з командного рядка.

Терміни, які використовуються у системі контролю версій:

- commit (коміт) – зафіксований стан файлів і каталогів у сховищі;
- branch (гілка) – зберігає у собі посилання на коміт, але у вигляді назви, яку задає розробник;
- репозиторій – місце, де зберігаються і підтримуються будь-які дані;
- stash – сховище незавершених змін.

Усі команди повинні бути запущені з головного каталогу проєкту.

Основні команди для управління проектом:

- `git init` – створює робочий репозиторій у місці ініціалізації даної команди;
- `«git checkout -»` – надає змогу переключитися на останню активну гілку;
- `git checkout -b newBranchName` – створення нової гілки на основі вже існуючої;
- `«git add *»`, `«git add .»` – додає файли та каталоги, які необхідно зберегти;
- `«git push origin LBranchName:SBranchName»` – додає локальну гілку до віддаленого репозиторію з назвою `«SBranchName»`;
- `«git stash save -m "повідомлення розробника, зміна"»` – зберігає дані, які були додані за допомогою команди `«git add *»` або `«git add .»` у сховище незавершених змін;
- `git stash apply` – надає можливість отримати зміни зі сховища незавершених змін будь-якій гілці або коміті;
- `git stash list` – виводить список зі сховища незавершених змін;
- `git commit -m "повідомлення розробника, зміна"` – створює коміт, шляхом зберігання проіндексованих (доданих) файлів або каталгів;
- `git merge branchToMerge` – поєднує вибрану гілку у поточну;
- `git rebase branchNameTo` – робить те саме, що і `«git merge branchToMerge»`, але ще зберігає послідовність історії;
- `git branch -M newBranchName` – надає можливості для того, щоб перейменувати гілку;
- `git log` – використовується для перегляду історії комітів та гілок.

Git поставляється з вбудованим демоном команди `git`, який запускає простий TCP-сервер, що працює по протоколу Git. Виділені HTTP-сервери Git допомагають (серед інших функцій), додаючи контроль доступу, відображаючи вміст сховища Git через вебінтерфейс та дозволяє керувати кількома

репозиторіями (сабмодулями). Вже існуючі репозиторії Git можуть бути клоновані й надані іншим користувачам у якості централізованого репозиторію. До нього також можна отримати доступ через віддалену оболонку, просто встановивши програмне забезпечення Git і дозволивши користувачеві увійти в систему. Сервери Git зазвичай прослуховують TCP-порт 9418.

TCP (англ. Transmission Control Protocol, Протокол управління передачею) – разом із протоколом IP є основний протокол Інтернету, через що протокол TCP/IP і отримав таку назву. Протокол TCP/IP управляє передачею даних у комп'ютерних мережах та працює на транспортному рівні моделі OSI (англ. transport layer, транспортний рівень).

3.2 Особливості інтерфейсу системи

В програмній архітектурі може бути багато рівнів між апаратним забезпеченням та кінцевим користувачем. Кожен з цих рівнів може мати як front end, так і back end. Front end – це абстракція, спрощення базового компонента через надання користувачу зручного (user-friendly) інтерфейсу [65].

Інтерфейс, що надає помилки, повинен надавати вичерпну інформацію у разі допущення помилки за допомогою вигляду (повідомлення про помилку, рядок, де вона була допущена, та деяка частина файлу, назва файлу), тільки у тому разі, якщо встановлений режим розробки (на реальному сервері будь-які помилки, пов'язані з кодом, повинні буди приховані).

Інтерфейс відображення виглядів:

- надавати вичерпну інформацію у разі допущення помилок користувача під час заповнення даних форми;
- можливість створення повідомлень трьох типів (вдало завершено – зеленого кольору, щось могло бути пропущено – жовтого та червоного кольору, якщо щось не було знайдено, або не було надано системі), які будуть відображатися користувачам або адміністратору в залежності від налаштувань повідомлення;

- можливість створення форм (великих текстових полів, випадяючого списку, кнопок, радіокнопок, прапорці, вибір файлу), таблиць;
- можливість створення будь-яких елементів, що присутні у Bootstrap 5.

Усе це робить графічний інтерфейс користувача чуйним, надає детальну інформацію у разі виникнення будь-якої помилки.

3.3 Експлуатаційні вимоги

Для роботи фреймворка на ПК повинно бути встановлено наступне програмне забезпечення:

- мова програмування PHP версії 7.4 або новіше;
- вебсервер Apache версії 2.4 або новіше;
- менеджер залежностей – composer 2.0 або новіше;
- сервер БД – MySQL версії 5.7 чи MariaDB версії 10.2 або новіше;

Мінімальні системні вимоги для роботи фреймворку:

- процесор Intel або AMD, що має два ядра або більше;
- права доступу до CLI;
- 2 ГБ ОП;
- операційна система, що сумісна з PHP версії 7.4 або новіше;
- як мінімум 15 МБ вільного місця на накопичувачі.

3.4 Тестування системи та аналіз результатів

Для налагодження програми використовувалось розширення для браузера Chrome під назвою xDebug та клас DD, що надає інформації про змінні, які передаються до обраного метода за допомогою параметрів. Тестування кожного модуля ПП проводилось вручну. Також було використано програмне забезпечення під назвою Apache JMeter для перевірки стабільності

роботи в залежності від кількості запитів та кількості користувачів, які одночасно перебувають на одній сторінці. Було проведено тестування швидкості за допомогою власного модуля DD, що допомагає вимірювати час виконання блоку кода, який буде знаходитися між методами `startTimeTracking` та `getEndTime`.

Система тестувалася під час розробки нових класів за допомогою xDebug. Також був створений окремий клас DD, який допомагав з налагодженням програми шляхом відображення детальної інформації про наступні параметри:

- повний шлях до файлу та / або класу та його назва;
- назва методу або функції, яка була викликана;
- рядок, в якому був викликаний метод налагодження класу DD;
- форматування отриманих даних;
- стилізація для кращого сприйняття інформації.

Шляхом багаторазових проходжень сценаріїв, які з'являлися по мірі розробки фреймворку, проводилось регресійне тестування.

Також були використані можливості PhpStorm для отримання даних про певні об'єкти в певний момент часу за допомогою розширення xDebug у браузері Chrome.

Під час проходження тестування усі можливі сторонні програми були вимкнені або закриті.

Для кожного тесту були створені максимально схожі умови. Після того, як один тест було завершено, БД повністю видалялася та створювалася заново для чистоти експерименту.

У програмі Apache JMeter були виставлені наступні налаштування:

- кількість користувачів, які будуть одночасно перебувати на сторінці – 360;
- час, за який кількість користувачів збільшиться до вказаного значення – одна секунда;

- кількість циклів виконання алгоритму дорівнює десяти.

Для порівняння фреймворків Yii2, Laravel та створеного фреймворка були проведені наступні тести:

- стрес-тестування за допомогою програми Apache JMeter;
- тестування швидкості за допомогою власного модуля DD, що допомагає вимірювати час виконання блоку кода, який буде знаходитися між методами `startTimeTracking` та `getEndResultTime`.

Тестування проводилося на ПК з наступними характеристиками:

- 32 ГБ оперативної пам'яті (ОП) за частотою 3200 МГц;
- 8 ГБ віртуальної пам'яті;
- жорсткий диск samsung 970 PRO 1TB (700 ГБ вільного місця на диску);
- процесор Intel Core i7 8700k з шістьма фізичними ядрами та 12 потоками;
- ОС – Windows 10 Pro 21H1.

Кожен тест, який запускав Apache JMeter, створював повний об'єкт додатка, що показує не тільки швидкодію створення запитів до БД, але й швидкість створення самого об'єкта та передачі цього запиту до БД.

Тестування швидкості за допомогою власного модуля DD проводилося наступним чином:

- додавалися нові записи до БД у кількості «10000»;
- зчитувалися вже існуючі записи з БД у кількості «10000».

Порівняння швидкості обробки запитів на вибірку до БД з використанням програми Apache JMeter зображено на рисунку 3.1.



Рисунок 3.1 – Порівняння швидкості обробки запитів на вибірку до БД з використанням програми Apache JMeter

Порівняння швидкості обробки запитів на вставку до БД з використанням програми Apache JMeter зображено на рисунку 3.2.



Рисунок 3.2 – Порівняння швидкості обробки запитів на вибірку до БД з використанням програми Apache JMeter

Тест, який показує швидкість створення запиту та отримання відповіді від БД використовуючи клас DD, що дозволяє виміряти швидкість виконання блоку коду. Порівняння швидкості виконання запиту до БД без урахування створення об'єкта вебдодатка зображено на рисунку 3.3.

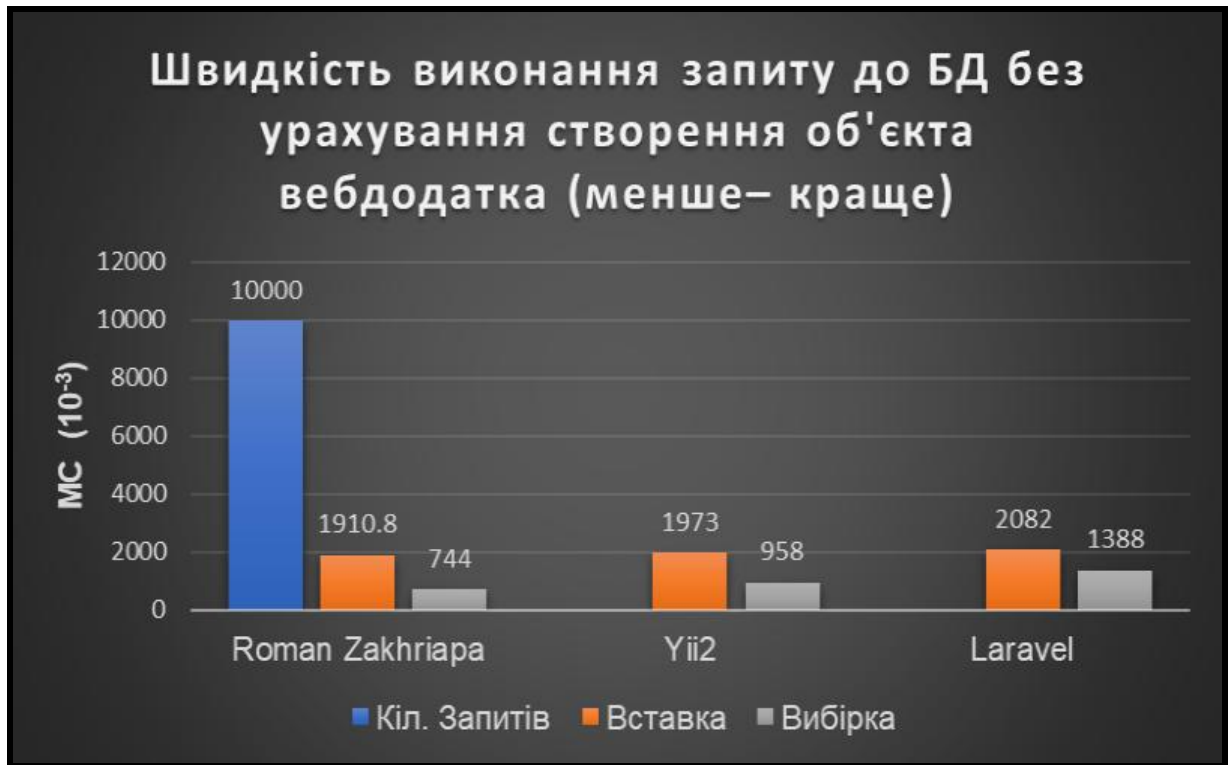


Рисунок 3.3 – Порівняння швидкості виконання запиту до БД без урахування створення об'єкта вебдодатка

На рисунку 3.4 порівнюється відносна швидкість вибірки, вставки та кількості помилок у відсотках. 100% – одна помилка. Кількість перевірок дорівнювала 18000 для отримання даних. Тест проводився у програмі Apache JMeter. Дані, які використовуються для створення діаграми наведені у таблиці 3.1.



Рисунок 3.4 – Відносне порівняння фреймворків з урахуванням часу на створення головного об'єкта фреймворку

Таблиця 3.1 – Відносне порівняння фреймворків

Фреймворк	Швидкість вибірки	Швидкість вставки	Помилки
Laravel	16%	15%	981%
Yii2	46%	41%	1455%
Roman Zakhriapa	100%	100%	100%

Порівнявши створений фреймворк з вже існуючими, ми можемо зробити висновок, що даний фреймворк обходить за швидкістю роботи вже існуючі за:

- вибіркою записів з БД на 54% у порівнянні з Yii2 та на 84% у порівнянні з Laravel;
- вставкою нових записів до БД на 59% у порівнянні з Yii2 та на 85% у порівнянні з Laravel;
- розроблений фреймворк має до десяти разів менше помилок ніж Laravel, та до 14 разів у порівнянні з Yii2.

На рисунку 3.5 зображене відносне порівняння фреймворків з використанням класу DD.



Рисунок 3.5 – Відносне порівняння фреймворків з використанням класу DD

Розглянувши рисунок 3.5, можна зробити висновок, що розроблений фреймворк оброблює запити на вибірку швидше ніж Yii2 на 22% та ніж Laravel на 46%. Порівнявши швидкість обробки запитів на вставку, отримаємо наступний результат: фреймворк Yii2 поступається лише на 3%, у той час, як Laravel на 8%.

Швидкодія фреймворку пояснюється тим, що його рівень захисту трохи нижчий, ніж в аналогів. Аналоги мають багато модулів, але не кожен з них використовується, але при цьому вони існують та збільшують час виконання усього коду. А кількість модулів у розробленого фреймворку значно менша, але самі модулі не менш якісні. Це можна пояснити тим, що інші фреймворки розробляють вже понад сім років, а кількість людей, які беруть участь у розробці, перевищує 500 фахівців і з кожним днем продовжує зростати.

Після закінчення розробки основних функцій було проведено тестування вручну. Завдяки створеному класу DD та розширенню xDebug були знайдені недоліки фреймворку, які були спочатку детально описані за допомогою тест-кейсів та баг-репортів, а пізніше виправлені. Також під час тестування були виявлені деякі неоптимізовані частини коду, які також були покращені шляхом

використання вже існуючих функцій мови програмування PHP, та скорочена кількості операцій, які було необхідно виконати для досягнення мети.

Під час тестування не було виявлено значних недоліків, тому можна вважати, що тестування фреймворку проведено успішно. Фреймворк виконує усі описані функції, що знаходяться у постанові завдання, тому перевірка на системні вимоги пройшла успішно.

3.5 Висновки за розділом 3

У третьому розділі визначені експлуатаційні вимоги, які необхідно дотримуватися для коректної роботи ПЗ. Для більш гнучкої та надійної розробки була обрана розподілена система контролю версій Git.

Після проведення тестування, можна зробити висновки, що фреймворк Yii2 та Laravel поступаєть вже розробленому фреймворку, як під час стрес-тестування за допомогою програми Apache JMeter, так і під час тестування за допомогою класу DD.

Розроблений показує кращі результати не тільки по часу виконання запитів до БД, алей й за кількістю помилок, які були допущені під час стрес-тестування у порівнянні з Yii2 і Laravel, та має до 14 разів менше помилок.

Тестування програмного продукту показало, що фреймворк є не тільки працездатним, але й навіть може показати кращий результат у порівнянні з деякими, вже існуючими аналогами, що свідчить про те, що даний фреймворк коректно реалізований та готовий до реальної експлуатації.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Системні вимоги до персонального комп'ютера для підтримки чи подальшої розробки програмного продукту

Для роботи фреймворка на ПК повинно бути встановлено наступне програмне забезпечення:

- мова програмування PHP версії 7.4 або новіше;
- вебсервер Apache версії 2.4 або новіше;
- менеджер залежностей – composer 2.0 або новіше;
- сервер БД – MySQL версії 5.7 чи MariaDB версії 10.2 або новіше;

Мінімальні системні вимоги для роботи фреймворку:

- процесор Intel або AMD, що має два ядра або більше;
- права доступу до CLI;
- 2 ГБ ОП;
- операційна система, що сумісна з PHP версії 7.4 або новіше;
- як мінімум 15 МБ вільного місця на накопичувачі.

4.2 Налаштування системи перед початком її роботи

Для встановлення програмного продукту необхідно завантажити архів з фреймворком, розархівувати його за допомогою будь-якої програми архіватора, наприклад, 7-Zip у директорію, за яку відповідає ваш поточний вебсервер Apache.

Після того, як архів був розархівований у потрібній вам директорії, необхідно відкрити командний рядок у кореневому каталозі фреймворка та запустити одну з команд встановлення залежностей.

Кореневим каталогом системи, вважається той каталог, у який ви розархівували файли фреймворку.

Пакети залежностей, які необхідні для роботи фреймворку, знаходяться у файлі під назвою «composer.json». Секція «require» містить у собі усі залежності, які необхідні для роботи системи під час експлуатації. Секція «require-dev», описує ті залежності, які необхідно обов’язково встановити під час розробки системи.

Усі команди «composer» необхідно запускати з кореневого каталогу фреймворка.

Можливі варіанти встановлення залежностей:

- «composer install --no-dev» – встановлює лише ті компоненти, які потрібні для роботи фреймворку під час його експлуатації;
- «composer install» – встановлює усі залежності, як для експлуатації, так і для розробки, які можуть надати додаткову інформацію під час розробки та знаходження існуючих помилок у системі.

Після того, як ви запустите одну з описаних вище команд, вам буде необхідно заповнити змінні середовища у файлі «.env».

Також система містить зразковий файл під назвою «.env.example», який необхідно скопіювати та перейменувати на «.env», чи тільки перейменувати на «.env» для того, щоб система могла знайти його та використати у якості файлу зі змінами середовища.

Класи, які розширюють клас Comand, придатні до використання лише з командного рядка. Кожен новий клас повинен бути доданий до масиву у файл «commands.php», вказуючи повну назву класу.

Для використання похідних класів Comand необхідно використовувати командний рядок з кореневого каталогу системи. Для того, щоб отримати допомогу стосовно взаємодії з командним рядком, використовуйте команду: «php command.php --help». Ця команда надасть список головних можливостей, де розробник може переглянути як взаємодіяти з командним рядком, так і з команди, які вже існують у системі. Для отримання інформації про команду, використовуйте наступну команду «php command.php commandName --help», де «commandName» – це команда, стосовно якої ви отримуєте інформацію.

4.3 Звертання до програми

Після того як команда «php command.php migrate --fresh» виконана, буде створені дві таблиці у БД, це «migrations» та «users». Структура таблиць зображена на рисунку 4.1.



rz_framework users	rz_framework migrations
id : int(11) unsigned	id : int(11) unsigned
name : varchar(255)	name : varchar(255)
email : varchar(255)	created_at : datetime
password : varchar(255)	

Рисунок 4.1 – Таблиці БД «migrations» та «users», які будуть створені після запуску команди «php command.php migrate --fresh»

Більшість роботи з фреймворком буде відбуватися у директоріях під назвою «app» та «resources», що знаходяться у кореневому каталозі.

Для створення відображення використовується функція «view», що приймає у якості першого параметра шлях до вигляду, який повинен знаходитися у директорії «resources/views», або у будь-якій дочірній директорії від цього шляху, також у якості другого параметру можна передати необхідні змінні до вигляду, які можуть бути використані для їх відображення

Для отримання більш повної інформації стосовно кожної директорії використовуйте другий розділ під назвою «Структура та архітектура проєкту», у якому детально описана кожна з них.

Для створення авторизації користувача у системі необхідно виконати наступні кроки:

- створити контролер у директорії «app/Http/Controllers» під назвою «AuthController» (лістинг 4.1);
- створити модель, яка буде відповідати за входження користувача до системи та перевіряти коректність даних, які ввів користувач за наступним шляхом «app/Http/Models» під назвою «LoginForm» (лістинг 4.2);

- створити модель у директорії «app/Http/Models» під назвою «User», яка буде відповідати таблиці БД «users» (лістинг 4.3);
- для захисту використовується функція «csrf», що генерує унікальний ідентифікатор, по якому користувач зможе успішно виконати запит до сервера;
- для перевірки унікальності ідентифікатора використовується функція «secure»;
- створити вигляд під назвою «main.php» у директорії «resources/views/layouts» для підключення головного макету сторінки та її стилів (лістинг 4.4);
- створити вигляд, який буде виводити форму для введення даних за наступним шляхом «resources/views» під назвою «login.advanced.php» (лістинг 4.5);
- додати назву «AuthController» контролера та його метода до файлу за наступним шляхом «app/routes/web.php» (лістинг 4.6).

Лістинг 4.1 – Контролер під назвою «AuthController»

```
class AuthController extends Controller
{
    public function login(Request $request)
    {
        $loginForm = new LoginForm();
        if ($request->isPost()) {
            secure();
            $loginForm->load($_POST);
            if ($loginForm->validate() && $loginForm->login()) {
                redirect('/');
            }
        }
        return view('login', [
            'loginAction' => $this->router->route('log-in-user'),
            'model' => $loginForm,
        ]);
    }
}
```

Лістинг 4.2 – Модель під назвою «LoginForm»:

```
class LoginForm extends Model
{
    protected $validationGeneralDescription = 'Виправте помилки та повторіть спробу';
    protected $validationTitle = 'Дані введені не вірно';
    protected array $fillable = ['email', 'password'];
    public function rules(): array
    {
        return [
            'email' => Rules::EMAIL,
            [['email', 'password'], Rules::REQUIRED],
            [['email'], Rules::MIN_STR, Rules::MIN_STR => 3],
        ];
    }
}
```

```

        [['password'], Rules::MIN_STR, Rules::MIN_STR => 8],
    ];
}
public function login(): bool
{
    $user = User::where('email', '=', $this->email)->first();
    if (!$user) {
        $this->addError('email', 'Your password or email is incorrect');
        return false;
    }
    if (!password_verify($this->password, $user->password)) {
        $this->addError('password', 'Password is incorrect');
        return false;
    }
    return Session::set('user', $user->primaryKey());
}
public function labels(): array
{
    return ['email' => 'Електронна Адреса', 'password' => 'Пароль'];
}
}

```

Лістинг 4.3 – Модель під назвою «User»:

```

class User extends Model
{
    public string $table = 'users';
    public array $fillable = ['name', 'email', 'password'];
    public static function primaryKey(): string
    {
        return 'id';
    }
}

```

Лістинг 4.4 – Код вигляду для входу до системи

```

<?php $this->title = 'Login';?>
@extends('layouts/main')
<?php
$form = Form::begin($loginAction, 'post', [
    'enctype' => Form::ENCTYPE_DEFAULT,
    'class' => 'custom-class'
]); ?>
<input type="hidden" name="csrf" value="{{ csrf() }}" />
<h1>Вхід до системи</h1>
<?=$form->input($model, 'email', [
    'type' => 'email',
    'class' => 'custom-class',
    'placeholder' => 'Електронна адреса'
    'note' => 'Використовуйте свій email для входу,
$form->label(); ?>
<?=$form->input($model, 'password', [
    'class' => 'custom-class',
    'placeholder' => 'Пароль'
$form->passwordField(true)->label(); ?>
<button type="submit" class="btn btn-primary">Вхід</button>
<?=$form->end(); ?>

```

Лістинг 4.5 – Код вигляду головного шаблону фреймворка

```

<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/css/bootstrap-5.1.css">
    <link rel="stylesheet" href="/css/index.css">
    <title>Auth Layout</title>

```

```

</head>
<body>
<div class="container">
  AUTH layout 1
  <?php require_once Paths::views('notification/notification.php'); ?>
  {!! $contents !!}
</div>
</body>
</html>

```

Лістинг 4.6 – Код, що додає створений метод контролера до масиву маршрутів

```

Router::get('/new-login', [AuthController::class, 'newLogin']->name('log-in-user-form'));
Router::post('/new-login', [AuthController::class, 'newLogin']->name('log-in-user'));

```

Зображення вигляду, який відповідає за вхід до системи, зображено на рисунку 4.2.

Рисунок 4.2 – Форма для входу до системи

Для входу до системи необхідно натиснути кнопку «Вхід».

Після підтвердження даних користувача відбувається їх валідація за наступними умовами:

- перевіряється чи заповнені поля;
- чи відповідає поле «Електронна Адреса» формату email;
- чи має поле «Електронна Адреса» як мінімум три символи та поле «Пароль» як мінімум вісім символів.

Якщо при натисканні кнопки «Вхід», форма не проходить валідацію даних, то у вигляді будуть відображені помилки. Вигляд, що відображає помилки вводу зображено на рисунку 4.3.

Дані введені некоректно ✕

Виправте помилки та повторіть спробу:

- Електронна Адреса
- Пароль

Вхід до системи

Електронна Адреса

Вкажіть як мінімум 3 символів

Пароль

Вкажіть як мінімум 8 символів

Вхід

Рисунок 4.3 – Форма, яка відображає помилки вводу

У моделі «LoginForm» під час спроби увійти до системи перевіряється чи був знайдений користувач у БД, який ввів свої дані. Якщо дані введено коректно, то користувача буде переадресовано до головної сторінки системи. Якщо дані введені некоректно, то буде відображено нове повідомлення, що зображено на рисунку 4.4.

Вхід до системи

Електронна Адреса

Електронна Адреса або пароль введені некоректно

Пароль

Вхід

Рисунок 4.4 – Форма, що відображає помилку, яка повідомляє користувача про те, що його дані не є коректними

Для того, щоб отримати список створених міграцій у системі з БД і вивести їх до вигляду, необхідно виконати наступні кроки:

- створити модель під назвою «Migrations», яка повинна розширювати базовий клас «BaseActiveRecord» та вказати назву таблиці у полі «\$table» для моделі «Migrations» (лістинг 4.7);
- створити відкритий метод у контролері, який буде отримувати дані з моделі та передавати їх до вигляду під назвою «list-migrations», який повинен бути створений у директорії «resources/views» (лістинг 4.8);
- створити контролер, або використовувати вже існуючий, у якості прикладу буде створений новий контролер під назвою «ExampleController» (лістинг 4.9);
- додати назву «ExampleController» контролера та його метода до файлу за наступним шляхом «app/routes/web.php» (лістинг 4.10).

Лістинг 4.7 – Код моделі «Migrations»

```
class Migrations extends BaseActiveRecord
{
    protected string $table = 'migrations';
}
```

Лістинг 4.8 – Код вигляду «list-migrations»

```
@extends('layouts/main')
<h1>Список виконаних міграцій</h1>
<ol>
    @foreach($migrations as $m)
        <li>Назва міграції: <span class="badge bg-secondary">{{ $m->name }}</span></li>
    @endforeach
</ol>
```

Лістинг 4.9 – Код класу «ExampleController»

```
class ExampleController extends Controller
{
    public function showMigrations()
    {
        $migrations = Migrations::all();
        return view('migrations', [
            'migrations' => $migrations,
        ]);
    }
}
```

Лістинг 4.10 – Код, що додає створений метод контролера до масиву маршрутів та вказує за яким посиланням необхідно перейти для того, щоб побачити перелік:

```
Router::get('/', [ExampleController::class, 'showMigrations']);
```

Вигляд, що виводить список виконаних міграцій у системі із таблиці БД під назвою «migrations», зображений на рисунку 4.5.

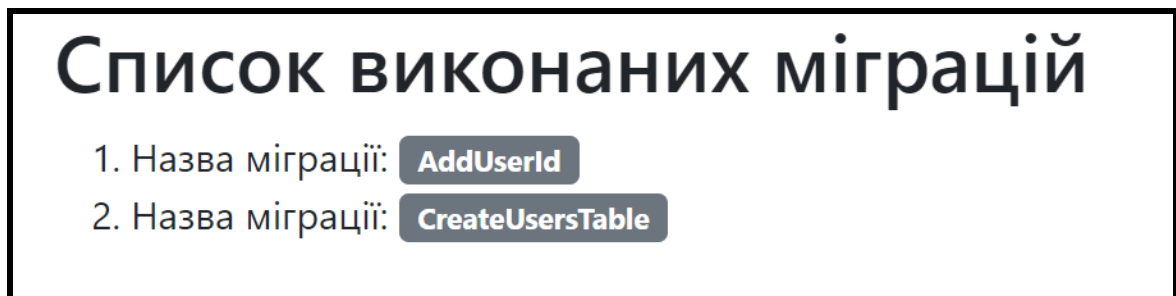


Рисунок 4.5 – Перелік, що відображає виконані міграції

Метод під назвою «findUserId», що додає нового користувача до БД, але перед тим як зберігати його до БД необхідно:

- створити модель класу «User»;
- записати до неї дані з масиву «\$data»;
- зробити валідацію даних;
- зашифрувати пароль користувача;
- зберегти його до БД.

Метод «findUserId» описаний у контролері «ExampleController»:

```
public function addNewUser()
{
    $data = ['name' => 'Ім'я Користувача', 'email' => 'useremail@gmail.com', 'password' =>
'AnyPassword000'];
    $user = new User();
    $user->load($data);
    if ($user->validate()) {
        $user->password = password_hash($user->password, \PASSWORD_DEFAULT);
        $user->save();
    }
}
```

Метод під назвою «findUserId», що знаходить користувача з БД за його унікальним ідентифікатором «id». Для отримання унікального ідентифікатора використовується клас «Router», що зберігає надіслані дані з запиту типу «GET» до маршруту «/update-user-by-id/{id}», де «{id}», це унікальний ідентифікатор.

Метод описаний у контролері «ExampleController»:

```

public function findUserById()
{
    $getData = Router::getActiveRoute()->parameters();
    $id = $getData['id'] ?? null;
    if (is_null($id)) {
        redirect('back');
    }
    $user = User::find($id);
}

```

Метод під назвою «updateUserEmailById», що оновлює поле «електронної адреси» користувача, валідує, нові дані та зберігає їх до БД. Метод описаний у контролері «ExampleController»:

```

public function updateUserEmailById()
{
    $getData = Router::getActiveRoute()->parameters();
    $id = $getData['id'] ?? null;
    $user = User::find($id);
    $user->email = 'updatedemail@gmail.com';
    if ($user->validate()) {
        $user->save();
    } else {
        redirect('home');
    }
}

```

Для того, щоб вищеописані методи почали процювати, необхідно додати до файлу «app/routes/web.php» наступний код:

```

Router::get('/add-new-user', [ExampleController::class, 'addNewUser']);
Router::get('/find-user-by-id/{id}', [ExampleController::class, 'findUserById']);
Router::get('/update-user-email-by-id/{id}', [ExampleController::class, 'updateUserEmailById']);

```

Для того, щоб скористатися одним із описаних методів необхідно перейти за одним із посилань, яке буде оброблено контроллером під назвою «ExampleController»:

- your-application-name/add-new-user – для того, щоб додати нового користувача з даних, що описані у контролері «ExampleController» під методом addNewUser»;

- your-application-name/find-user-by-id/1 – для того, щоб знайти користувача за унікальним ідентифікатором «1», необхідно скористатися методом методом «findUserById»;

– `your-application-name/update-user-email-by-id/1` – для того, щоб змінити «електронну адресу» користувачу з унікальним ідентифікатором «1» іншу, що описана у контролері «`ExampleController`» під методом «`updateEmailById`».

4.4 Висновки за розділом 4

У четвертому розділі описано:

- призначення й основні можливості фреймворку;
- встановлення та налаштування фреймворку для початку його роботи;
- зазначено у яких директоріях необхідно створювати, контролери, вигляди та моделі, а також їх взаємодія. За допомогою створених класів можна створювати вебзастосунки, які можуть взаємодіяти з БД, виводити повідомлення користувачам системи;
- визначені мінімальні вимоги для ПК, на якому фреймворк може працювати без помилок;
- визначені мінімальні вимоги до програмного забезпечення, яке необхідно для коректної роботи фреймворку;
- надано інструкцію по створенню нових виглядів, контролерів та моделей;
- продемонстровані методи для взаємодії з БД, які дозволяють оновлювати, видаляти та додавати нові записи до БД.

Цей розділ призначений для ознайомлення з базовими можливостями фреймворку, які він надає.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи магістра були вирішені наступні задачі:

- вивчена та проаналізована предметна область;
- обрані інструменти та технології, за допомогою яких було створене програмне забезпечення;
- найкращим чином реалізований архітектурний шаблон MVC, завдяки використанню нових функцій PHP, що значно покращило швидкодію фреймворку;
- підготовлені файли «.htaccess», які дозволяють використовувати цей фреймворк на будь-якому сервері, що підтримує PHP версії 7.4, або новіше, та Apache 2.4, або новіше;
- розроблений власний клас DD для налагодження програми;
- розроблена програма.

Для тестування фреймворку використані наступні інструменти:

- клас DD, який вимірює швидкість виконання блоку коду;
- стрес-тестування за допомогою програми Apache JMeter, яке показало, що даний фреймворк має кращі результати у порівнянні з вже існуючими аналогами, що свідчить про те, що даний фреймворк коректно реалізований та готовий до реальної експлуатації.

У якості додаткових частин фреймворку розроблено декілька класів, які демонструють його можливості.

Розроблений фреймворк можна використовувати для створення вебзастосунків, яким не потрібна велика кількість надлишкових класів та методів, але потрібна гнучка та потужна система, що спроможна при мінімальних затратах ресурсів системи надавати найкращі результати по обробці кількості запитів будь-якого типу.

Кожна з частин фреймворку повністю відповідає поставленим вимогам, а отже всі поставлені задачі виконані і мета досягнута.

Користуючий програмою для стрес-тестування Apache JMeter було створенно навантаження на систему, яке показало, що створений фреймворк має кращі показники за:

- вибіркою записів з БД на 54% у порівнянні з Yii2 та на 84% у порівнянні з Laravel;
- вставкою нових записів до БД на 59% у порівнянні з Yii2 та на 85% у порівнянні з Laravel;
- розроблений фреймворк має до десяти разів менше помилок ніж Laravel, та до 14 разів у порівнянні з Yii2.

Під час тестування за допомогою класу DD отримали наступні результати:

- за кількістю вибірок з БД, розроблений фреймворк показав кращі результати ніж Yii2 на 22% та ніж Laravel на 46%;
- за кількістю вставок до БД, розроблений фреймворк показав кращі результати на 3% ніж Yii2 та на 8% ніж Laravel.

Розроблений показує кращі результати не тільки по часу виконання запитів до БД, алей й за кількістю помилок, які були допущені під час стрес-тестування у порівнянні з Yii2 і Laravel, та має до 14 разів менше помилок.

Практична цінність роботи полягає в тому, що розроблена гнучка система для підвищення ефективності розробки вебзастосунків, яка дозволяє прискорити створення вебзастосунків шляхом використання вбудованого шаблонізатора, що значно пришвидшує створення виглядів та їх інтерпретацію, а також покращити розуміння коду завдяки інтуїтивним назвам методів та класів. Також корисна властивість розробленого фреймворку полягає у належному ступіні захисту інформації. Фреймворк використовує методи, які реалізовані у мові програмування PHP, що значно впливають на швидкодію системи в цілому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Describing why php is the best choice [Electronic resource]. – Access mode: <https://light-it.net/blog/why-use-php-main-advantages-and-disadvantages>.
2. Murach J. murach's PHP and MySQL, Second Edition [Text] / J. Murach, R. Harris. – Fresno : Associates, Inc. 2010 – 872 p.
3. Usage statistics of PHP for websites [Electronic resource]. – Access mode: <https://w3techs.com/technologies/details/pl-php>.
4. Веб-застосунок [Електрон. ресурс]. – Режим доступу: <http://wiki.kubg.edu.ua/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA>.
5. What is frameworks? [Electronic resource]. – Access mode: <https://techterms.com/definition/framework>.
6. Знакомство с фреймворками. Часть 1. HTML/CSS, PHP и Python [Электрон. ресурс]. – Режим доступа: <https://timeweb.com/ru/community/articles/znakomstvo-s-freymvorkami-chast-1-html-css-php-i-python-1>.
7. Why do people need a software framework? [Electronic resource]. – Access mode: https://www.wikiwand.com/en/Software_framework.
8. The difference between a software framework and a library [Electronic resource]. – Access mode: <https://www.wikiwand.com/ru/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA>.
9. Каркас архитектуры [Электронный ресурс]. – Режим доступа: https://ru.wikiversity.org/wiki/%D0%9A%D0%B0%D1%80%D0%BA%D0%B0%D1%81_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D1%8B.
10. Cross-site request forgery [Electronic resource]. – Access mode: https://www.wikiwand.com/en/Cross-site_request_forgery.

11. MVC Design Pattern [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/mvc-design-pattern/>.

12. Об'єктно-реляційна проєкція [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D0%B2%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F?oldformat=true.

13. Міграція бази даних [Електрон. ресурс]. – Режим доступу: https://www.wikiwand.com/uk/%D0%9C%D1%96%D0%B3%D1%80%D0%B0%D1%86%D1%96%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.

14. Authentication versus Authorization [Electronic resource]. – Access mode: <https://stackoverflow.com/questions/6556522/authentication-versus-authorization>.

15. What is PDO? [Electronic resource]. – Access mode: <https://www.drupal.org/docs/7/system-requirements/what-is-pdo>.

16. Використання RESTful [Електрон. ресурс]. – Режим доступу: <http://edu.asu.in.ua/mod/book/view.php?id=117&chapterid=257>.

17. Предметно-орієнтована мова програмування [Електрон. ресурс]. – Режим доступу:

https://www.wikiwand.com/uk/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%BE%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.

18. HMVC [Електрон. ресурс]. – Режим доступу: <https://www.wikiwand.com/uk/HMVC>.

19. HTML [Електрон. ресурс]. – Режим доступу: <https://www.wikiwand.com/uk/HTML>.

20. Браузер [Електрон. ресурс]. – Режим доступу:
<https://www.wikiwand.com/uk/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80>

21. Вебсервер [Електрон. ресурс]. – Режим доступу:
<https://www.wikiwand.com/uk/%D0%92%D0%B5%D0%B1%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80>

22. HTTP [Електрон. ресурс]. – Режим доступу:
<https://www.wikiwand.com/uk/HTTP>

23. Гіпертекст [Електрон. ресурс]. – Режим доступу:
<https://www.wikiwand.com/uk/%D0%93%D1%96%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82>

24. CSS [Electronic resource]. – Access mode:
<https://www.wikiwand.com/en/CSS>.

25. Main page of Yii2 [Electronic resource]. – Access mode:
<https://yiiframework.com/>.

26. Main page of laravel [Electronic resource]. – Access mode:
<https://laravel.com/>.

27. Pros and Cons of Laravel [Electronic resource]. – Access mode:
<https://nimapinfotech.com/blog/pros-and-cons-of-laravel>.

28. Advantages and disadvantages of Laravel Framework for web Development [Electronic resource]. – Access mode: <https://ddi-dev.com/blog/programming/pros-and-cons-of-laravel-framework-for-web-app-development/>.

29. Main page of Symfony [Electronic resource]. – Access mode:
<https://symfony.com/>.

30. Symfony Introduction [Electronic resource]. – Access mode:
<https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/introduction.html>.

31. Using Symfony for a wide variety of projects [Electronic resource]. – Access mode: <https://www.trustradius.com/products/symfony/reviews?qs=pros-and-cons>.

32. The Good and the Bad of Symfony Web Development [Electronic resource]. – Access mode: <https://www.developers.dev/tech-talk/technology/the-good-and-the-bad-of-symfony-web-development.html>.

33. The basic outline of Symfony's pros and cons [Electronic resource]. – Access mode: <https://sapient.pro/blog/symfony-pros-and-cons-and-examples-of-projects/>.

34. Main page of CodeIgniter [Electronic resource]. – Access mode: <https://codeigniter.com/>.

35. Pros and Cons of CodeIgniter Framework <https://www.mindfiresolutions.com/blog/2018/03/pros-cons-codeigniter-framework/>.

36. PHP 2020 [Electronic resource]. – Access mode: <https://www.wikiwand.com/en/PHP>.

37. Беляков, А. Ю. Объектно-ориентированное программирование / А. Ю. Беляков. – Пермь: ИПЦ «Прокрость», 2017. – 87 с.

38. Schildt H. C++ from the Ground Up, Third Edition [Text] / H. Schildt. – Berkeley : Brandon A. Nordin, 2003. – 602 p.

39. Looking At The PHP 8.0 Performance So Far In Early 2020 [Electronic resource]. – Access mode: https://www.phoronix.com/scan.php?page=news_item&px=PHP-8.0-Performance-Bench-2020.

40. IntelliJ IDEA [Electronic resource]. – Access mode: https://www.wikiwand.com/en/IntelliJ_IDEA.

41. PhpStorm [Electronic resource]. – Access mode: <https://www.wikiwand.com/en/PhpStorm>.

42. Інтерфейс командного рядка [Електрон. ресурс]. – Режим доступу: https://www.wikiwand.com/uk/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D1%80%D1%8F%D0%B4%D0%BA%D0%B0.

43. PhpStorm vs Visual Studio Code [Electronic resource]. – Access mode: <https://stackshare.io/stackups/phpstorm-vs-visual-studio-code>.

44. PhpStorm Features [Electronic resource]. – Access mode: <https://www.jetbrains.com/phpstorm/features/>.

45. Install PhpStorm [Electronic resource]. – Access mode: <https://www.jetbrains.com/help/phpstorm/installation-guide.html/>.

46. Composer (software) [Electronic resource]. – Access mode: [https://www.wikiwand.com/en/Composer_\(software\)](https://www.wikiwand.com/en/Composer_(software)).

47. Zandstra, M. PHP 8 Objects, Patterns, and Practice, Sixth Edition [Text] / M. Zandstra – Brighton : Apress, 2021 – 856 p.

48. World Wide Web Consortium [Electronic resource]. – Access mode: https://www.wikiwand.com/en/World_Wide_Web_Consortium.

49. Introduction to SQL [Electronic resource]. – Access mode: https://www.w3schools.com/sql/sql_intro.asp/.

50. Seyed M.M. “Saied” Tahaghoghi Hugh E. Williams Learning [Text] / M.M. Seyed “Saied” Tahaghoghi E. Hugh Williams – Sebastopol : O’Reilly Media, Inc. 2007 – 600 p.

51. Елементи теорії баз даних [Електрон. ресурс]. – Режим доступу: <http://comp.ucoz.net/Subd/DustNavchInf15.pdf>.

52. About Business Components [Electronic resource]. – Access mode: https://docs.oracle.com/cd/B31104_02/books/ConfigApps/ConfigApps_BusComp2.html.

53. Delisle M. Creating your MySQL Database: Practical Design Tips and Techniques [Text] / M. Delisle Birmingham : Packt Publishing, 2007 – 94 p.

54. Chintan M. Ankit B. Hetal O. Subhash S. MySQL 8 Administrator's Guide [Text] / M. Chintan B. Ankit O. Hetal S. Subhash. Birmingham : Packt Publishing Ltd, 2018 – 729 p.

55. Database [Electronic resource]. – Access mode <https://www.wikiwand.com/en/Database>.

56. What is Bootstrap and Why you should use it in Web Development [Electronic resource]. – Access mode: <https://www.yogihosting.com/what-is-bootstrap/>.

57. HTTPS [Electronic resource]. – Access mode: <https://www.wikiwand.com/en/HTTPS>.

58. Communications security [Electronic resource]. – Access mode: https://www.wikiwand.com/en/Communications_security.

59. Конфігураційний файл .htaccess [Електрон. ресурс]. – Режим доступу: <https://hostiq.ua/wiki/ukr/htaccess/>.

60. Active record pattern [Electronic resource]. – Access mode: https://www.wikiwand.com/en/Active_record_pattern.

61. Regular expression [Electronic resource]. – Access mode: https://www.wikiwand.com/en/Regular_expression.

62. Формальна мова [Електрон. ресурс]. – Режим доступу: https://www.wikiwand.com/uk/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0.

63. URL [Electronic resource]. – Access <https://www.wikiwand.com/en/URL>.

64. Console Commands [Electronic resource]. – Access <https://symfony.com/doc/current/console.html>.

65. Git [Електрон. ресурс]. – Режим доступу: <https://www.wikiwand.com/uk/Git>.

66. Tsitoara, M. Beginning Git and GitHub [Text] / M. Tsitoara – Antananarivo : Mariot Tsitoara, 2020. – 289 p.

ДОДАТОК А
Текст програми

A.1 Текст файла Application.php

```
<?php
/**
 * Class Response
 *
 * @author Roman Zakhriapa
 * <utmostcreator@gmail.com>
 * @package Framework
 */

namespace Framework;
use Framework\db\Database;
use Framework\helpers\Config;
use Framework\helpers\StringHelper;
use Framework\routing\Router;
class Application extends Container
{
    public static Application $app;
    public static ?array $config = null;
    public string $userClass;
    public Router $router;
    public Request $request;
    public Response $response;
    public Controller $controller;
    public Session $session;
    public Database $db;
    public static function create()
    {
        Session::initIfItDoesNotExist();
        $app = new static();
        $config = Config::get();
        static::$config = $config;
        $app->userClass = $config['userClass'] ?? '';
        $app->layout = $config['layout']['value'] ?? '';
        $app->view = new View();
        $app->request = new Request();
        $app->response = new Response();
        $app->session = new Session();
        $app->router = Router::getInstance($app-
>request, $app->response);
        static::$app = $app;
        $defConnection = $config['default'];
        $dbConfig = $config['connections'][$defConnection];
        $app->db = new Database($dbConfig);
        $primaryValue = $app->session->get('user');
        if ($primaryValue) {
            $primaryKey = $app-
>userClass::primaryKey();
            $app->user = $app-
>userClass::findOne([$primaryKey =>
$primaryValue]);
        }
    }
}
```

```
$basePath = basePath();
$app->bindProviders($basePath);
}
public static function app(): Application
{
    try {
        if (static::$app) {
            return static::$app;
        }
        throw new \Exception("Application is not
yet defined!");
    } catch (\Exception $exception) {
        exit($exception->getMessage());
    }
}
public static function getInstance()
{
    return self::$app;
}
public static function getDb()
{
    return self::$app->db;
}
public function run()
{
    $page = $this->router->resolve();
    echo $page;
}
public function login(UserModel $user): bool
{
    $this->user = $user;
    $primaryKey = $user->primaryKey();
    $primaryValue = $user->{$primaryKey};
    $this->session->set('user', $primaryValue);
    return true;
}
private function bindProviders($basePath)
{
    $providerDir = StringHelper::normalizeSlashes(Config::get('provi
ders.directory'));
    $providers = require
sprintf("$basePath/%sproviders.php",
$providerDir);
    foreach ($providers as $provider) {
        $instance = new $provider();
        if (method_exists($instance, 'bind')) {
            $instance->bind($this);
        }
    }
}
```

A.2 Текст файла Request.php

```
<?php
namespace Framework;
class Request
{
    private ?string $uri = null;
    private ?array $urlParts = [];
    private string $requestPath;
    public bool $hasParams = false;
    private string $method = 'get';
    private ?string $referer = null;
    public function __construct()
    {
        $this->uri =
rawurlencode($_SERVER['REQUEST_URI']);
        $this->urlParts = isset($this->uri) ?
parse_url($this->uri) : [];
        $this->referer =
$_SERVER['HTTP_REFERER'] ?? null;
        $this->method =
isset($_SERVER['REQUEST_METHOD']) ?
strtolower($_SERVER['REQUEST_METHOD']) :
'get';
        $this->requestPath = $this->uri ?? '/';
        if ($this->urlParts && strpos($this->requestPath, '?')) {
            $this->requestPath = substr($this->requestPath, 0, strpos($this->requestPath, '?'));
        }
        $this->hasParams = $this->urlParts['query']
?? false;
    }
    public function getPath()
    {
        $path = $this->uri ?? '/';
        if (!$this->hasParams) {
            return $path;
        }
        return $this->requestPath;
    }
}
```

```
public function getMethod(): string
{
    return $this->method;
}
public function isGet(): bool
{
    return $this->method === 'get';
}
public function isPost(): bool
{
    return $this->method === 'post';
}
public function getBody()
{
    $body = [];
    if ($this->isGet()) {
        foreach ($_GET as $key => $value) {
            $body[$key] = filter_input(INPUT_GET,
$key, FILTER_SANITIZE_SPECIAL_CHARS);
        }
    }
    if ($this->isPost()) {
        foreach ($_POST as $key => $value) {
            $body[$key] =
filter_input(INPUT_POST,
$key, FILTER_SANITIZE_SPECIAL_CHARS);
        }
    }
    return $body;
}
public function getUri(): string
{
    return $this->uri;
}
public function refererPage(): string
{
    return $this->referer ?? "";
}
}
```

A.3 Текст файла Response.php

```
<?php
namespace Framework;
class Response
{
    public const SERVER_INTERNAL_ERROR =
500;
    public const CLIENT_NOT_FOUND = 404;
    public const CLIENT_BAD_REQUEST = 400;
    public const REDIRECT_OTHER_REDIRECT
= 303;
    public const REDIRECT_MOVED_TEMPORARILY = 302;
    public const REDIRECT_MOVED_PERMANENTLY = 301;
    public function setStatusCode(int $code)
    {
        http_response_code($code);
    }
    public static function redirectForeverTo($path =
'/')
    {
        header('Location: ' . URL::getBase(false) .
$path, $replace = true, $response_code = 301);
        exit;
    }
    public function redirect(string $url, int
$statusCode = 303): void
    {
        switch ($url) {
            case 'home':
            case '':
            case '/':
                header('Location: ' . URL::getBase(),
true, $statusCode);
```

```
                break;
            case 'error':
                throw new
\InvalidArgumentException('Error; Incorrect URL
is specified');
                break;
            case 'forbidden':
                $statusCode = 403;
                header('Location: ' . URL::getBase() .
'forbidden.php', true, $statusCode);
                break;
            case 'back':
                $isTheSameUrl = URL::getBase(false) .
app()->request->getUri();
                if (!empty(app()->request->getUri()) &&
$isTheSameUrl !== app()->request-
>referrerPage()) {
                    header('Location: ' . app()->request-
>referrerPage(), true, $statusCode);
                } else {
                    header('Location: ' .
URL::getBase(false), true, $statusCode);
                }
                header("Refresh:0");
                break;
            default:
                header('Location: ' .
URL::getBase(false) . $url, true, $statusCode);
                break;
        }
    }
    exit;
}
```

A.4 Текст файла Router.php

```
<?php
namespace Framework\routing;
use Framework\Application;
use Framework\helpers\StringHelper;
use Framework\middlewares\AuthMiddleware;
use Framework\middlewares\BaseMiddleware;
use Framework\Request;
use Framework\Response;
use Framework\validation\ValidationException;
use Modules\DD;
class Router
{
    public const DEFAULT_REDIRECT_METHOD
= 'get';
    public Request $request;
    public Response $response;
    public static string $redirectMethod =
self::DEFAULT_REDIRECT_METHOD;
    protected ?Route $activeRoute = null;
    protected static ?Router $instance = null;
    protected static array $routes = [];
    private static array $rules = [];
    private function __construct(Request $request,
Response $response)
    {
        $this->request = $request;
        $this->response = $response;
    }
    public static function getInstance(Request
$request, Response $response): Router
    {
        if (is_null(static::$instance)) {
            return static::$instance = new
static($request, $response);
        }
        return static::$instance;
    }
    public static function get(string $path,
$callback): Route
    {
        return static::addNew(__FUNCTION__,
$path, $callback);
    }
    public static function post(string $path,
$callback): Route
    {
        return static::addNew(__FUNCTION__,
$path, $callback);
    }
    /* @description is designed for pages like 404,
403, 400, 500...
    * as they must be independent of other routes
    * may be rename it to "errorHandler"?
    */
    public static function addSystem(string $path,
\Closure $method): void
    {
        $route = self::$routes[$path] = $method;
    }
}
```

```
    public static function addNew(string $method,
string $path, $callback): Route
    {
        if (!empty(static::$rules)) {
            self::calculateAccessRules($method,
$path, $callback);
        } else {
            static::checkIfRouteAlreadyAdded($method,
$path);
            if (is_array($callback)) {
                static::$routes[$method][$path] = new
Route($method, $path, new
RouteHandler($callback));
            } elseif (is_callable($callback)) {
                static::$routes[$method][$path] =
$callback;
            }
        }
        return static::$routes[$method][$path];
    }
    public static function group(array $array,
\Closure $fillInTheSuppliedRoutes): void
    {
        static::$rules = $array;
        $fillInTheSuppliedRoutes();
        static::$rules = [];
    }
    private static function
calculateAccessRules(string $method, string
$path, $callback): void
    {
        $allowedAccessNames = ['role', 'permission',
'rules'];
        $defaultMiddleWares = ['auth'];
        if (!is_array(static::$rules['middleware'])) {
            if (in_array(static::$rules['middleware'],
$defaultMiddleWares)) {
                if (static::$rules['middleware'] === 'auth')
                {
                    $auth = new AuthMiddleware();
                    $auth->execute();
                }
                if (static::$rules['middleware'] ===
'something else') {
                }
                $middlewareClassNamePostfix =
'Middleware';
                $middlewareClassName =
ucfirst(static::$rules['middleware'])
.$middlewareClassNamePostfix;
                $class =
class_exists($middlewareClassName) ? new
$middlewareClassName() : null;
                if ($class && is_a($class,
BaseMiddleware::class)) {
                    $class->execute();
                }
            }
            return;
        }
    }
}
```

```

    }
}
foreach (static::$rules['middleware'] as $rule)
{
    $searchedVal = ':';
    $accessName =
StringHelper::extractString($rule, $searchedVal);
    $ruleName =
StringHelper::extractString($rule, $searchedVal,
false);
    if (!in_array($accessName,
$allowedAccessNames)) {
        throw new
\InvalidArgumentException('Invalid Access Name
is specified!');
    }
    static::checkIfRouteAlreadyAdded($path,
$method);
    if (

isset(static::$routes['accessControls'][$method])
&&

isset(static::$routes['accessControls'][$method][$a
ccessName]) &&

isset(static::$routes['accessControls'][$method][$a
ccessName][$ruleName]) &&
        in_array($path,
static::$routes['accessControls'][$method][$acces
sName][$ruleName])
    ) {
        $msg = "An attempt to add the same
route twice. Route name: [$path]; METHOD:
[$method] in GROUP";
        throw new
\InvalidArgumentException($msg);
    }

    static::$routes['accessControls'][$method][$acces
sName][$ruleName][] = $path;
    static::$routes[$method][$path] = new
Route($method, $path, new
RouteHandler($callback));
}
}
/* TODO to be removed - unused */
public static function middleware(string $string)
{
}
private static function
checkIfRouteAlreadyAdded(string $method, string
$path)
{
    if (isset(static::$routes[$method]) &&
in_array($path,
array_keys(static::$routes[$method]))) {
        $msg = "This Route is already registered!
Route: [$path]; METHOD: [$method]";
        throw new
\InvalidArgumentException($msg);
    }
}

```

```

}
/* @description this is used for named routes
mechanic
*/
public static function route(string $name, array
$parameters = []): string
{
    $methodExistIn = self::routeExistIn($name,
'post');
    $methodExistIn ??=
self::routeExistIn($name, 'get');
    self::$redirectMethod = $methodExistIn;
    self::$redirectMethod ??=
self::DEFAULT_REDIRECT_METHOD;
    foreach (self::$routes[self::$redirectMethod]
as $route) {
        if ($route->name() !== $name) {
            continue;
        }
        $finds = [];
        $replaces = [];
        foreach ($parameters as $key => $value) {
            array_push($finds, "{{${key}}}");
            array_push($replaces, $value);
            array_push($finds, "{{${key}}?}");
            array_push($replaces, $value);
        }
        $path = $route->path;
        $path = str_replace($finds, $replaces,
$path);
        if (strpos($path, '{') {
            throw new \Exception('Wrong view or
parameters supplied to router');
        }
        return $path;
    }
    throw new \InvalidArgumentException('no
route with that name "' . $name . '"');
}
private static function routeExistIn(string
$name, string $method)
{
    return in_array($name, array_map(fn($el) =>
$el->name(), self::$routes[$method])) ? $method :
null;
}
public function resolve()
{
    $this->setCurrentActiveRoute();
    return $this->resolveController();
}
public static function getRoutes()
{
    return Router::$routes;
    DD::dd(Router::$routes);
}
private function setCurrentActiveRoute(): void
{
    $path = $this->request->getPath();
    $method = $this->request->getMethod();
    $callback = $this->match($method, $path) ??
false;
}

```

```

        if ($callback === false) {
            abort(404);
        }
        $this->activeRoute = $callback;
    }
    public static function getActiveRoute(): Route
    {
        return static::$instance->activeRoute;
    }
    private function resolveController()
    {
        if (!$this->activeRoute) {
            abort(500);
        }
        $controller = $this->activeRoute-
>getActiveController();
        Application::$app->controller = $controller;
        foreach ($controller->getMiddlewares() as
        $middleware) {
            $middleware->execute();
        }
        try {
            return $this->dispatch([$controller,
            $controller->action], [$this->request, $this-
            >response, $this->activeRoute]);
        } catch (\Throwable $e) {
            if ($e instanceof ValidationException) {
                $_SESSION['errors'] = $e->getErrors();
                return redirect('back');
            }
            if (isDev()) {
                throw $e;
            }
        }
    }
    public function dispatch(array $callback, array
    $params)
    {
        if (!is_callable($callback)) {

```

```

            throw new
            \InvalidArgumentException("Invalid
            callback
            passed to dispatch");
        }
        return call_user_func_array($callback,
        $params);
    }
    private function match(string $method, string
    $path): ?Route
    {
        foreach (static::$routes[$method] as $route) {
            if ($route instanceof Route && $route-
            >matches($method, $path)) {
                return $route;
            }
        }
        return null;
    }
    public function redirect($path)
    {
        header("Location: {$path}", true, 301);
        exit;
    }
    public function errorHandler(int $code, callable
    $handler)
    {
        static::$routes[$code] = $handler;
    }
    public function dispatchNotAllowed()
    {
        static::$routes[400] ??= fn() => "not allowed";
        return (static::$routes[400])();
    }
    public function dispatchError()
    {
        static::$routes[500] ??= fn() => "server error";
        return (static::$routes[500])();
    }
}

```

A.5 Текст файла Route.php

```
<?php
namespace Framework\routing;
use Framework\BaseObject;
use Framework\helpers\StringHelper;
use Framework\Controller;
class Route extends BaseObject
{
    protected array $parameters = [];
    protected string $method;
    protected string $path;
    protected ?string $name = null;
    protected RouteHandler $handler;
    private int $priority;
    * @param string $method
    * @param string $path
    * @param RouteHandler $handler [TODO it
can \Closure or callable, but then a simple path to
a view file would not work!]
    */
    public function __construct(string $method,
string $path, ?RouteHandler $handler)
    {
        $this->method = $method;
        $this->path = $path;
        $this->handler = $handler;
    }
    public function name(string $name = null)
    {
        if ($name) {
            $this->name = $name;
            return $name;
        }
        return $this->name;
    }
    public function matches(string $method, string
$path): bool
    {
    }
    public function normalisePath(string $path):
string
```

```

    {
        $path = trim($path, '/');
        $path = "{/$path}"/";
        $path = preg_replace('/[V]{2,}', '/', $path);
        return $path;
    }
    public function className()
    {
        return $this->handler->className;
    }
    public function methodName()
    {
        return $this->handler->methodName;
    }
    public function getActiveController():
?Controller
    {
        $className = $this->className();
        $methodName = $this->methodName();
        if (!class_exists($className) ||
(class_exists($className) &&
!method_exists($className, $methodName))) {
            abort(500);
        }
        $controller = new $className();
        $controller->action = $methodName;
        return $controller;
    }
    public function parameters(): array
    {
        return $this->parameters;
    }
    public function priority(int $value): Route
    {
        $this->priority = $value;
        return $this;
    }
}
```

A.6 Текст файла MySqlConnection.php

```

<?php
namespace Framework\db\Connection;
use
Framework\db\Exception\ConnectionException;
use Framework\db\Migration\MySQLMigration;
use
Framework\db\QueryBuilder\MySQLQueryBuilder;
use PDO;
class MySqlConnection extends Connection
{
    private PDO $pdo;
    private string $database;
    public function __construct($config)
    {
        [
            'host' => $host,
            'port' => $port,
            'database' => $database,
            'username' => $username,
            'password' => $password,
        ] = $config;
        if (empty($host) || empty($database) ||
empty($username)) {
            throw new
\InvalidArgumentException('Connection
incorrectly configured');
        }
        $this->database = $database;
        $dsn =
sprintf("mysql:host=%s;port=%s;dbname=%s;charset=utf8", $host, $port, $database);
        $this->createPdo($dsn, $username,
$password);
    }
    public function createPdo(string $dsn,
$username, $password)
    {
        try {
            $this->pdo = new PDO($dsn, $username,
$password);
            if (isDev()) {
                $this->pdo-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
                $this->pdo-
>setAttribute(PDO::ATTR_EMULATE_PREPARE
S, false);
            }
        } catch (\PDOException $e) {
            if (isDev()) {
                print "Error!: " . $e->getMessage() .
"<br/>";
                die();
            } else {
                throw new
ConnectionException('Connection incorrectly
configured');
            }
        }
    }

    public function pdo(): PDO
    {
        return $this->pdo;
    }
    public function query(): MySQLQueryBuilder
    {
        return new MySQLQueryBuilder($this);
    }
    public function createTable(string $table):
MySQLMigration
    {
        return new MySQLMigration($this, $table,
'create');
    }
    public function alterTable(string $table):
MySQLMigration
    {
        return new MySQLMigration($this, $table,
'alter');
    }
    public function getTables(): array
    {
        $statement = $this->pdo()->prepare('SHOW
TABLES');
        $statement->execute();
        $results = $statement-
>fetchAll(PDO::FETCH_NUM);
        $results = array_map(fn($result) =>
$result[0], $results);
        return $results;
    }
    public function hasTable(string $name): bool
    {
        $tables = $this->getTables();
        return in_array($name, $tables);
    }
    public function dropTables(): int
    {
        $this->pdo-
>setAttribute(PDO::ATTR_EMULATE_PREPARE
S, false);
        $statement = $this->pdo->prepare("
SELECT CONCAT('DROP TABLE IF
EXISTS `', table_name, `')
FROM information_schema.tables
WHERE table_schema = '{$this-
>database}';
");
        $statement->execute();
        $dropTableClauses = $statement-
>fetchAll(PDO::FETCH_NUM);
        $dropTableClauses = array_map(fn($result)
=> $result[0], $dropTableClauses);
        $clauses = [
            ...$dropTableClauses,
        ];
        $totalTableCount = count($clauses);
        $droppedTableCount = 0;
        foreach ($clauses as $key => $tableToDrop)
        {

```

```

$statement = $this->pdo->prepare($sql);
if ($statement->execute()) {
    preg_match('/^(.*)$/', $tableToDrop, $m);
    echo sprintf("Table with name: %s", $m[0]) . PHP_EOL;
}
}

return $totalTableCount;
}

$droppedTableCount++;
}

$droppedTableCount;
}

===

```

A.7 Текст файла BaseActiveRecord.php

```

<?php
namespace Framework\db;
use Framework\db\Connection\Connection;
use Framework\db\Connection\MysqlConnection;
use Framework\db\Connection\SqliteConnection;
use Framework\helpers\Config;
use Modules\DD;
abstract class BaseActiveRecord
{
    protected Connection $connection;
    protected string $table;
    protected array $attributes = [];
    protected array $dirty = [];
    public function __get(string $property)
    {
        $getter = 'get' . ucfirst($property) . 'Attribute';
        $value = null;
        if (method_exists($this, $property)) {
            $relationship = $this->$property();
            $method = $relationship->method;
            $value = $relationship->$method();
        }
        if (method_exists($this, $getter)) {
            $value = $this->$getter($this->attributes[$property] ?? null);
        }
        if (isset($this->attributes[$property])) {
            $value = $this->attributes[$property];
        }
        if (isset($this->casts[$property]) &&
            is_callable($this->casts[$property])) {
            $value = $this->casts[$property]($value);
        }
        return $value;
    }
    public function __set(string $property, $value)
    {
        $setter = 'set' . ucfirst($property) . 'Attribute';
        if (!in_array($property, $this->dirty)) {
            array_push($this->dirty, $property);
        }
        if (method_exists($this, $setter)) {
            $this->attributes[$property] = $this->$setter($value);
            return;
        }
        $this->attributes[$property] = $value;
    }
    public function setConnection(Connection $connection): self
    {
        $this->connection = $connection;
        return $this;
    }
    public function getConnection(): Connection
    {
        if (!isset($this->connection)) {
            $this->connection = app('database');
        }
        return $this->connection;
    }
}

//
//
public function setTable(string $table): self
{
    $this->table = $table;
    return $this;
}
public function getTable(): string
{
    if (!isset($this->table)) {
        throw new \Exception('Table is not set
and getTable is not defined');
    }
    return $this->table;
}
public function getAttributes()
{
    return $this->attributes;
}
}
* creates a model and then
* populates the array of attributes for specific
model
*/
public static function with(array $attributes = []):
self
{
    $model = new static();
    $model->attributes = $attributes;
    return $model;
}
* @throws \Exception
*/
public static function query()
{
    $model = new static();
    $queryBuilder = $model->getConnection()->query();
    return (new ModelCollector($queryBuilder,
static::class))
->from($model->getTable());
}
* @throws \Exception
*/
public static function __callStatic(string
$method, array $parameters = [])
{
    return static::query()->$method(...$parameters);
}
* @alias to update
* @description this method designed either to
update an existing data or insert a new one
*/
public function save(): self
{
    $values = [];
    foreach ($this->dirty as $dirtyProp) {
        $values[$dirtyProp] = $this->attributes[$dirtyProp];
    }
}

```

```

    }
    $keys = $this->getFillable();
    $dbValues = array_map(fn ($el) => $this->attributes[$el], $keys);
    $data = [$keys, $dbValues];
    $query = static::query();
    if (isset($this->attributes['id'])) {
        $query
            ->where('id', $this->attributes['id'])
            ->update(...$data);
        return $this;
    }
    $query->insert(...$data);
    $this->attributes['id'] = $query->getLastInsertId();
    $this->dirty = [];
    return $this;
}
* updates the record
* refers to save that implements the logic
*/
public function update(): self
{
    $this->save();
    return $this;
}
public function delete($columnName = 'id'): self
{
    if (isset($this->attributes[$columnName])) {
        static::query()
            ->where($columnName, $this->attributes[$columnName])
            ->delete();
    }
    return $this;
}
public function getDirty()
{
    return $this->dirty;
}
public function toInt($value): int

```

```

{
    return (int)$value;
}
public function hasOne(string $class, string $foreignKey, string $primaryKey = 'id'): Relationship
{
    $model = new $class();
    $query = $class::query()->from($model->getTable())->where($foreignKey, $this->attributes[$primaryKey]);
    return new Relationship($query, 'first');
}
public function hasMany(string $class, string $foreignKey, string $primaryKey = 'id')
{
    $model = new $class();
    $query = $class::query()->from($model->getTable())->where($foreignKey, $this->attributes[$primaryKey]);
    return new Relationship($query, 'all');
}
public function belongsTo(string $class, string $foreignKey, string $primaryKey = 'id')
{
    $model = new $class();
    $query = $class::query()->from($model->getTable())->where($primaryKey, $this->attributes[$foreignKey]);
    return new Relationship($query, 'first');
}
public static function find(int $id): self
{
    return static::where('id', $id)->first();
}
public function getDb()
{
    return app()->getDb();
}
}

```

A.8 Текст файла Model.php

```

<?php
namespace Framework;
use Framework\db\BaseActiveRecord;
use Framework\db\Connection\Connection;
use Framework\db\Query;
use Framework\helpers\ArrayHelper;
use Framework\helpers\StringHelper;
use Framework\helpers\UtilsHelper;
use Framework\notification\Message;
use Modules\DD;
abstract class Model extends BaseActiveRecord
{
    public const SCENARIO_NEW = 'new';
    public const SCENARIO_EDIT = 'edit';
    public const SCENARIO_DELETE = 'delete';
    public array $errors = [];
    public const SINGLE_ERROR_MESSAGE =
false;
    public ?string $scenario = null;
    protected ?int $id = -1;
    protected array $fillable = [];
    protected Connection $connection;
    private ?array $skippedFields = [];
    private ?Model $oldObject;
    protected $checkIfHasChanges = true;
    protected $validationGeneralDescription = 'The
following fields are incorrect';
    protected $validationTitle = 'Form Validation';
    abstract public function rules(): array;
    abstract public static function tableName():
string;
    public function __construct()
    {
        $this->booted();
    }
    public function load($data = [])
    {
        $this->oldObject = clone $this;
        foreach ($data as $key => $value) {
            $this->{$key} = $value;
        }
    }
    public function loadToAttributes($data = [])
    {
        $this->oldObject = clone $this;
        foreach ($data as $key => $value) {
            $this->{$key} = $value;
        }
    }
    protected function booted()
    {
    }
    public function getFillable(): array
    {
        return $this->fillable;
    }
    abstract protected function scenarios(): array;
    protected function getActiveScenario(): array
    {
        if (is_null($this->scenario)) {
            if (empty($this->getFillable())) {
                throw new
\InvalidArgumentException("Please set a
scenario");
            }
            return $this->getFillable();
        }
        if ($this->scenario && empty($this-
>scenarios())) {
            throw new
\InvalidArgumentException("Please make sure
you have the right scenarios selected");
        }
        if (isset($this->scenarios()[$this->scenario]))
        {
            return $this->scenarios()[$this->scenario];
        }
        throw new
\InvalidArgumentException("Scenario not found,
please set a correct one or add a new");
    }
    public function validate($attributes = null,
$clearErrors = true): bool
    {
        if ($clearErrors) {
            $this->clearErrors();
        }
        $scenarioAttributes = $attributes ?? $this-
>getActiveScenario();
        $scenarioAttributes = $scenarioAttributes ??
$this->getFillable();
        $scenarioAttributes
        =
is_string($scenarioAttributes)
        ?
[$scenarioAttributes] : $scenarioAttributes;
        $rules = $this->rules();
        foreach ($rules as $item => $rule) {
            list($ruleAttributes, $validator, $params) =
$this->getValidatorAndAttributes($item,
$scenarioAttributes, $rule);
            validate($this, $ruleAttributes, $validator,
$params);
        }
        if ($this->hasErrors()) {
            $this->notifyValidationErrors();
        }
        if ($this->checkIfHasChanges) {
            return !$this->hasErrors() && $this-
>isDirty();
        }
        return !$this->hasErrors();
    }
    protected function private($params)
    {
        DD::dl($params);
        echo 'private';
        exit;
    }
    protected function addErrorForRule(string $attr,
string $ruleName, $params = [])
    {
    }

```

```

        $message = $this->errorMessages()[$ruleName] ?? "";
        foreach ($params as $key => $value) {
            $message = str_replace("{{$key}}",
            $value, $message);
        }
        $this->errors[$attr][] = $message;
    }
    public function addError(string $attr, string
    $message, string $validator = "")
    {
        if ($message) {
            $this->errors[$attr] = [];
            $this->errors[$attr][$validator] =
            $message;
        }
    }
    public function errorMessages(): array
    {
        return [
            Rules::REQUIRED => 'This field is
            required',
            Rules::EMAIL => 'This field must be a
            valid email address',
            Rules::MIN_STR => 'This field must
            contain at least {min} chars',
            Rules::MAX_STR => 'This field must
            contain at most {max} chars',
            Rules::MATCH => 'This field must be
            same as {match}',
            Rules::UNIQUE => 'Record with this {field}
            already exists',
            Rules::MSG_UNIQUE_TOO_SHORT =>
            'The {field} must have at least {n} chars',
            Rules::ARRAY_FROM_STR => 'The
            {field} must have provide a property to fill',
            Rules::IN_ARRAY => 'The {field} must be
            in {value_list}',
        ];
    }
    public function hasError($fieldName): bool
    {
        if ($this->hasErrors()) {
            return isset($this->errors[$fieldName]);
        }
        return false;
    }
    public function hasErrors(): bool
    {
        return !UtilsHelper::isEmpty($this->errors);
    }
    public function getFirstError($fieldName): string
    {
        if ($this->hasError($fieldName)) {
            $ruleName = array_key_first($this->
            errors[$fieldName]);
            return $this->errors[$fieldName][$ruleName];
        }
        return "";
    }
    public function labels(): array

```

```

    {
        return [];
    }
    public function getLabel($attribute): string
    {
        return $this->labels()[$attribute] ??
        StringHelper::uppercaseWordsAndReplaceSpecifi
        er('_', $attribute);
    }
    public static function create(array
    $mixedKeyValArr, array $valuesArr = [])
    {
        if (ArrayHelper::isAssoc($mixedKeyValArr)) {
            Query::getInst()-
            >insert(static::tableName(),
            array_keys($mixedKeyValArr),
            array_values($mixedKeyValArr));
        } elseif (!empty($valuesArr)) {
            Query::getInst()-
            >insert(static::tableName(), $mixedKeyValArr,
            $valuesArr);
        }
    }
    public static function deleteWhereIn($field,
    array $arr): int
    {
        return Query::getInst()-
        >deleteWhereIn(static::tableName(), $field, $arr);
    }
    protected function
    checkIncomingAttributes($attributes): void
    {
        if (
            !isset($attributes) &&
            is_string($attributes) &&
            !isset($attributes) &&
            is_array($attributes)
        ) {
            throw new
            \InvalidArgumentException("You must use either
            an array of attributes or a single attribute name");
        }
    }
    public function addDirty($attrName)
    {
        $this->dirty[] = $attrName;
    }
    protected function isDirty(): bool
    {
        if (isset($this->oldObject)) {
            foreach ($this->oldObject as $attrName =>
            $attrValue) {
                if (in_array($attrName, $this->
                getFillable()) && $this->oldObject->{$attrName}
                != $this->{$attrName}) {
                    $this->addDirty($attrName);
                }
            }
        }
        if (empty($this->dirty)) {

```

```

        Application::app()->session-
>setFlash(Message::WARNING,          $this-
>validationTitle, 'No fields were changed');
    }
    return count($this->dirty) > 0;
}
protected function clearErrors($attribute = null)
{
    if (is_null($attribute)) {
        $this->errors = [];
    } else {
        unset($this->errors[$attribute]);
    }
}
protected function notifyValidationErrors(): void
{
    $attribNames = array_keys($this->errors);
    $fields      = array_map(fn($attr) =>
sprintf("<li>%s</li>", $this->getLabel($attr)),
$attribNames);
    $desc = sprintf("%s:<br /><ul>", $this-
>validationGeneralDescription) . implode("\n",
$fields);
    $desc .= "</ul>";
    Application::$app->session->setFlash(
        Message::DANGER,
        $this->validationTitle,
        $desc,
        Message::ADMIN_VISIBLE
    );
}
protected function
getValidatorAndAttributes($item,
$scenarioAttributes, $rule): array
{
    $ruleValidator = "";
    $ruleAttributes = $params = [];
    $singleAttribute = is_string($item) &&
in_array($item, $scenarioAttributes, true);
    if ($singleAttribute) {
        $ruleAttributes[] = $item;
        if ($singleRule = is_string($rule)) {
            $ruleValidator = $rule;

```

```

        } else {
            $ruleValidator = isset($rule[0]) ?
array_shift($rule) : null;
            $params = $rule;
        }
    } elseif (is_array($rule) && isset($rule[0],
$rule[1])) {
        $ruleAttribute = array_shift($rule);
        if (is_array($rule[0])) {
            $ruleValidator = array_shift($rule[0]);
            $params = $rule[0];
        } else {
            $ruleValidator = array_shift($rule);
        }
        if (is_array($ruleAttribute) &&
!empty(array_intersect($ruleAttribute,
$scenarioAttributes))) {
            $ruleAttributes = $ruleAttribute;
        } elseif (in_array($ruleAttribute,
$scenarioAttributes, true)) {
            $ruleAttributes[] = $ruleAttribute;
        }
        $ruleValidator = is_array($ruleValidator) ?
$ruleValidator[0] : $ruleValidator;
        $params = empty($params) &&
ArrayHelper::isAssoc($rule) ? $rule : $params;
    }
    if (empty($ruleValidator)) {
        throw new
\InvalidArgumentException('$ruleValidator must
be set
[NOTE: check if you have field in scenario
or in fillable]');
    }
    if (empty($ruleAttributes)) {
        throw new
\InvalidArgumentException('$ruleAttributes must
be set');
    }
    return array($ruleAttributes, $ruleValidator,
$params);
}
}

```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
Кафедра програмних засобів

Дипломна кваліфікаційна робота

**Дослідження та програмна реалізація фреймворку
для підвищення ефективності розробки
вебзастосунків**

Виконав:
ст. гр. КНТ-120м

Р.А. Захряпа

Керівник:
к.т.н., доцент

І.Я. Зеленцова

2021

Рисунок Б.1 – Слайд 1

МЕТА РОБОТИ

Об'єкт дослідження – засоби підвищення ефективності розробки вебзастосунків.

Предмет дослідження – фреймворки з використанням останніх технологій та мови програмування PHP.

Мета роботи – пришвидшення процесу розробки вебзастосунків шляхом використання шаблонізатора.

2

Рисунок Б.2 – Слайд 2

ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ РОЗРОБКИ ТА ДОСЛІДЖЕНЬ

Актуальність розробки полягає в тому, що:

- ◆ широко використовується ООП та мова PHP;
- ◆ забезпечується можливість мінімізувати вихідний код на користь швидкості його виконання;
- ◆ значно пришвидшується створення виглядів завдяки шаблонізатору;
- ◆ зручно реалізувати компоненти та їх методи з обробки вхідних даних.

3

Рисунок Б.3 – Слайд 3

ЗАДАЧІ, ЯКІ ПОТРІБНО ВИРІШИТИ

У даному фреймворку повинні бути враховані вимоги:

- ◆ усунення недоліків інших аналогів;
- ◆ спрощення написання коду;
- ◆ зменшення витрат часу на розробку вебзастосунка;
- ◆ фреймворк повинен надавати лише необхідні функції для пришвидшення обробки запитів та надання відповіді для клієнтської частини.

4

Рисунок Б.4 – Слайд 4

ОСНОВНІ КОМПОНЕНТИ СИСТЕМИ:

- ◆ маршрутизатор;
- ◆ валідатор;
- ◆ шаблонізатор;
- ◆ міграції БД;
- ◆ клас Form;
- ◆ файл конфігурацій;
- ◆ клас для роботи з сесіями;
- ◆ клас для створення повідомлень;
- ◆ класи для взаємодії з БД;
- ◆ обробка помилок та їх відображення.

5

Рисунок Б.5 – Слайд 5

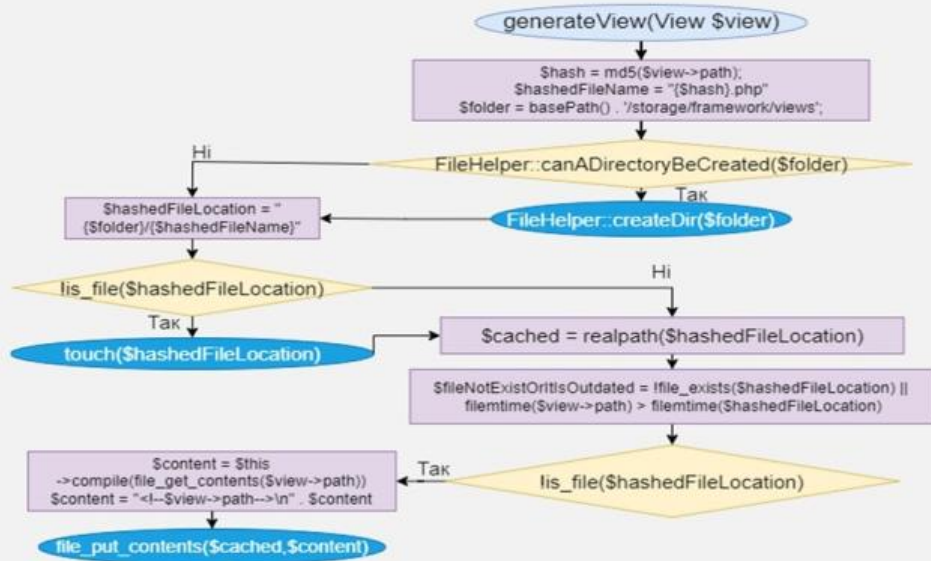
ПОРІВНЯННЯ PHP ФРЕЙМВОРКІВ

Параметри порівняння	Yii 2.31	Laravel 8.41.0	Symfony 5.2.1	Codeigniter 4.0.4	Розроблена програма
Підтримуючі версії PHP	>=5.1–8.0	>=7.3-8.0	>=7.2.5–8.0	>=7.3	>= 7.4
Документація і приклади	Детально описана, містить багато прикладів; є можливості завантаження документації	Надає велику кількість прикладів з детальним описом	Велика кількість прикладів, але вони не дуже детально описані	Недостатньо детальна, мало прикладів; Іноді сайт з документацією не працює	Надає приклади по використанню компонентів та класів системи
Жорстка структура каталогів	+	+	+	—	+
Складність встановлення	Складно	Легко	Помірна	Легко	Легко
Складність налаштування	Довго, через потребу в налаштуванні файлів .htaccess	Необхідно задати декілька параметрів в одному файлі	Швидко та інтуїтивно	Іноді документація не надає опис усіх параметрів	У разі використання файлів .htaccess, потребує лише налаштування двох файлів
Вбудований шаблонізатор	—	blade	twig	—	AdvancedEngine
Бібліотека для клієнтського інтерфейсу	Модифікована версія Bootstrap 3	Модифікована версія Bootstrap 5	Надає можливості встановлення сторонніх стилів	Надає можливості встановлення сторонніх стилів	Bootstrap 5

6

Рисунок Б.6 – Слайд 6

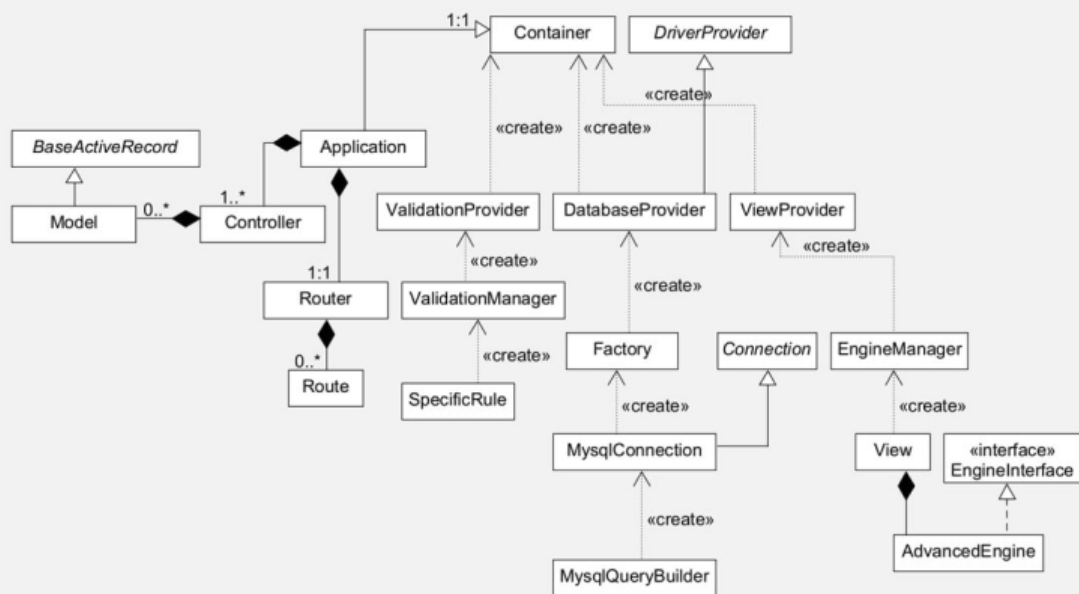
АЛГОРИТМ ШАБЛОНІЗАТОРА КЛАСУ ADVANCEDENGINE ПО ГЕНЕРАЦІЇ PHP ФАЙЛІВ



7

Рисунок Б.7 – Слайд 7

UML-ДІАГРАМА ФРЕЙМВОРКУ



8

Рисунок Б.8 – Слайд 8

ТАБЛИЦІ БД, ЯКІ МОЖЛИВО СТВОРИТИ ЗА ДОПОМОГОЮ «МІГРАЦІЙ» БД

Для виконання міграцій необхідно виконати наступні дії:

- ◆ відкрийте командний рядок у кореневому каталозі;
- ◆ введіть команду «php command.php migrate».

Міграція під назвою «migrate» створює наступні таблиці БД:

v	⚙	rz_framework users	v	⚙	rz_framework migrations
↑		id : int(11) unsigned	↑		id : int(11) unsigned
●		name : varchar(255)	●		name : varchar(255)
●		email : varchar(255)	■		created_at : datetime
●		password : varchar(255)			

11

Рисунок Б.11 – Слайд 11

ПРИКЛАД ВАЛІДАЦІЇ ДАНИХ

Якщо при натисканні кнопки «Вхід», форма не проходить валідацію даних, то у вигляді будуть відображені помилки

Дані введені некоректно ✕

Виправте помилки та повторіть спробу:

- Електронна Адреса
- Пароль

Вхід до системи

Електронна Адреса

Електронна Адреса ❗

Важить як мінімум 3 символи

Пароль ❗

Пароль ❗

Важить як мінімум 8 символів

Вхід

Вхід до системи

Електронна Адреса

email@email.com ❗

Електронна Адреса або пароль введені некоректно

Пароль

***** ✔ ❗

Вхід

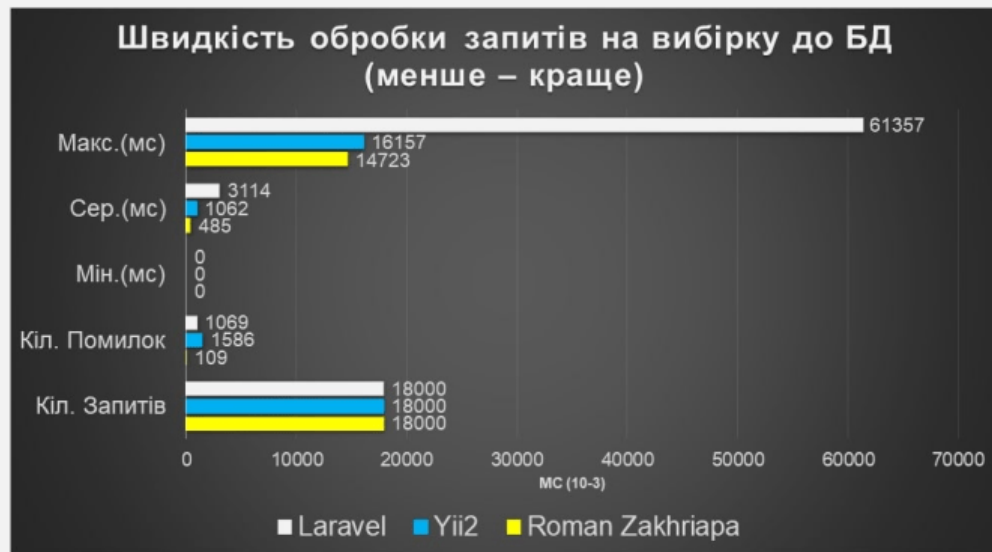
Поле «Електронна Адреса» та «Пароль» не відповідають умові, дожвині, яка задана у Моделі

Поле «Електронна Адреса» або «Пароль» не відповідає жодному запису з БД

12

Рисунок Б.12 – Слайд 12

ПОРІВНЯННЯ ШВИДКОСТІ РОБОТИ РОЗРОБЛЕНОГО ФРЕЙМВОРКА З
RНР-ФРЕЙМВОРКАМИ ЗА ДОПОМОГОЮ APACHE JMETER, ТЕСТ №1



13

Рисунок Б.13 – Слайд 13

ПОРІВНЯННЯ ШВИДКОСТІ РОБОТИ РОЗРОБЛЕНОГО ФРЕЙМВОРКА З
RНР-ФРЕЙМВОРКАМИ ЗА ДОПОМОГОЮ APACHE JMETER, ТЕСТ №2



14

Рисунок Б.14 – Слайд 14

ВІДНОСНЕ ПОРІВНЯННЯ ФРЕЙМВОРКІВ З УРАХУВАННЯМ ЧАСУ НА СТВОРЕННЯ ГЛОВНОГО ОБ'ЄКТА ФРЕЙМВОРКУ З ВИКОРИСТАННЯ ІНСТРУМЕНТУ JMETER



Забезпечує скорочення кількості помилок майже в 14 разів. Швидкість виконання коду краще на 22% ніж у Yii2, та на 84% ніж у Laravel.

15

Рисунок Б.15 – Слайд 15

ПОРІВНЯННЯ ШВИДКОСТІ РОБОТИ РОЗРОБЛЕНОГО ФРЕЙМВОРКА З РНР-ФРЕЙМВОРКАМИ ЗА ДОПОМОГОЮ КЛАСУ DD



16

Рисунок Б.16 – Слайд 16



Рисунок Б.17 – Слайд 17

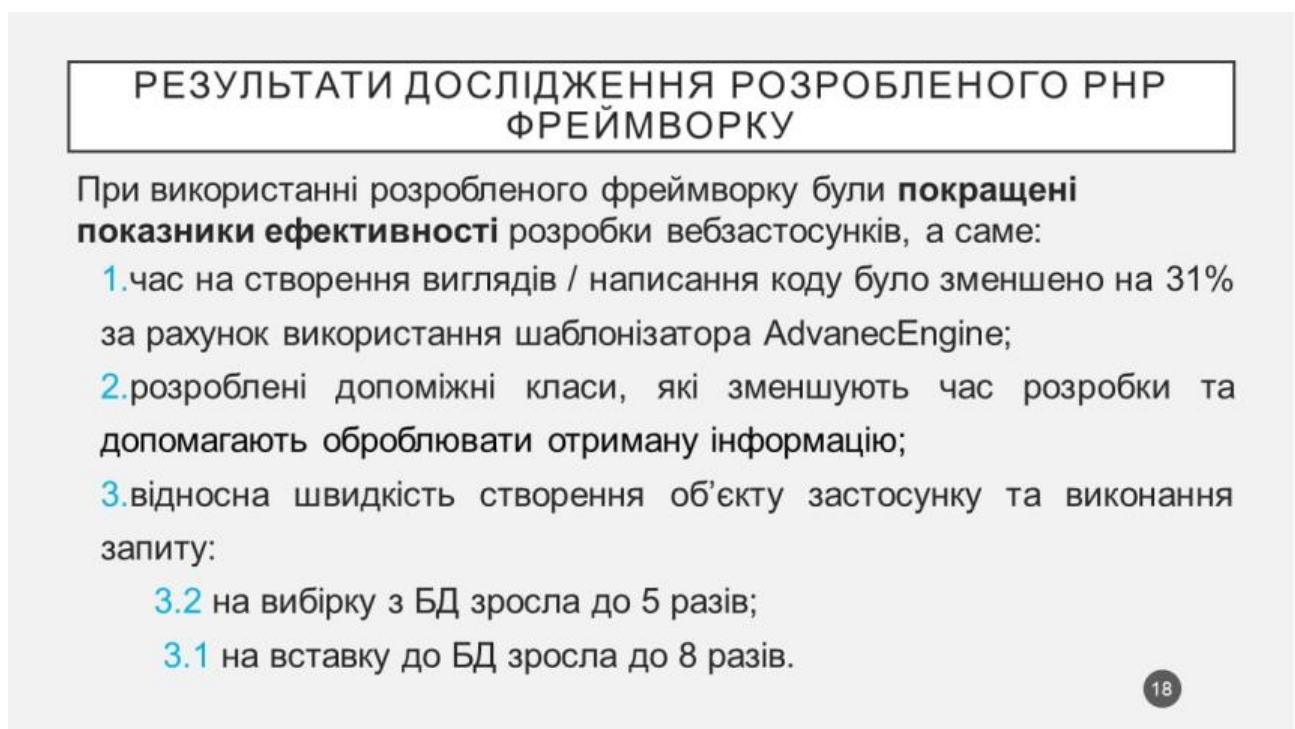


Рисунок Б.18 – Слайд 18

ВИСНОВКИ

1. Розроблено PHP-фреймворк, який дозволяє прискорити розробку вебзастосунків покращити розуміння коду завдяки інтуїтивним назвам методів, класів та вбудованого шаблонізатора, що значно пришвидшує створення виглядів та їх інтерпретацію;
2. Використані методи, які реалізовані у мові програмування, що значно впливають на швидкодію системи в цілому;
3. Після виходу фреймворку у відкритий доступ, його функціонал зможуть розширювати будь-які користувачі, для чого їм буде потрібно створити запит у системі контролю версій та описати, що було додано, видалено або змінено.

19

Рисунок Б.19 – Слайд 19

ПЕРСПЕКТИВИ РОЗРОБКИ

Фреймворк буде розповсюджуватися як відкрите програмне забезпечення після розроблення ще декілька модулів / функцій системи, таких як:

- ◆ інтернаціоналізація;
- ◆ кешування сторінок;
- ◆ підтримка інших типів БД (MariaDB, PostgreSQL);
- ◆ створення контролерів та моделей за допомогою командного рядка.

20

Рисунок Б.20 – Слайд 20