МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-2-22 Віктор КУЛАГІН
Науковий керівник: к.т.н., доц. Тетяна ДЕМКІВСЬКА
Рецензент: д.т.н., проф. Віктор ЧУПРИНКА

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 комп'ютерні науки

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____Володимир ЩЕРБАНЬ

«___»_____2023 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Кулагіну Віктору Петровичу

1.**Тема кваліфікаційної роботи**: Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства.

Науковий керівник роботи: Демківська Тетяна Іванівна, к.т.н., доц.

затверджені наказом закладу вищої освіти від 12.09.2023 року, № 210-уч.

2.**Вихідні дані до кваліфікаційної роботи**: Розробка кафедри комп'ютерних наук, рекомендована література, додатки.

3.**Зміст кваліфікаційної роботи**: Вступ; Розділ 1. Основні принципи роботи фреймворка Symfony; Розділ 2. Проєктування автоматизованої системи обліку робочого часу робітників підприємства; Розділ 3. Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства; Висновки; Список використаних джерел; Додаток А. Програмний код; Додаток Б. Публікація; Додаток В. Презентація.

4.**Дата видачі завдання** 08.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	30.08.2023	
2	Розділ 1. Основні принципи роботи фреймворка Symfony	06.09.2023	
3	Розділ 2. Проєктування автоматизованої системи обліку робочого часу робітників підприємства	28.09.2023	
4	Розділ 3. Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства	21.10.2023	
5	Висновки	29.10.2023	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	06.11.2023	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку (за 14 днів дозахисту)	07.11.2023	
8	Подача кваліфікаційної роботи для рецензування	07.11.2023	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату	08.11.2023	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	10.11.2023	

З завданням ознайомлений:

Студент _____ Віктор КУЛАГІН

Науковий керівник _____ Тетяна ДЕМКІВСЬКА

АНОТАЦІЯ

Кулагін В. П. Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства.

Кваліфікаційна робота за спеціальністю 122 - «Комп'ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2023 рік.

В кваліфікаційній роботі було розроблено автоматизовану систему обліку робочого часу робітників підприємства, яка значно спрощує облік робочого часу працівників, покращує взаємодію працівника підприємства з адміністрацію, надаю можливість працівнику бачити виконану роботу та кількість часу, витрачену на кожну задачу, а адміністрації – формувати звіти робочого часу працівників.

Автоматизована система обліку робочого часу робітників підприємства розроблена на основі фреймворка Symfony з використання сучасних методів та інструментів розробки, що гарантую довготривалу підтримку, зведення до мінімуму проблем з безпекою програмного забезпечення, легкість додавання нового функціоналу, а також інтеграцію з іншими програмними комплексами та технологіями.

Ключові слова: Framework, Symfony, автоматизована система, облік часу, звіти.

ANNOTATION

Kulahin V.P. Development of software for the automated system of recording the working time of the workers in enterprise.

Qualification work by specialty 122 - "Computer Science and Technologies" - Kyiv National University of Technology and Design, Kyiv, 2023.

In the qualification thesis, an automated system for recording the working time of the company's employees was developed, which greatly simplifies the recording of the working time of the employees, improves the interaction of the employee of the company with the administration, provides the opportunity for the employee to see the work performed and the amount of time spent on each task, and gives an opportunity for the administration to form employees' working time reports.

The automated system for recording the working hours of the company's employees is developed on the basis of the Symfony Framework using modern development methods and tools, which guarantees long-term support, minimizing software security problems, ease of adding new functionality, and integration with other software complexes and technologies.

Keywords: Framework, Symfony, automated system, working time recording, reports

ЗМІСТ

ВСТУП

Актуальність роботи. На сьогоднішній день паперовий варіант обліку робочого часу працівників підприємства вже дуже застарілий і не актуальний. Він потребує багато часу для його ведення обліку працівників підприємства та актуалізації облікових даних. Паперовий варіант обліку ускладнює формування різноманітної звітності, пошук необхідної інформації тощо. Крім того виникають складності з забезпеченням безпеки зберігання даних та конфіденційністю інформації.

В результаті проведеного дослідження розроблено програмний продукт, який допоможе вести облік працівників, вести облік робочого часу кожного працівника, облік часу витраченого на кожну конкретну задачу, формувати звітність про кількість витраченого часу на кожного клієнта та формувати звіти про витрачений час. Даний програмний продукт буде вести облік відгулів, лікарняних, відпусток та формувати різноманітні звіти.

Автоматизація різноманітних процесів на підприємстві дозволяє полегшити роботу працівників та надає можливості для швидкого формування різноманітних звітів. Це дає змогу контролювати ефективність кожного працівника та полегшує багато внутрішніх процесів підприємства.

Використання автоматизованої системи дозволяє уникнути багатьох проблем, притаманних паперовим документам. Зберігання інформації в електронному вигляді надає безліч переваг, як то відсутність необхідності фізичного місця для зберігання даних, небезпеки знищення даних тощо.

Мета дослідження. Метою даного дослідження є розробка автоматизованої системи обліку робочого часу робітників підприємства на базі Symfony, проведення аналізу існуючих аналогічних систем та вивчення можливостей Symfony для розробки власної системи, яка буде відповідати специфічним вимогам підприємства.

Об’єктом дослідження цієї роботи є фреймворк Symfony та його можливості для розробки автоматизованої системи обліку робочого часу робітників підприємства.

Предметом дослідження є розробка автоматизованої системи обліку робочого часу робітників підприємства.

Елементи наукової новизни. Створення автоматизованої системи обліку робочого часу робітників підприємства під конкретні та специфічні вимоги.

Практична значущість роботи. Створена автоматизована система обліку робочого часу робітників підприємства значно підвищить продуктивність на підприємстві, спростить взаємодію між працівниками і адміністрацією, забезпечить безпеку даних та конфіденційність інформації, надасть можливість формувати різноманітні звіти тощо.

Апробація результатів роботи. Автоматизована система обліку робочого часу робітників підприємства має простий та зрозумілий інтерфейс та надає широкий спектр можливостей для використання. В подальшому, автоматизована система може легко бути модифікована та модернізована під нові вимоги підприємства.

РОЗДІЛ 1
ОСНОВНІ ПРИНЦИПИ РОБОТИ ФРЕЙМВОРКА SYMFONY

Фреймворк (англ. Framework, каркас, платформа, структура, інфраструктура) - інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проєкту та написання коду.

Фреймворк - це програмне середовище, яке спрощує та прискорює створення програмного забезпечення. За використання фреймворків ви пишете лише код, який реалізує логіку, специфічну для вашого продукту. Нам не доводиться самостійно забезпечувати роботу з базою даних, автентифікацію, підтримку сеансів тощо. Все це реалізовано у фреймворках.

1.1. Огляд популярних фреймворків

Symfony - це провідний фреймворк для PHP, який використовується для створення веб-сайтів і веб-додатків. Першу версію Symfony розроблено у 2007 році, і він базується на наборі компонентів, які також носять назву Symfony Components.

Symfony Components - це набір відокремлених і можливих до багаторазового використання компонентів, які утворюють фундамент для таких популярних PHP-програм, як Prestashop, Drupal і Laravel.

Особливостями Symfony є висока стабільність та надійність, що роблять його популярним серед досвідчених розробників. Фреймворк Symfony дозволяє з ранніх етапів розробки створити основу для масштабування і розширення функціональності. Він також славиться високою гнучкістю налаштувань та можливістю інтеграції з різними популярними CMS та сервісами, а програмні рішення, побудовані на основі Symfony, зазвичай відзначаються високою оптимізацією.

Symfony також спрощує інтеграцію з фреймворками JavaScript, що є важливим аспектом розробки.

Зараз над розробкою Symfony Framework працює понад 600 000 розробників із понад 120 країн, що свідчить про його велику спільноту.

Філософія Symfony покладає акцент на підтримку професіоналізму, найкращих практик, стандартизацію та сумісність програм, що робить його одним із найпопулярніших фреймворків у світі на сьогоднішній день.

Рис. 1.1 Логотип фреймворка Symfony

Laravel. Цей фреймворк для PHP є одним із найпопулярніших у всьому світі і широко використовується для створення складних та індивідуальних веб-сайтів і програм. Він також відзначається своєю відомою давністю, бо був запущений ще у 2011 році. Протягом цього часу фреймворк постійно вдосконалювався і оновлювався, що зробило його одним із найзручніших інструментів для розробки.

Серед недоліків цього фреймворку можна відзначити необхідність великої кількості ручної роботи для фахівців, що, можливо, сповільнює процес розробки, але водночас надає гнучкість при створенні програмних рішень.

Рис. 1.2 Логотип фреймворка Laravel

CodeIgniter. Цей фреймворк, розроблений на мові програмування PHP, був створений у 2013 році та має відкритий вихідний код. Він ідеально підходить для розробки програм, де важливо спростити процес написання коду. Однією з відмінних рис CodeIgniter є його висока продуктивність.

Це робить цю платформу однією з кращих виборів для створення веб-додатків, оскільки чистий вихідний код гарантує високу швидкодію та мінімізує кількість помилок. Крім того, програмні рішення, побудовані на основі цього фреймворку, легко піддаються комплексному тестуванню.

З основних недоліків можна виділити слабка система кешування та поганий стиль написання коду.

Рис. 1.3 Логотип фреймворка CodeIgniter

Yii – це високоефективний PHP-фреймворк, розроблений для швидкої створення сучасних веб-додатків. Цей універсальний фреймворк призначений для використання в будь-якому типі веб-додатків, які використовують PHP.

Завдяки його компонентній структурі та відмінній підтримці кешування, фреймворк особливо підходить для розробки великих проєктів, таких як портали, форуми, системи управління контентом (CMS), інтернет-магазини або RESTful-додатки.

Цей фреймворк надає зручний набір інструментів для створення ефективних інтерфейсів, тому він високо оцінюється фахівцями, які займаються розробкою інтерфейсів користувача (UI). Yii є одним з найшвидших фреймворків для PHP.

Крім того, Yii має інтеграцію з jQuery, що спрощує спільну роботу команди фахівців над проєктом та взаємодію зі сторонніми розробниками.

Рис. 1.4 Логотип фреймворка Yii

Zend – об'єктно-орієнтований фреймворк, який виділяється своєю багатофункціональною архітектурою та простотою у використанні для роботи з різними видами програмних рішень та розширення їхнього функціоналу.

Цей фреймворк часто використовується великими компаніями, оскільки забезпечує високу надійність та захищеність проєкту.

Рис. 1.5 Логотип фреймворка Zend

Phalcon - це фреймворк для PHP, створений з використанням мови Сі, має глибоку низькорівневу архітектуру, яка гарантує максимальну продуктивність під час використання. Він дуже зручний для розробки веб-додатків.

Phalcon спрощує роботу з шаблоном проєктування MVC, надаючи функції та класи, які можна використовувати для різних видів програмних рішень.

Рис. 1.6 Логотип фреймворка Phalcon

CakePHP – фреймворк, який багато хто вважає зразком надійності, відзначається гнучкою архітектурою, здатною до масштабування, та наявністю сучасних допоміжних компонентів, таких як класи FlashComponent та FlashHelper. Ці компоненти дуже зручні для роботи над інтернет-ресурсами та веб-додатками. CakePHP також славиться відмінною продуктивністю.

Ще однією важливою особливістю даного фреймворка є перетворення тем на плагіни, що дозволяє створювати одні з найкращих тем серед усіх перерахованих фреймворків. Завдяки можливості використовувати плагіни як теми, розширюється їх функціональність, і робота з програмним кодом стає набагато простішою.

Рис. 1.7 Логотип фреймворка CakePHP

1.2. Переваги та недоліки використання фреймворків

Використання фреймворка не є обов'язковим, але він є одним з можливих інструментів для розробки програмних продуктів.

Використання фреймворків надає впевненість у тому, що розроблений програмний продукт повністю відповідає бізнес-правилам, має чітку структуру та його можна легко підтримувати та оновлювати.

Фреймворки дозволяють розробникам заощаджувати час, використовуючи загальні модулі та фокусуючись на інших аспектах розробки.

Парадигма Symfony Framework та інших фреймворків полягає в "інвестуванні у завдання, а не у технологію". Це основний принцип будь-якого фреймворка.

Фреймворк дозволяє уникнути витрат часу на розробку загальних компонентів, щоб повністю сконцентруватися на бізнес-правилах. Наприклад, фреймворк звільняє розробника від необхідності витратити 2-3 дні на створення форми автентифікації. Заощаджений час можна використати для більш конкретних завдань.

Використання фреймворка гарантує можливість оновлення та обслуговування в майбутньому, забезпечуючи довговічність розроблених програм.

Symfony Framework використовує PHP, і розроблені програми можуть взаємодіяти з іншими бібліотеками PHP.

Крім фреймворків, існують інші рішення для розробки веб-сайтів та додатків, такі як системи керування вмістом (CMS) та їхні модулі, а також пакетні професійні рішення (CRM, електронна комерція) і т.д.

Перш ніж прийняти рішення про використання існуючих рішень або розробку власного продукту, важливо докладно визначити поточні та майбутні потреби і порівняти різні рішення, які доступні на ринку.

1.3. Основи розробки за допомогою Symfony Framework

Фреймворк PHP представляє собою набір класів, призначених для сприяння розробці веб-програм.

Symfony, з свого боку, є відкритою інфраструктурою Model-View-Controller (MVC) для розробки сучасних веб-додатків, які розвиваються швидко. Цей фреймворк містить набір компонентів PHP, які можна використовувати повторно у програмах незалежно від платформи. У Symfony існують значні можливості та активна спільнота розробників. Він дозволяє гнучку настройку за допомогою YAML, XML або анотацій.

Symfony також інтегрується з різними незалежними бібліотеками та модулями PHP. Цей фреймворк, в першу чергу, натхненний ідеями, які здебільшого характерні для інших популярних засобів для розробки веб-додатків, таких як Ruby on Rails, Django та Spring.

Компоненти Symfony активно використовуються в численних проєктах з відкритим вихідним кодом, таких як Composer, Drupal та phpBB. Одним з важливих компонентів є HttpFoundation, який розуміє HTTP та надає зручний об'єкт для обробки запиту та відповіді, який може використовуватися іншими компонентами. Крім того, існують допоміжні компоненти, такі як Validator, які сприяють перевірці даних. Ядро компонента є серцем системи і відповідає за обробку HTTP-запитів та створення середовища для роботи додатків.

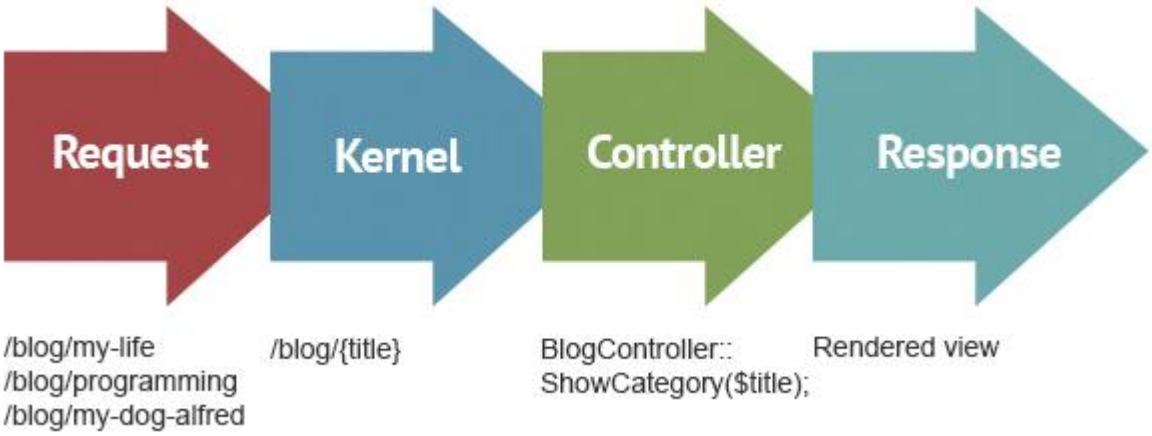


Рис. 1.8 Обробка http-запиту

Symfony має чудово організовану структуру, допомагаючи розробникам легко створювати веб-додатки. Цей фреймворк надає можливість писати чистий код та використовувати ефективні підходи до програмування. Більш того, Symfony є дуже гнучким і підходить як для створення невеликих веб-сайтів, так

і для опрацювання великих корпоративних додатків з великою кількістю користувачів.

Symfony ґрунтується на архітектурному підходу Model-View-Controller (MVC). У цій архітектурі:

- Модель (Model) відображає структуру даних та бізнес-логіку наших об'єктів.
- Перегляд (View) відображає дані з моделі користувачу у потрібному вигляді залежно від ситуації.
- Контролер (Controller) обробляє всі запити від користувача, виконує фактичну роботу, взаємодіє з моделлю та надає перегляду необхідні дані для відображення користувачеві.

Ця архітектурна модель допомагає легко організовувати розробку та підтримувати чіткий розділ обов'язків між компонентами системи або веб-застосунку.

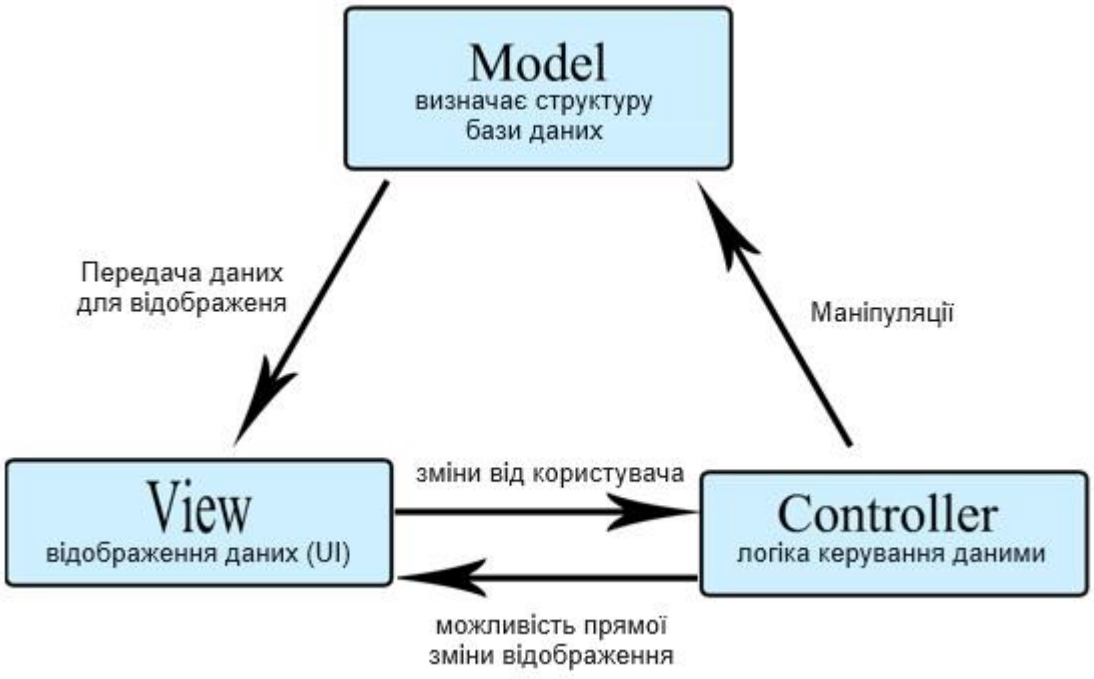


Рис. 1.9 Архітектура Model-View-Controller (MVC)

Крім того, Symfony надає безліч функцій:

- валідатори (Validator) використовуються для валідації даних;
- форми (Type) використовуються для створення форм;

- сервіси (Service) відповідають за опис бізнес процесів та іншої логіки;
- слухачі (Listener та Event Subscriber) можуть перехоплювати події (Event) та додавати додатковий функціонал до них;
- перекладач (Translator) забезпечую підтримку мултимовність проєкту;
- тощо

При роботі з базами даних Symfony використовує Doctrine для об'єктного реляційного відображення (ORM).

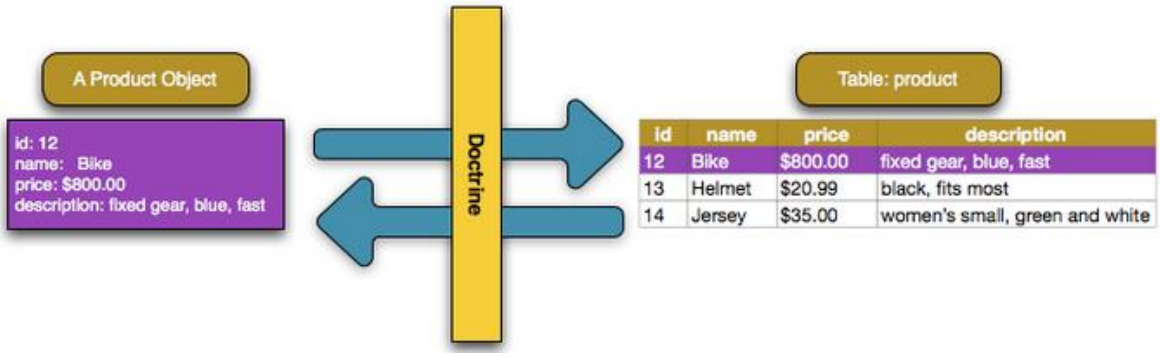


Рис. 1.10 Об'єктно реляційне відображення (ORM)

Дуже зручним механізмом розгортання та зміни структури та даних баз даних в Symfony є міграції.

Іншим ключовим моментом є те, що він надає засоби розробки для реєстрації, тестування та кешування.

Symfony спрямований на прискорення створення та підтримки веб-застосунків, а також для уникнення витрат часу для розв'язування тривіальних задач у розробці (наприклад, написання валідаторів форм).

Symfony ставить за мету дати розробникам повний контроль над конфігурацією: майже все можливо налаштувати, від структури каталогів до сторонніх бібліотек.

Symfony використовує систему шаблонів Twig.

Сутність (Entity). В Symfony моделям відповідають сутності (Entity).

У фреймворку Symfony, сутність (Entity) представляє собою клас, який відображає об'єкт або таблицю в базі даних. Сутності використовуються для моделювання структури даних вашого додатку і взаємодії з базою даних. Кожен

екземпляр сутності відображає конкретний запис в базі даних і містить властивості (полі), які відповідають колонкам в таблиці бази даних.

Symfony надає інструменти для опису сутностей за допомогою анотацій або XML-конфігурації. Клас сутності може також включати методи для виконання операцій над цією сутністю, таких як збереження, оновлення і видалення.

Використання сутностей дозволяє Symfony автоматично створювати SQL-запити для взаємодії з базою даних і спрощує роботу з даними в вашому додатку. Вони є важливою частиною інфраструктури ORM (Object-Relational Mapping) Symfony, яка спрощує роботу з базами даних і дозволяє розробникам працювати з даними як з об'єктами в коді.

Для зручності та пришвидшення написання коду в Symfony присутній дуже зручний інструмент – генератор. Він може генерувати код для багатьох елементів фреймворка, як то сутності, контролери, міграції тощо.

Для створення класу User та будь-яких полів, які нам потрібні, можна використати команду make:entity. Приклад використання цієї команди наведено на рис. 1.11.

```
$ php bin/console make:entity

Class name of the entity to create or update:
> User

New property name (press <return> to stop adding fields):
> name

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

Can this field be null in the database (nullable) (yes/no) [no]:
> no

New property name (press <return> to stop adding fields):
> lastname

Field type (enter ? to see all types) [string]:
> string

Can this field be null in the database (nullable) (yes/no) [no]:
> no
```

Рис. 1.11 Використання команди make:entity

Результатом роботи цієї команди буде новий файл src/Entity/User.php (див. Рис 1.12).

Цей клас називається «сутністю». Він описую об’єкт User, його поля для зберігання інформації, тип цієї інформації, структуру таблиці в базі даних. Також він описує типи доступи до даних, зв’язки з іншими сутностями тощо. В класі сутності навіть можна описати правила валідації даних при роботі с формами.

Тепер ми зможемо зберігати об’єкти User і запитувати їх у таблиці користувачів у своїй базі даних.

Кожну властивість в сутності продукту можна зіставити зі стовпцем у цій таблиці. Зазвичай це робиться за допомогою атрибутів:

#[ORM\Column]

Якщо програміст хоче використовувати інший спосіб опису полів в базі даних, то Symfony підтримую і це. Наприклад, якщо ми хочемо використовувати XML замість анотацій, то все що нам потрібно, це додати type:xml і dir: '%kernel.project_dir%/config/doctrine' до зіставлення сутностей у нашому файлі config/packages/doctrine.yaml.

```php
use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
#[ORM\Entity(repositoryClass:
UserRepository::class)]
class User
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    private ?string $name = null;

    #[ORM\Column]
    private ?string $lastname = null;

    public function getId(): ?int
    {
        return $this->id;
    }
    // ... getter and setter methods
}
```

Рис. 1.12 Код сутності User

Doctrine підтримує широкий спектр типів полів, кожне зі своїми параметрами.

Команда make:entity — це інструмент, який полегшує життя. Однак, також необхідно створити код для виконання таких завдань, як додавання або видалення полів, методів або оновлення конфігурації. Але завдяки використанню Symfony багато процесів вже реалізовані за нас. Крім того, вони навіть можна сказати, приховані від нас. В об’єктно орієнтованому програмуванні (ООП) це називається інкапсуляцією.

Перегляд (View). У фреймворку Symfony, перегляд (View) представляє собою складову частину шаблону, яка відповідає за відображення і візуалізацію даних користувачу. Перегляди в Symfony використовуються для відображення вмісту веб-сторінок і додатків, що відображають інформацію, яка передається з контролера.

Перегляди можуть бути створені у формі шаблонів, які містять HTML, CSS, JavaScript та спеціальні теги або команди для вставки даних і керування виглядом сторінки. Symfony підтримує різні движки для створення переглядів, такі як Twig, PHP, та інші.

Перегляди розділені від контролера, що дозволяє відокремити бізнес-логіку від візуального представлення. Це сприяє більшій структурної чистоті та розширюваності коду. Після обробки запиту контролером, він може передавати дані перегляду, який потім відповідає за генерацію HTML-сторінки або іншого вмісту, який відображається користувачу в браузері.

Перегляди допомагають розділити логіку додатку і його представлення, що спрощує розробку та підтримку додатків Symfony.

Шаблон — це дуже зручний спосіб упорядкувати та відобразити HTML у програмі, незалежно від того, чи потрібно нам відобразити HTML із контролера чи створити вміст електронного листа.

Twig - це дуже гнучкий, ефективний і безпечний інструмент для створення шаблонів.

Мова шаблонів Twig дозволяє створювати компактні та читабельні шаблони, які виявляються більш зрозумілими для веб-дизайнерів і, в певних відношеннях, мають більшу потужність, ніж шаблони PHP.

Приклад шаблону Twig представлено нижче (див. Рис. 1.13).

Синтаксис Twig базується на трьох конструкціях:

{{ ... }}, використовується для відображення вмісту змінної або результату обчислення виразу;

{% ... %}, використовується для запуску певної логіки, наприклад умови або циклу;

{# ... #}, який використовується для додавання коментарів до шаблону (на відміну від коментарів HTML, ці коментарі не включені до відтвореної сторінки).

```
<!DOCTYPE html>

<html>
    <head>
        <title>Welcome to Symfony!</title>
    </head>

    <body>
        <h1>{{ page_title }}</h1>

        {% if user.isLoggedIn %}
            Hello {{ user.name }}!
        {% endif %}

        {# ... #}
    </body>

</html>
```

Рис. 1.13 Приклад шаблону Twig

Twig не дає можливості запустити PHP-код у шаблонах, але він надає утиліти для запуску певної логіки в шаблонах. Вони поділяються на теги (Tags), фільтри (Filters), функції (Functions), оператори (Operators) та тести (Tests).

Наприклад, фільтри змінюють вміст перед відтворенням. Як наведено у коді вище, фільтр upper змінює вміст змінної tittle у верхній регістр:

{{ title|upper }}

Twig поставляється з довгим списком тегів, фільтрів і функцій, доступних за замовчуванням. У програмах Symfony ми також можемо використовувати ці фільтри та функції Twig, визначені Symfony. Крім цього ми можемо створювати власні фільтри та функції Twig.

Twig швидкий у продакшені (середовище prod), оскільки шаблони компілюються в PHP і автоматично кешуються. Але одночасно він зручний у використанні при розробці (середовище dev), оскільки шаблони автоматично перекомпілюються, коли ми їх змінюємо.

Twig має кілька параметрів конфігурації, щоб визначити такі речі, як формат, що використовується для відображення чисел і дат, кешування шаблону тощо.

Для прикладу, покажемо на сторінці профілю користувача його ім’я та кількість нових повідомлень. Для цього необхідно створити файл шаблону, який буде містити код для відображення цієї інформації та файл контролера, який буде рендерити цей шаблон. Файли шаблонів створюється у каталозі templates/. Створимо notifications.html.twig в подкаталозі user, код якого наводиться на рис. 1.14.

```
{# templates/user/notifications.html.twig #}

<h1>Hello {{ user_first_name }}!</h1>
<p>
    You have {{ notifications|length }} new notifications.
</p>
```

Рис. 1.14 Шаблон notifications.html.twig

Шаблони Twig підтримують наслідування, що значно спрощую написання інших шаблонів та надає дуже широкі можливості. Наведений код представляє собою базовий шаблон (template) у форматі Twig. Розглянемо його структуру та функції (рис. 1.15).

```
{# templates/base.html.twig #}
<!doctype html>
<html lang="en">
<head>
    {% block head %}
        <title>{% block title %}{% endblock %}</title>
        <meta name="description" content="
            {% block description %}{% endblock %}">
        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}
        {% block javascripts %}
            {{ encore_entry_script_tags('app') }}
        {% endblock %}
        {% block keywords %}{% endblock %}
        {% block og %}{% endblock %}
    {% endblock %}
</head>
<body>
{% block header %}{% endblock %}
{% block content %}{% endblock %}
</body>
</html>
```

Рис. 1.15 шаблон notifications.html.twig

{# templates/base.html.twig #}: Ця частина представляє собою коментар у Twig і вказує на шлях до файлу шаблону у проєкті.

<!doctype html> та інші HTML-теги: Ці теги визначають загальну структуру сторінки, включаючи мову, заголовок і мета-теги. Ця частина сторінки включає всі необхідні мета-теги та посилання на зовнішні ресурси.

{% block head %} ... {% endblock %}: Це блок Twig, який дозволяє вкладати вміст, що відображається в <head> сторінки. У цьому блоку можна визначити заголовок сторінки, мета-теги, таблиці стилів і JavaScript-файли. Вміст цього блоку може бути розширений в дочірніх шаблонах.

{% block title %}{% endblock %}: Це вкладений блок в блоку head, який визначає заголовок сторінки. В цьому блоці можна встановити конкретний заголовок в дочірньому шаблоні.

{% block description %}{% endblock %}: Це вкладений блок, що дозволяє визначити опис сторінки для мета-тега "description". Знову ж таки, цей блок може бути заповнений в дочірньому шаблоні.

{% block stylesheets %} ... {% endblock %}: Цей блок відповідає за включення таблиць стилів (CSS) для сторінки. Використовується {{ encore_entry_link_tags('app') }}, що дозволяє підключити CSS, визначений для сторінки з використанням Symfony Encore або іншого інструменту.

{% block javascripts %} ... {% endblock %}: Аналогічно до блоку стилів, цей блок відповідає за включення JavaScript-файлів на сторінці за допомогою {{ encore_entry_script_tags('app') }}.

{% block header %}{% endblock %} та {% block content %}{% endblock %}: Ці блоки призначені для вставки вмісту заголовка і тіла сторінки відповідно. Вони можуть бути розширені в дочірніх шаблонах, де можна визначити конкретний вміст для заголовка та сторінки.

Загальна ідея цього шаблону полягає в тому, щоб мати загальну структуру сторінки, але надавати можливість розширення і налаштування для конкретних сторінок у дочірніх шаблонах. Twig дозволяє структурувати та організовувати ваш веб-вміст у шаблонах, забезпечуючи вищий рівень гнучкості та рефакторингу проєкту.

Код на рис. 1.16 представляє собою шаблон Twig, який розширює і встановлює конкретні блоки для створення конкретної сторінки.

```
{% extends 'base.html.twig' %}

{% block title %}Головна сторінка{% endblock %}
{% block description %}Головна сторінка{% endblock %}
{% block header %}
    Site logo and header menu
{% endblock %}
{% block content %}Вміст сторінки{% endblock %}
```

Рис. 1.16 Шаблон notifications.html.twig

Ось опис кожного рядка цього коду:

{% extends 'base.html.twig' %}: Ця інструкція вказує, що поточний шаблон розширює (наслідує) шаблон, який має шлях 'base.html.twig'. Це означає, що весь вміст базового шаблону (base.html.twig) буде утримувати загальну

структуру сторінки, але може бути розширений конкретним вмістом, який визначається в поточному шаблоні.

{% block title %}{% endblock %}: Цей рядок встановлює вміст блоку title. Вміст в цьому блоку буде використаний для визначення заголовка сторінки. У цьому випадку заголовок встановлюється як "Головна сторінка".

{% block description %}{% endblock %}: Схожий на попередній рядок, цей рядок встановлює вміст блоку description. Вміст в цьому блоку буде використаний для визначення мета-тега "description" сторінки.

{% block header %}{% endblock %}: Цей рядок встановлює вміст блоку header. Вміст в цьому блоку буде використаний для визначення заголовку сторінки, наприклад, для розміщення логотипу та меню заголовка сторінки.

{% block content %}{% endblock %}: Цей рядок встановлює вміст блоку content. Вміст в цьому блоку буде використаний для визначення основного вмісту сторінки.

Цей шаблон дозволяє вставити конкретний вміст у відповідні блоки і отримати сторінку із заздалегідь визначеною структурою, такою як заголовок, опис, заголовок сторінки та основний вміст.

Шаблони Twig можуть бути використані як складові частини інших шаблонів використовуючи ключове слово extends, як показано на рис. 1.17.

```
{% extends 'base.html.twig' %}
```

Рис. 1.17 Наслідування одного шаблону від іншого

Цей код використовує Twig-директиву {% include %} для включення вмісту іншого Twig-шаблону в поточний шаблон. Ось опис цього коду:

{% include 'user/notifications.html.twig'%}: Цей рядок вказує Twig на те, що потрібно включити вміст шаблону, який знаходиться за шляхом 'user/notifications.html.twig'. Тобто, вміст зазначеного шаблону буде вставлений прямо в поточний шаблон на місці цієї директиви.

Цей підхід дозволяє вставляти інші шаблони або їх частини в поточний шаблон, щоб компонувати складні сторінки або використовувати підключення

зокрема для рефакторингу коду та підтримки більшої гнучкості при створенні веб-сторінок. В одному шаблоні може бути багато таких {% include %} директив для розбиття сторінки на компоненти або включення загальних елементів на кілька сторінок.

Потім створимо контролер, який рендерить цей шаблон і передає йому необхідні змінні. Файл шаблону створюється у каталозі src/Controller/.

Контролер (Controller). У фреймворку Symfony, контролер (Controller) - це складова частина додатка, яка відповідає за обробку вхідних HTTP-запитів і прийняття рішень щодо відповіді на ці запити. Контролери виконують функцію посередника між вхідним запитом користувача і внутрішньою логікою додатка.

Основним завданням контролера є взяття вхідного запиту, аналіз його параметрів, ініціювання відповідних операцій, обробка даних і визначення, яким чином відповідь має бути сформована. Контролер вирішує, які дані повинні бути відправлені до шаблону для відображення інформації на сторінці.

У Symfony, контролери зазвичай представлені у вигляді класів PHP, які реалізують відповідний інтерфейс або успадковуються від базових контролерів, надаючи можливість визначити методи, які відповідають за різні дії (Action) або маршрути (Routes) додатку. Ці методи обробляють запити і виконують необхідні дії, після чого повертають відповідь, яку потім фреймворк відправляє користувачеві.

Контролери допомагають відокремити логіку обробки запитів від логіки відображення і надають можливість ефективно керувати всією логікою додатка в масштабах веб-проєктів.

У той час як контролер це зазвичай PHP клас, дія зазвичай є методом у класі контролера. Дія — це метод mainPageAction(), який знаходиться в класі контролера MainController, код якого відображено на рис. 1.18.

```php
// src/Controller/MainController.php
namespace App\Controller;

use Symfony\Component\Security\Core\User\UserInterface;
use
Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class MainController extends AbstractController
{
    // ...

    public function mainPageAction(UserInterface $user =
null): Response
    {
        // отримуємо інформацію про користувача та про його
        // повідомлення
        $userFirstName = $user->getFirsname();
        $userNotifications = $user->getNotifications();

        // шлях до шаблона, відносно каталогу /templates
        return $this->render('main_page.html.twig', [
            // передаємо змінні у шаблон
            'user_first_name' => $userFirstName,
            'notifications' => $userNotifications,
        ]);
    }

    // ...
}
```

Рис. 1.18 контролер MainController.php

Даний код представляє собою контролер Symfony у файлі MainController.php, який відповідає за обробку запитів і відправку відповідей на основну сторінку веб-додатку. Розглянемо кожну частину коду:

namespace App\Controller;: Цей рядок визначає простір імен (namespace) для класу контролера, який міститься у цьому файлі.

use Symfony\Component\Security\Core\User\UserInterface;: Ця директива use імпортує клас UserInterface з бібліотеки Symfony для подальшого використання його в методі контролера.

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;: Ця директива use імпортує базовий контролер AbstractController з Symfony, який буде використовуватися для створення власного контролера.

use Symfony\Component\HttpFoundation\Response;: Ця директива use імпортує клас Response з Symfony для створення HTTP-відповіді.

class MainController extends AbstractController: Визначає клас MainController, який успадковує функціональність з базового контролера AbstractController.

public function mainPageAction(UserInterface $user = null): Response: Цей метод, mainPageAction, відповідає за обробку запитів на основну сторінку. Він приймає об'єкт користувача ($user), який реалізує інтерфейс UserInterface, або може бути null, якщо користувач не авторизований. Метод повертає об'єкт класу Response, який представляє собою відповідь сервера.

В тілі методу отримується інформація про користувача, така як його ім'я та сповіщення.

return $this->render('main_page.html.twig', [...]): Цей рядок відправляє відповідь, використовуючи шаблон main_page.html.twig, і передає в нього змінні для відображення. Змінні, такі як user_first_name та notifications, передаються у шаблон для відображення на сторінці.

Отже, цей контролер обробляє запити на основну сторінку, отримує дані користувача та повідомлень, та відправляє сторінку з відповідним вмістом у відповідь.

Висновки до розділу 1

В цьому розділі я обґрунтував необхідність використання фреймворка в цілому та вибір фреймворка Symfony Framework зокрема.

Symfony є інфраструктурою, яка використовує парадигму Model-View-Controller (MVC). Це дозволяє логічно поділити додаток на моделі, представлення та контролери для кращого керування.

Парадигма "інвестування в завдання, а не в технологію": Symfony ставить акцент на розробку, яка спрямована на вирішення конкретних завдань та відповідає бізнес-потребам, а не використовується безконтрольно для технологічних експериментів.

Symfony надає компоненти PHP, які можна використовувати повторно для розробки різних аспектів додатка. Це спрощує створення структурованого коду та допомагає розробникам зосередитися на реалізації бізнес-правил.

Гнучка конфігурація Symfony дозволяє використовувати конфігураційні файли у форматах, таких як YAML, XML або анотації. Це дозволяє адаптувати фреймворк до конкретних потреб проєкту.

Symfony підтримує інтеграцію з різними бібліотеками та модулями PHP, що розширює можливості фреймворка та дає розробникам доступ до багатьох сторонніх рішень.

Був розглянутий ряд популярних PHP фреймворків. На ринку існує багато фреймворків, таких як Symfony, Laravel, CodeIgniter, Yii, тощо. Кожен фреймворк призначений для розв'язання конкретних завдань та має свої унікальні особливості. Symfony виділяється великою кількістю функціональних можливостей та активною спільнотою розробників, які використовують цей фреймворк для створення веб-додатків. Використання фреймворків має великий спектр переваг, які були наведені в цьому розділі.

Були розглянуті основи розробки за допомогою Symfony. Цей фремворк базуються на архітектурі MVC, де модель представляє структуру даних, представлення показує дані користувачу, а контролер обробляє запити користувачів та взаємодіє з моделлю.

Загалом, фреймворки, такі як Symfony, роблять розробку програмних продуктів більш структурованою та продуктивною, але вимагають розуміння їхньої роботи та концепцій, які лежать в основі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБЛІКУ РОБОЧОГО ЧАСУ РОБІТНИКІВ ПІДПРИЄМСТВА

Метою є розробка програмного забезпечення для автоматизованої системи обліку робочого часу працівників на підприємстві.

Автоматизована система обліку робочого часу працівників на підприємстві складається з трьох основних частин: серверної (бекенд), користувацької (фронтенд) та системи зберігання даних. Серверна частина відповідає за обробку даних, користувацька - за взаємодію користувача системи з цими даними, система зберігання відповідає за зберігання даних.

Схема веб-застосунку - це абстрактне представлення структури та взаємозв'язків всіх складових елементів веб-застосунку. Така схема допомагає візуалізувати, як веб-застосунок працює та як взаємодіє з різними компонентами. Основні елементи веб-застосунку:

Клієнт: Клієнт - це кінцевий користувач веб-застосунку. Це може бути веб-браузер на комп'ютері, мобільний додаток на смартфоні або інше пристрій, що звертається до веб-застосунку через мережу Інтернет внутрішню мережу підприємства (інтранет).

Сервер: Сервер - це комп'ютер або серверна ферма, на якому розміщується веб-застосунок. Він відповідає за обробку запитів від клієнтів та надсилання їм відповідей.

Веб-застосунок: Це основна частина, яка розробляється. Веб-застосунок містить серверний код, базу даних, статичні файли, які необхідні для роботи додатку.

База даних: База даних використовується для збереження інформації, яка може бути доступною з веб-застосунку. Вона зберігає дані, які можуть бути витягнуті та відображені для користувачів.

Серверна сторона (Backend): Серверна сторона включає в себе серверний код, який обробляє запити від клієнтів, взаємодіє з базою даних та надсилає

відповіді клієнтам. Це може бути написано на різних мовах програмування, таких як PHP, Python, Ruby, Java, тощо.

Клієнтська сторона (Frontend): Клієнтська сторона включає в себе код, який виконується в браузері клієнта. Вона відповідає за відображення вмісту та інтерактивність для користувачів. Зазвичай використовуються мови, такі як HTML, CSS та JavaScript для створення клієнтської сторони.

Мережа: Мережа передає запити та відповіді між клієнтом та сервером. Це може бути Інтернет або будь-яка інша мережа, яка з'єднує обидві сторони.

Проксі та балансери навантаження: Для забезпечення надійності та масштабованості додатку, можуть використовуватися проксі-сервери та балансери навантаження, які розподіляють запити між декількома серверами.

Зовнішні сервіси та API: Додаток може взаємодіяти з іншими зовнішніми сервісами та API, які надають додатковий функціонал або дані.

Ця схема ілюструє загальну структуру веб-застосунку та взаємодію між його складовими частинами. Клієнти надсилають запити на сервер, який відповідає на них, оброблюючи їх за допомогою серверного коду та отримуючи доступ до бази даних. Клієнтська сторона відображає інформацію для користувачів та дозволяє їм взаємодіяти з додатком через браузер.

Програмний продукт, що розробляється повинен виконувати наступні основні задачі:

- облік працівників підприємства;
- облік робочого часу кожного працівника підприємства;
- облік відпусток, відгулів та лікарняних працівників підприємства;
- мати особистий кабінет для кожного працівника підприємства;
- кожен працівник повинен мати роль в системі та відповідні права та функціонал;
- формувати звітність про робочий час працівника: загальна кількість робочих годин, кількість не робочих годин, кількість годин витрачених на кожну конкретну задачу, кількість годин витрачених на кожного окремого клієнта;

- додаткова функціональність;

Для розробки програмного забезпечення для автоматизованої системи обліку робочого часу працівників на підприємстві мною був обраний наступний стек технологій:

- мова програмування PHP 8;
- мова програмування JavaScript;
- мова розмітки гіпертексту HTML;
- мова стилів CSS (SCSS);
- PHP фреймворк Symfony;
- середовище розробки для PHP PhpStorm;
- додаткові бібліотеки та технології (docker, composer, webpack, JQuery та інші;

З точки зору алгоритмічного забезпечення автоматизована система обліку робочого часу працівників на підприємстві складається також з трьох основних частин:

- робочий простір працівника підприємства;
- робочий простір адміністратора системи;
- системи зберігання даних;

Процес входу до автоматизованої системи з використанням сторінки авторизації та подальших дій користувача буде наступним:

Сторінка авторизації: Перший крок - це сторінка авторизації, на якій користувач повинен ввести свої ідентифікаційні дані, такі як ім'я користувача та пароль. Сторінка авторизації служить як точка входу до системи та перевіряє, чи користувач має доступ до автоматизованої системи.

Автентифікація: Коли користувач вводить свої ідентифікаційні дані, система виконує процес автентифікації, щоб перевірити, чи вони є валідними. Це включає в себе перевірку пароля та ім'я користувача в базі даних. Якщо дані валідні, користувачу дозволяється увійти в систему.

Перенаправлення до простору працівника або адміністратора: Після успішної авторизації система вирішує, до якої частини системи (простору)

перенаправити користувача. Це залежить від прав доступу користувача. Якщо користувач є звичайним працівником, його перенаправлять до простору працівника, де він зможе виконувати свої завдання та взаємодіяти з даними, що його цікавлять. Якщо користувач має адміністративні права, його перенаправлять до простору адміністратора системи, де він може керувати налаштуваннями, додавати користувачів, встановлювати права доступу тощо.

Взаємодія з системою: Після перенаправлення користувач взаємодіє з системою відповідно до своїх прав доступу та завдань. Він може переглядати дані, вводити інформацію, редагувати записи, створювати звіти тощо.

Завершення сесії: Коли користувач завершує роботу з системою, він може вийти зі свого облікового запису або закрити браузер. Система автоматично завершує сесію та виводить користувача з системи.

Це загальний процес входу та взаємодії з автоматизованою системою, де користувач має можливість обирати між простором працівника та адміністратора в залежності від його прав доступу та потреб.

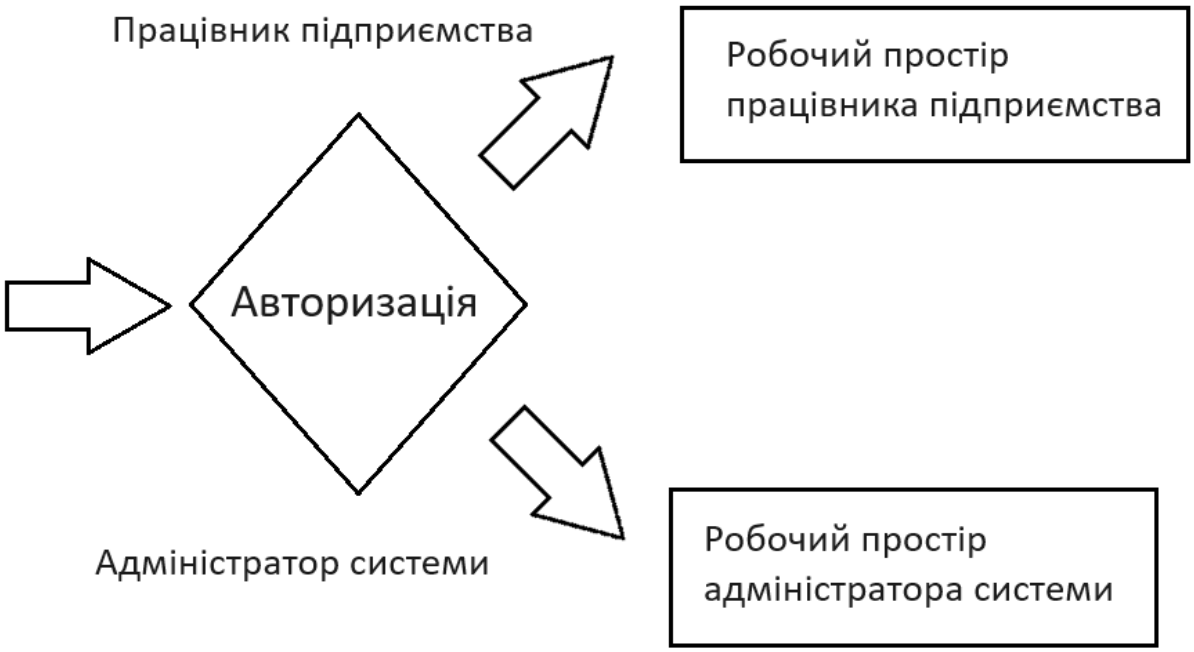


Рис. 2.1 Точка входу

2.1. Проєктування робочого простору працівника підприємства

Після успішної авторизації користувача системи, якщо він має статус працівника підприємства, система перенаправить його до робочого простору працівника підприємства. Нижче наведена схема функціональності робочого простору працівника підприємства:

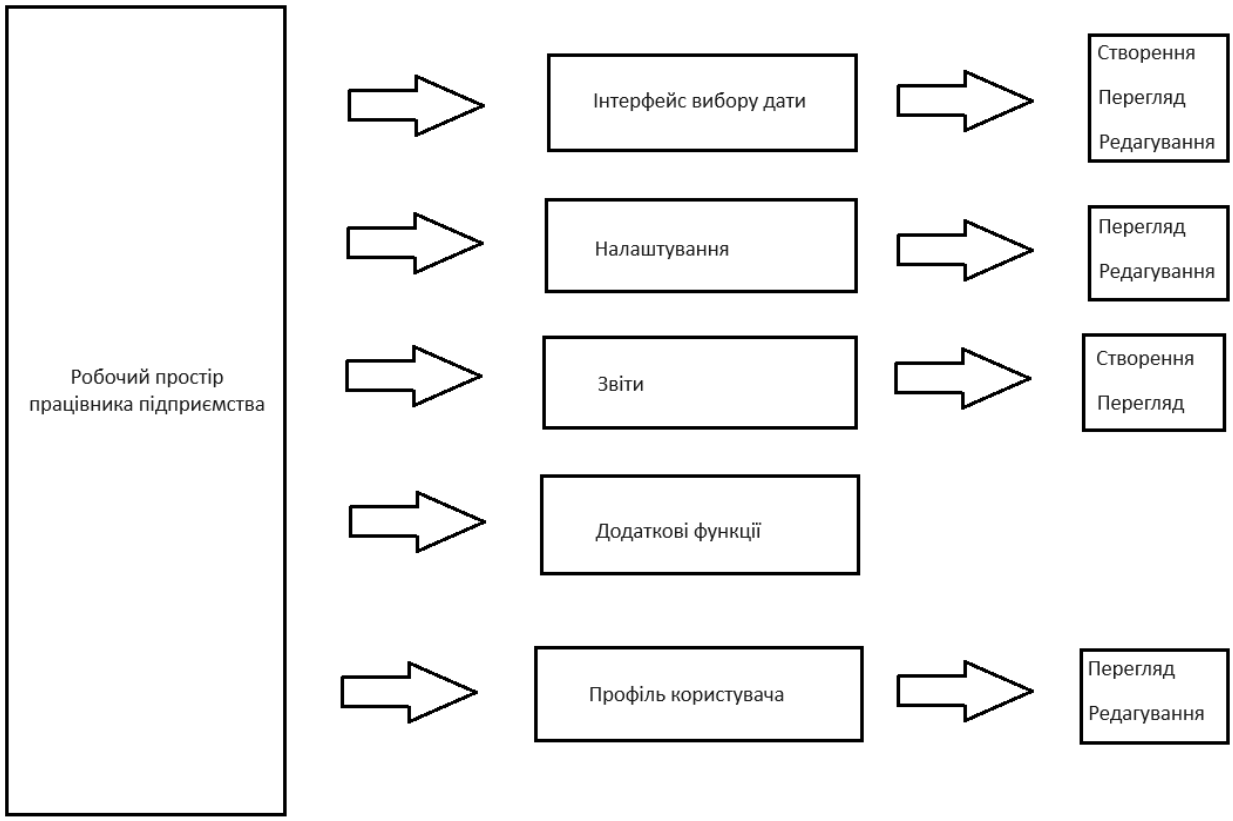


Рис. 2.2 Алгоритмічне забезпечення робочого простору працівника підприємства

Розглянемо більш детально кожен з розділів системи:

Робочий простір працівника підприємства. У робочому просторі працівник може вносити дані про свій робочий час, а саме про виконану роботу та час, затрачений на її виконання.

Працівник має можливість переглядати історію його робочого часу, свої попередні записи про робочий час для внесення корекцій або перевірки. Якщо виникла необхідність, працівник може вносити корективи у свої записи про робочий час, наприклад, у випадку помилки.

Інтерфейс вибору дати. Надає можливість обрати дату, з якою будуть проводитися дії (перегляд, створення та редагування записів).

Налаштування інтерфейсу та системні налаштування. Користувач може змінювати налаштування інтерфейсу для кращого адаптування до своїх потреб.

Звіти. Система надає можливість генерувати звіти на основі внесених даних, такі як звіти про робочий час працівника, про лікарняні, відпустку, відгули, відпрацювання тощо. Це дозволяє відслідковувати важливі дані щодо робочого часу працівника. Вже згенеровані звіти доступні для перегляду та зберігання.

Додаткові функції.

Інтеграція та взаємодія з іншими системами дозволяє обмін даними та координацію між різними системами, такими як Jira, календар Google тощо.

Імпорт даних про користувачів з друкованих форм: процес завантаження інформації про працівників підприємства із паперових форм або інших документів у систему автоматизованої системи. Цей процес передбачає конвертацію даних із фізичних носіїв, таких як анкети, заяви, анкети про працевлаштування чи інші документи, у цифровий формат для подальшого зберігання та обробки в системі.

Підрахунок заробітної плати та соціального пакету: розрахунок фінансових параметрів на основі робочого часу та правил підприємства.

Перегляд інформації про інших працівників та їх контактні дані: доступ до інформації про співробітників підприємства, до якої працівник має доступ.

Створення запиту до адміністрації: можливість користувачів звертатися до адміністрації з питаннями, пропозиціями або проханнями.

Інформаційний розділ компанії: доступ до додаткової інформації про підприємство та його процеси.

Профіль користувача. Користувач може переглядати та редагувати власний профіль, включаючи особисті дані, контактну інформацію тощо.

Ці функції та можливості робочого простору системи дозволяють користувачам зручно та ефективно управляти своїм робочим часом, отримувати необхідну інформацію та взаємодіяти з іншими аспектами діяльності підприємства.

На рис. 2.3 наведено схематичне відображення інтерфейсу робочого простору працівника підприємства. Він включає в себе такі елементи:

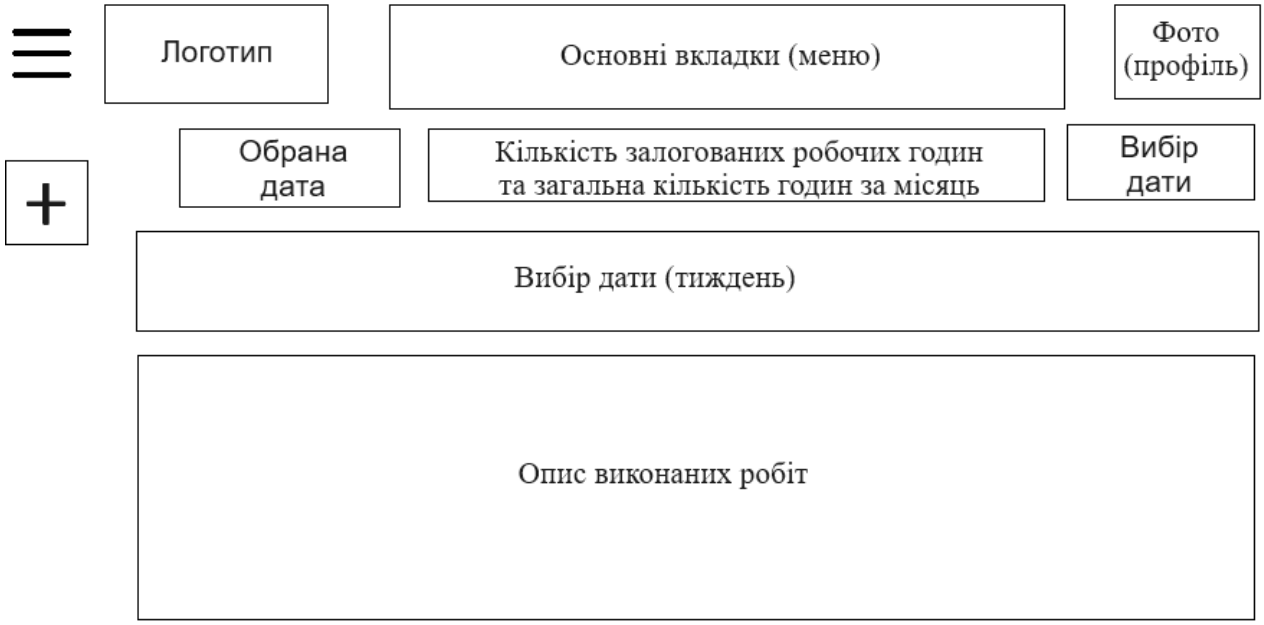


Рис. 2.3 Інтерфейсу робочого простору працівника підприємства

1. Головне меню (☰). Включає в себе посилання для переходу до додаткових функцій, як перегляд інформації про інших працівників підприємства, створення запиту до адміністрації, інформаційний розділ компанії тощо.
2. Логотип. Містить у собі назву підприємства або автоматизованої системи та логотип.
3. Основні вкладки (меню) містять посилання на робочий простір працівника, інтерфейс звітів та інші основні розділи системи.
4. Фото (профіль) містить фотокартку працівника, посилання на профіль користувача та вихід із системи.
5. Кнопка з зображення знаку плюс (+) відкриває форму для додавання запису про виконану роботу, відображену на рис. 2.4.

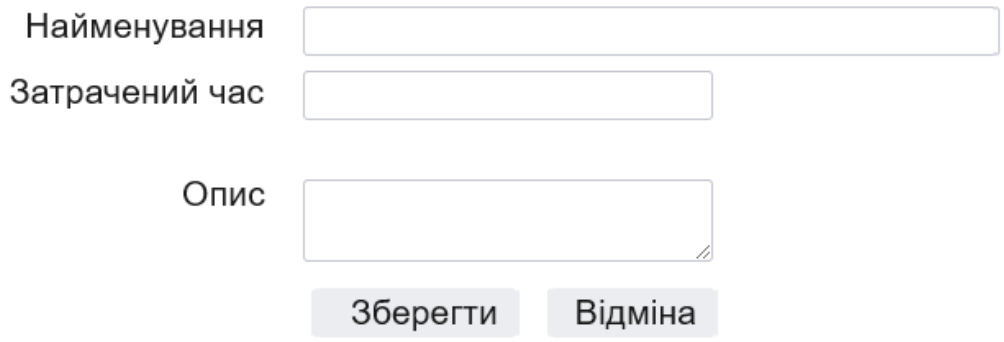


Рис. 2.4 Форма додавання запису про робочий час

6. Обрана дата відображає дату, з якою відбувається взаємодія.
7. Елемент кількість залогованих робочих годин та загальна кількість годин за місяць містить інформацію про те, на скільки залогований час відповідає загальному часу за місяць.
8. Елемент вибору дати (загальний) надає можливість обрати дату за межами тижня. Його вигляд зображено на рис. 2.5.

Рис. 2.5 Інтерфейс вибору дати

9. Вибір дати (тиждень) надає можливість обрати дату в межах тижня.
10. Опис виконаних робіт містить інформацію про виконану роботу та кількість затрачених годин на її виконання.

2.2. Проєктування робочого простору адміністратора системи

Процес авторизації користувача з правами адміністратора системи передбачає перенаправлення його до простору адміністратора системи. На рис. 2.6 наведено схему робочого простору адміністратора системи.

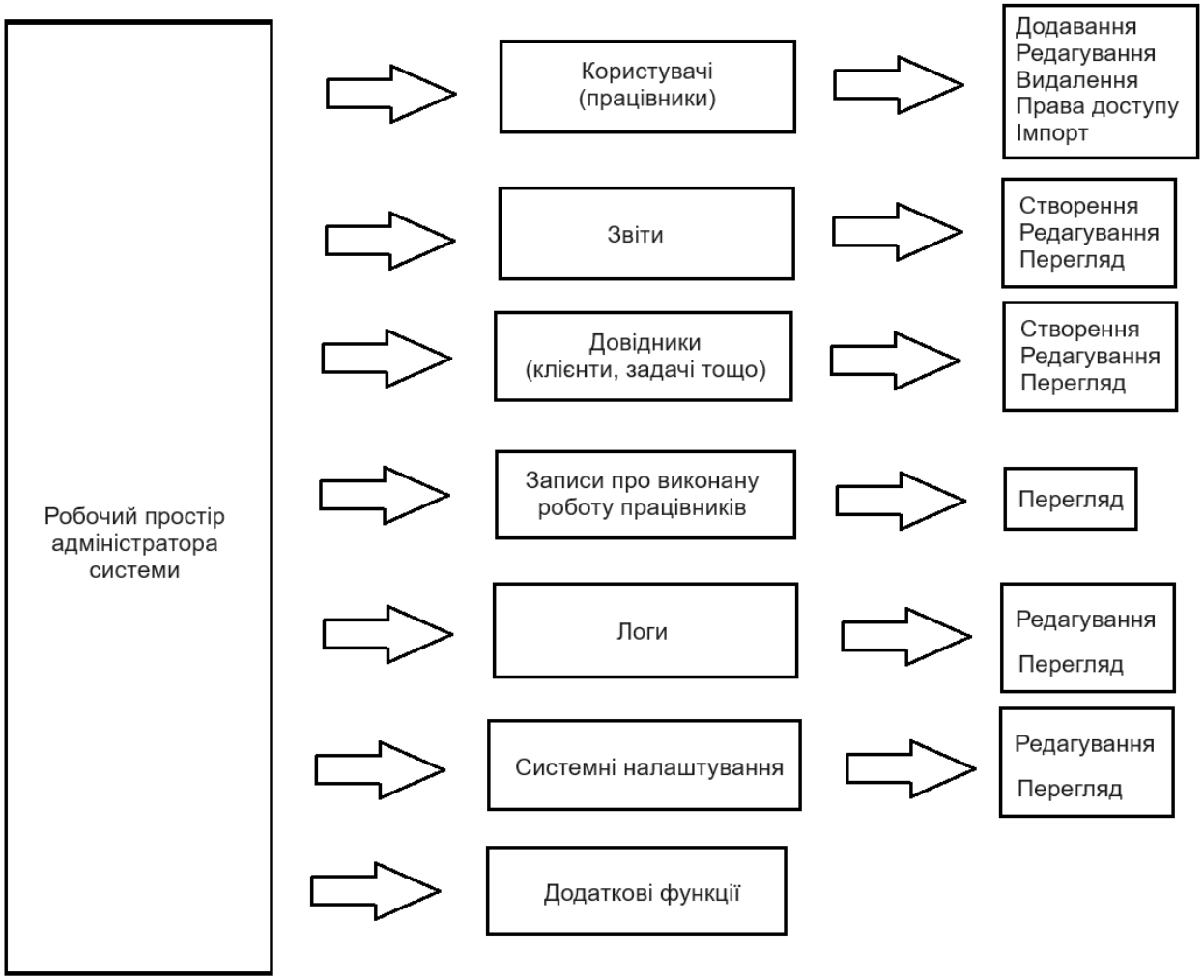


Рис. 2.6 Алгоритмічне забезпечення робочого простору адміністратора системи

Робочий простір адміністратора системи - це спеціальний інтерфейс, який надає адміністратору повний контроль над автоматизованою системою обліку робочого часу працівників на підприємстві. Нижче подано більш детальний опис кожного з елементів робочого простору адміністратора.

Користувачі (працівники). Адміністратор може створювати нові облікові записи для працівників підприємства. Це включає в себе введення особистих даних, таких як ім'я, прізвище, контактна інформація та інше.

Адміністратор може змінювати інформацію про користувачів, включаючи їхні права доступу та ролі в системі.

Адміністратор має можливість видаляти користувачів, якщо вони більше не працюють на підприємстві або якщо це необхідно з інших причин.

Адміністратор може надавати та керувати правами доступу різних користувачів до різних функціональних частин системи.

Система дозволяє імпортувати дані про користувачів з інших джерел або з друкованих форм для швидкого створення облікових записів.

Звіти. Адміністратор може генерувати різні звіти про робочий час працівників, включаючи інформацію про відгули, відпустки, лікарняні, відпрацювання та інше.

Згенеровані звіти доступні для перегляду в інтерфейсі. Адміністратор може аналізувати статистику робочого часу працівників та приймати управлінські рішення на підставі цих даних.

Звіти можуть бути надіслані по електронній пошті або надруковані для подальшого архівування та розповсюдження.

Клієнти, компанії, задачі. Адміністратор може додавати нових клієнтів та компанії до системи. Це допомагає в управлінні замовленнями та проєктами, пов'язаними з цими клієнтами.

Адміністратор може редагувати інформацію про клієнтів, компанії та задачі, а також видаляти їх, коли це необхідно.

Адміністратор може додавати та назначати користувачам задачі для виконання.

Записи про виконану роботу. Адміністратор може переглядати записи про виконану роботу працівниками підприємства. Це допомагає відстежувати продуктивність та результати роботи.

Довідкова система підприємства. Адміністратор може створити інформаційний розділ, де розміщується важлива інформація для працівників підприємства, така як правила та політика компанії. Крім того адміністратор надає доступ до важливої інформації про підприємство, що може бути корисною для користувачів системи.

Логи системи. Адміністратор має можливість переглядати логи системи для відстеження дій користувачів та подій, які відбуваються в системі.

Системні налаштування. Адміністратор може налаштовувати різні параметри системи, включаючи часові зони, робочі графіки, налаштування електронної пошти та інше.

Додаткові функції. Робочий простір адміністратора містить інші додаткові функції, які відповідають потребам конкретного підприємства, такі як взаємодія з іншими системами або створення запитів до адміністрації.

Робочий простір адміністратора системи надає адміністратору всі необхідні інструменти для ефективного управління системою та забезпечення її правильної роботи.

2.3. Проєктування системи зберігання даних

Для зберігання інформації та маніпулювання цією інформацією використовується система зберігання даних MySQL.

MySQL - це безкоштовна система управління реляційними базами даних, яку розробила компанія "TcX" з метою покращення швидкодії обробки великих обсягів даних. У січні-лютому 2008 року Sun Microsystems придбала розробника бази даних MySQL за один мільярд доларів. Після того, як Sun Microsystems була поглинута компанією Oracle Corporation у 2009 році, MySQL перейшла під контроль Oracle.

Ця система керування базами даних (СКБД) з відкритим вихідним кодом була створена як альтернатива комерційним рішенням. MySQL спочатку була схожа на mSQL, але з часом вона значно розширилася і зараз вважається однією з найпоширеніших СКБД. Ця система активно використовується, передусім, для створення динамічних веб-сторінок завдяки відмінній підтримці різних мов програмування.

MySQL - це компактний багатопотоковий сервер баз даних, який славиться високою швидкістю, надійністю та простотою використання. Вона є вигідним вибором для малих і середніх додатків, адже її вихідний код може бути скомпільованим для різних платформ. Найбільш повноцінні можливості сервера реалізуються в UNIX-системах, де підтримується багатопотоковість і забезпечується підвищена продуктивність системи в цілому.

Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

Для забезпечення потрібного функціоналу будуть створені наступні таблиці для зберігання даних та відповідні зв’язки між ними, рис. 2.7.

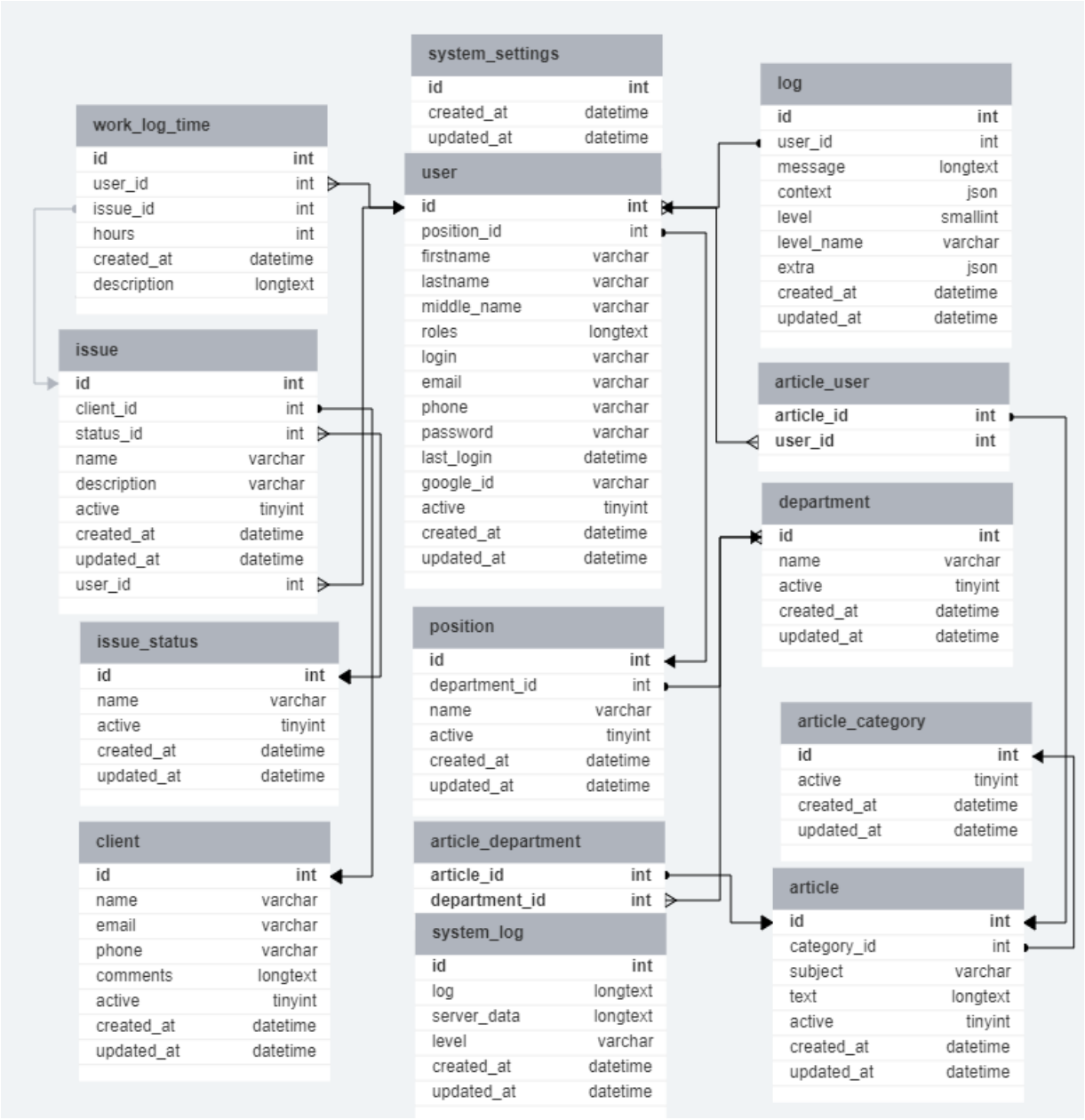


Рис. 2.7 Алгоритмічне забезпечення системи зберігання даних

Давайте розглянемо кожну з цих таблиць більш детально:

Таблиця article_category зберігає інформацію про категорії статей. Має колонки:

- id (унікальний ідентифікатор категорії);
- active (прапорець для позначення активності категорії);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця article зберігає інформацію про статті. Має колонки:

- id (унікальний ідентифікатор статті);
- category_id (зовнішній ключ, що посилається на id в таблиці article_category, для вказання категорії статті);
- subject (назва статті);
- text (текст статті);
- active (прапорець для позначення активності статті);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця client зберігає інформацію про клієнтів. Має колонки:

- id (унікальний ідентифікатор клієнта);
- name (ім'я клієнта);
- email (електронна пошта);
- phone (номер телефону);
- comments (додаткові коментарі);
- active (прапорець для позначення активності клієнта);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця department зберігає інформацію про відділи або підрозділи. Має колонки:

- id (унікальний ідентифікатор відділу);
- name (назва відділу);
- active (прапорець для позначення активності відділу);

- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця article_department визначає зв'язок між статтею (з таблиці article) і відділом (з таблиці department). Вона слугую для визначення чи має відділ доступ про перегляду статті. Має дві колонки: article_id і department_id, які є зовнішніми ключами, посилаючись на відповідні записи в таблицях article та department.

Таблиця issue_status зберігає інформацію про статуси задач. Має колонки:

- id (унікальний ідентифікатор статусу);
- name (назва статусу);
- active (прапорець для позначення активності статусу);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця position зберігає інформацію про посади. Має колонки:

- id (унікальний ідентифікатор посади);
- department_id (зовнішній ключ, що посилається на id в таблиці department, для вказання відділу, до якого відноситься посада);
- name (назва посади);
- active (прапорець для позначення активності посади);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця system_log зберігає журнал системних подій або логів. Має колонки:

- id (унікальний ідентифікатор запису логу);
- log (текстове поле для зберігання логів);
- server_data (текстове поле для зберігання додаткових даних сервера);
- level (рівень важливості логу);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця user зберігає інформацію про користувачів системи (працівників підприємства). Має колонки:

- id (унікальний ідентифікатор користувача);
- position_id (зовнішній ключ, що посилається на id в таблиці position, для вказання посади користувача);
- firstname (ім'я користувача);
- lastname (прізвище користувача);
- middle_name (по батькові користувача);
- roles (список ролей користувача);
- login (логін користувача);
- email (електронна пошта користувача);
- phone (номер телефону користувача);
- password (пароль користувача);
- last_login (дата та час останнього входу користувача);
- google_id (ідентифікатор Google, для зв’язування локального запису з записом Google);
- active (прапорець для позначення активності користувача);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця article_user визначає зв'язок між статтєю (з таблиці article) і користувачем (з таблиці user). Вона слугую для визначення чи має користувач доступ про перегляду статті. Має дві колонки: article_id і user_id, які є зовнішніми ключами, посилаючись на відповідні записи в таблицях article та user.

Таблиця issue зберігає інформацію про задачі. Має наступні колонки:

- id (унікальний ідентифікатор завдання);
- user_id (зовнішній ключ, який посилається на id в таблиці user, вказує на користувача, який відповідає за завдання);
- client_id (зовнішній ключ, який посилається на id в таблиці client, вказує на клієнта, який пов'язаний з завданням);

- status_id (зовнішній ключ, який посилається на id в таблиці issue_status, вказує на статус завдання);
- name (назва завдання);
- description (опис завдання);
- active (прапорець для позначення активності завдання);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця log зберігає журнал подій. Має наступні колонки:

- id (унікальний ідентифікатор запису логу);
- user_id (зовнішній ключ, який посилається на id в таблиці user, вказує на користувача, який створив лог);
- message (текстове поле для зберігання логів);
- context (JSON-поле для зберігання додаткових даних логу);
- level (рівень важливості логу);
- level_name (назва рівня логу);
- extra (JSON-поле для зберігання додаткової інформації);
- created_at (дата та час створення запису);
- updated_at (дата та час оновлення запису);

Таблиця work_log_time зберігає інформацію про робочий час користувачів над задачами. Має наступні колонки:

- id (унікальний ідентифікатор запису робочого часу);
- user_id (зовнішній ключ, який посилається на id в таблиці user, вказує на користувача, який працював над завданням);
- issue_id (зовнішній ключ, який посилається на id в таблиці issue, вказує на завдання, над яким працювали);
- hours (кількість годин робочого часу);
- created_at (дата та час створення запису);
- description (опис робочого процесу);

Висновки до розділу 2

В даному розділі проведено проєктування автоматизованої системи обліку робочого часу працівників на підприємстві, описано її ключові функції та інтерфейси для взаємодії з користувачами та адміністраторами. Крім цього, розглянуто кожну з трьох основних частин системи: робочий простір працівника підприємства, робочий простір адміністратора системи та система зберігання даних.

Для робочих просторів "працівник підприємства" та "адміністратор системи" були розроблені схеми, які описують основні функції, алгоритми роботи та елементи управління цими просторами. Це включає в себе можливості створення, редагування та видалення облікових записів користувачів, управління правами доступу, генерацію та обробку різних видів звітів, а також можливість створення, редагування та видалення клієнтів, компаній та задач для виконання. Додатково, описано важливі аспекти збереження та обробки даних.

Для системи зберігання даних була розроблена структура бази даних майбутньої системи. Були описані таблиці та поля цих таблиць, необхідні для збереження даних про користувачів, їх робочий час, права доступу, клієнтів, компанії, задачі та іншу інформацію. Також були визначені зв'язки між цими об'єктами, що визначає взаємозв'язки даних в системі.

Результати, отримані в цьому розділі, мають велике значення для подальшої розробки автоматизованої системи, оскільки вони надають чітку та структуровану основу для подальших етапів проєктування та розробки. Вони спрощують процес розробки та забезпечують високу ефективність проєкту в цілому.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Створення проєкту у системі контролю версій (BitBucket) та його налаштування

Першим чином я створив проєкт для моєї магістерської роботи у системи контролю версій BitBucket (https://bitbucket.org/).

Bitbucket - це веб-платформа для керування версіями та спільної роботи над програмними проєктами з використанням систем керування версіями (Version Control System, VCS) - зазвичай Git та Mercurial. Він надає засоби для зберігання, організації, відстеження змін і спільної розробки програмного коду.

Основні функції та переваги Bitbucket включають:

- надає приватні репозиторії, де команди та розробники можуть зберігати свій програмний код.
- підтримує роботу з системами керування версіями, такими як Git і Mercurial, що дозволяє відстежувати зміни в коді, створювати гілки для відокремленої розробки та об'єднувати їх.
- дозволяє кільком розробникам одночасно спільно працювати над одним проєктом. Він надає можливості коментування коду, створення та відстеження завдань, а також сприяє спільній комунікації серед команди.
- підтримує різноманітні типи проєктів, включаючи веб-розробку, розробку мобільних додатків, розробку вбудованих систем тощо.
- легко інтегрується з іншими інструментами розробки, такими як системи неперервної інтеграції/розробки (CI/CD), проєкти для відстеження задач, комунікаційні інструменти та багато інших.
- надає засоби для керування доступом до коду, встановлення прав доступу, створення аудиту коду та контролю за безпекою.
- пропонує послугу хостингу в хмарі, що робить його доступним із будь-якого місця, де є підключення до Інтернету.

Bitbucket корисний для розробників, команд розробників та організацій, які ведуть розробку програмного забезпечення та бажають ефективно керувати своїми проєктами, вести спільну роботу над кодом та забезпечувати безпеку та надійність проєктів. Інтерфейс системи Bitbucket зображено на рис. 3.1.

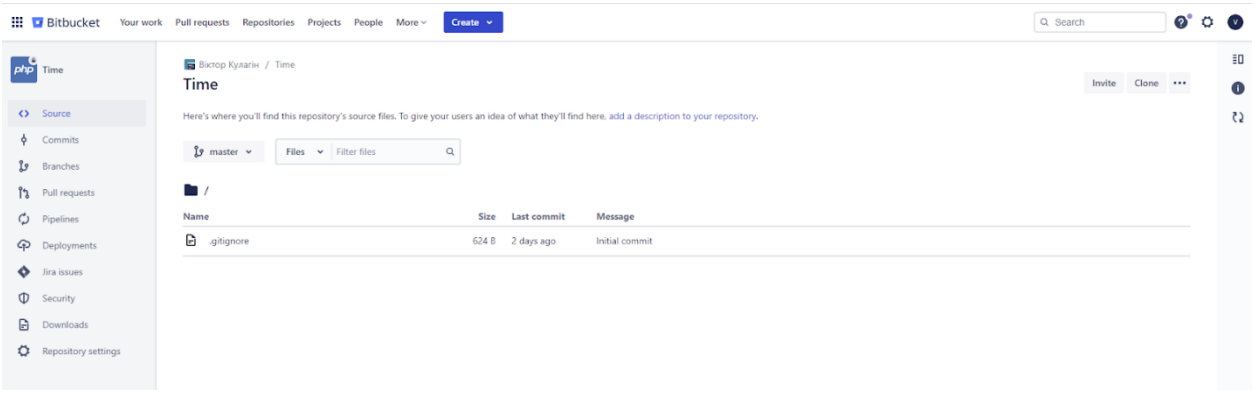


Рис. 3.1 Інтерфейс системи контролю версій (BitBucket)

3.2. Створення проєкту у середовищі розробки PHPStorm

Наступним кроком було створення проєкту у середовищі розробки PHP PhpStorm. Інтерфейс середовища розробки PHPStorm зображено на рис. 3.2.

PHPStorm - це інтегроване середовище розробки (Integrated Development Environment, IDE), створене компанією JetBrains спеціально для розробки веб-застосунків на мові програмування PHP.

Ця IDE надає інструменти та середовище для комфортної розробки, тестування і налагодження веб-проєктів, що використовують PHP, а також для роботи з іншими технологіями веб-розробки.

Основні характеристики і функції PHPStorm включають в себе:

- надає високоякісну підтримку PHP, включаючи автодоповнення коду, відстеження помилок, інтеграцію з іншими інструментами PHP. Крім того, вона підтримує інші мови програмування, такі як HTML, CSS, JavaScript, SQL і багато інших.
- має розширені можливості для редагування коду, включаючи підсвічування синтаксису, перетягування та опускання, автодоповнення, рефакторинг, вбудовані інструменти пошуку та заміни, історію редагування та багато інших.

- інтегрується з різними системами керування версіями, включаючи Git, Mercurial, Subversion та інші, що дозволяє розробникам легко відстежувати та керувати версіями свого коду.
- надає інструменти для відлагодження коду, включаючи можливість встановлення точок зупинки, відстежування змін змінних, виконання коду крок за кроком та інші можливості для відлагодження.
- підтримує роботу з різними системами управління базами даних, включаючи MySQL, PostgreSQL, SQLite та інші. Це дозволяє розробникам працювати з базами даних, виконувати запити SQL та аналізувати дані.
- легко інтегрується з іншими інструментами розробки, такими як системи неперервної інтеграції/розробки (CI/CD), системи керування завданнями та інші.
- підтримує багато відомих фреймворків для розробки веб-застосунків, такі як Laravel, Symfony, Yii, Zend та багато інших.

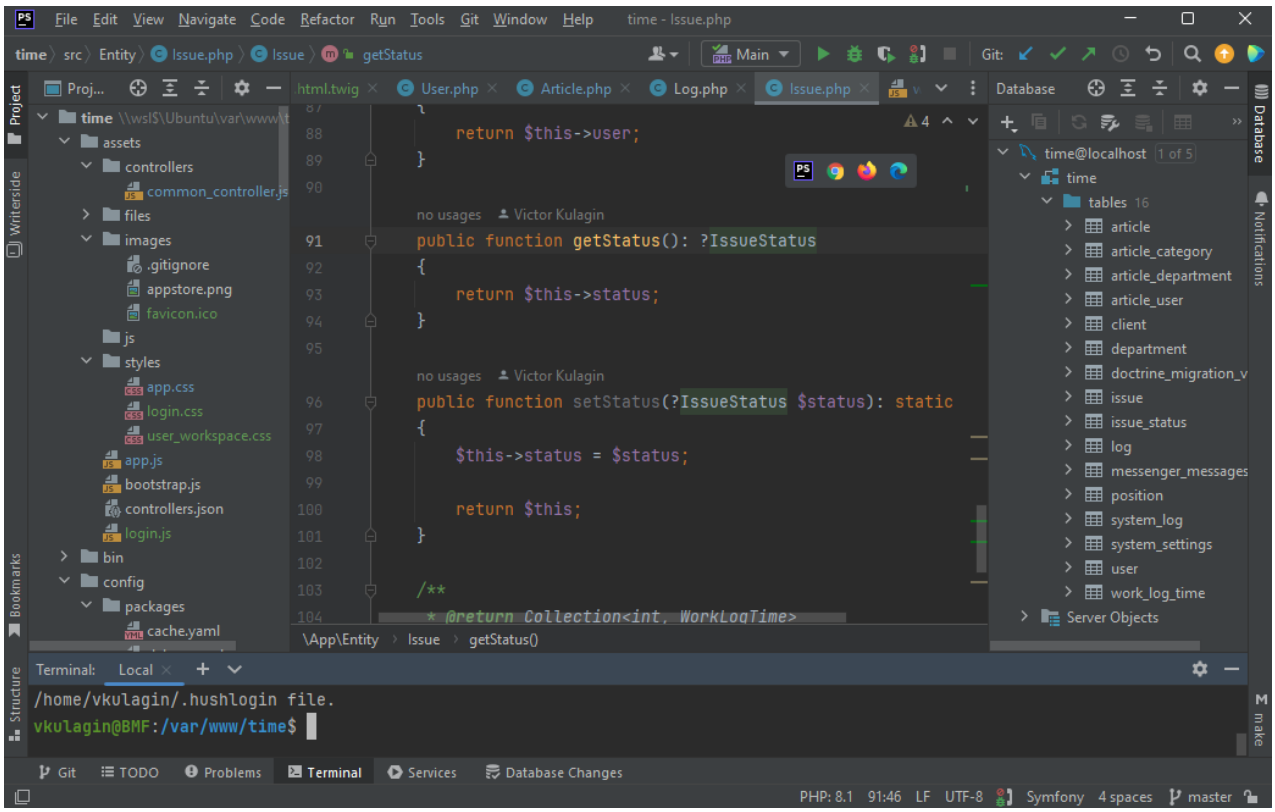


Рис. 3.2 Інтерфейс середовища розробки PHPStorm

PHPStorm є потужним інструментом для розробників PHP, які розробляють веб-додатки та бажають поліпшити продуктивність та якість свого коду. Вона надає засоби для розробки, відлагодження та спільної роботи над PHP-проєктами.

3.3. Створення оточення розробки у Docker

Наступним кроком я налаштував Docker. Docker - це платформа для розробки, доставки та запуску застосунків у вмісті, відомому як контейнери. Контейнери є стандартизованими вмістом, яке включає в себе програмний код, залежності та конфігурацію, що дозволяє застосунку працювати в будь-якому середовищі, яке підтримує Docker. Він надає незалежність застосунків від інфраструктури, роблячи їх переносними та легко масштабованими.

Основні компоненти Docker включають в себе Docker Engine, Docker Hub та Docker Compose.

Docker Engine - це рішення для створення та керування контейнерами. Воно включає в себе сервер Docker, який керує контейнерами, та клієнт Docker, який надає інтерфейс користувача для взаємодії з Docker Engine.

Docker Hub - це хмарне сховище для обміну контейнерами та шарів контейнерів. Docker Hub містить багато готових образів контейнерів, які можна використовувати як основу для ваших власних контейнерів.

Docker Compose - це інструмент для опису та запуску складних багоконтейнерних застосунків. За допомогою Docker Compose можна визначити, які контейнери повинні бути запущені разом, і як вони мають взаємодіяти. Інтерфейс програми Docker зображено на рис. 3.3.

Основні переваги використання Docker включають в себе:

- Переносність - контейнери можуть працювати на будь-якому середовищі, де встановлений Docker, незалежно від операційної системи чи хмарної платформи.
- Ізоляція - контейнери ізолюють застосунки та їх залежності, забезпечуючи безпеку та надійність.

- Масштабованість – надає можливість легко масштабувати контейнерні застосунки, додаючи або видаляючи контейнери за необхідності.
- Ефективність розробки - Docker спрощує розробку та тестування застосунків, оскільки можна створювати ізольовані контейнери для різних компонентів.
- Швидкість доставки - контейнери дозволяють швидко розгортати застосунки в будь-якому середовищі.

Docker став популярним інструментом у сферах розробки програмного забезпечення, розгортання, безпеки та інфраструктури через свою простоту та ефективність у керуванні контейнерами.

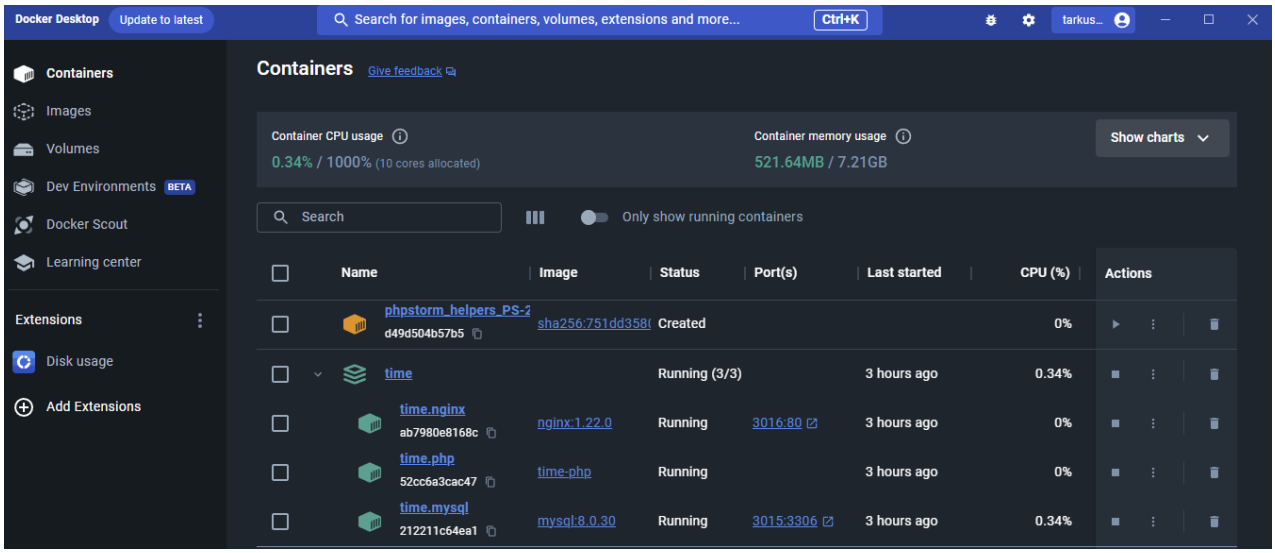


Рис. 3.3 Інтерфейс програми Docker

Для розробки мені необхідно налаштувати три Docker контейнери (Containers), а саме PHP, NGINX та MySQL. PHP відповідає за обробку PHP коду, NGINX являє собою веб-сервер, а MySQL – це база даних.

Конфігурування Docker відбувається у файлі docker-compose.yml. В файловій структурі проєкту я створив окремий каталог, в якому зібрані файли конфігурацій Docker та окремо PHP, NGINX та MySQL. Структура цього каталогу зображена на рис. 3.4.

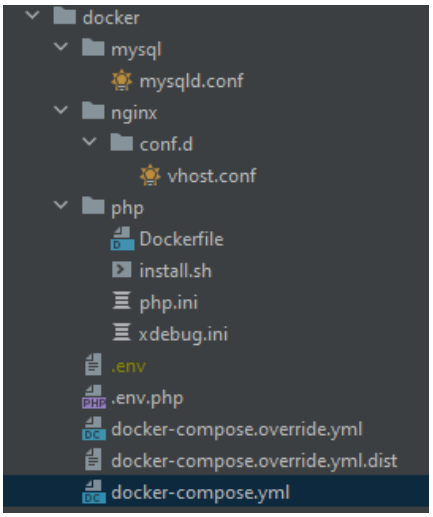


Рис. 3.4 Структура каталогу з конфігураційними файлами Docker

Файл mysqld.conf відповідає за конфігурацію MySQL, vhost.conf за конфігурацію NGINX, а php.ini містить в собі налаштування PHP. Файли .env.php та .env містять в собі зміні оточення, необхідні для роботи Docker та Symfony.

Файли docker-compose.yml та Dockerfile описують процес установки Docker контейнера та образів. На рис. 3.5 та рис. 3.6 наведено код цих файлів.

Файл docker-compose.yml встановлює і налаштовує три контейнери: PHP, Nginx та MySQL, і визначає їхні залежності та параметри.

Основні частини коду:

version: '3.9': Версія синтаксису Docker Compose. У цьому випадку використовується версія 3.9.

services: Цей розділ визначає набір сервісів (контейнерів), які потрібно запустити.

php: Сервіс для PHP контейнера.

env_file: Вказує файл із змінними середовища, які будуть передані у контейнер.

container_name: Ім'я PHP контейнера.

build: Конфігурація для побудови Docker образу із вказаного Dockerfile (див. Рис. 3.6).

volumes: Монтує різні томи і файли у контейнер.

```
version: '3.9'
services:
  php:
    env_file:
      - .env.php
    container_name: ${COMPOSE_PROJECT_NAME}.php
    build:
      dockerfile: php/Dockerfile
    volumes:
      - ..:/var/www/app
      - ./php/php.ini:/usr/local/etc/php/conf.d/php.ini
      - ./php/xdebug.ini:/usr/local/etc/php/conf.d/xdebug.ini
      - ./php/install.sh:/var/www/app/bin/install.sh
    depends_on:
      - mysql
  nginx:
    container_name: ${COMPOSE_PROJECT_NAME}.nginx
    image: nginx:1.22.0
    volumes:
      - ../../..:/var/www/app
      - ./nginx/conf.d/vhost.conf:/etc/nginx/conf.d/default.conf
    ports:
      - ${DOCKER_PORT_NGINX}:80
    depends_on:
      - php
  mysql:
    container_name: ${COMPOSE_PROJECT_NAME}.mysql
    image: mysql:8.0.30
    ports:
      - ${DOCKER_PORT_MYSQL}:3306
    volumes:
      - type: bind
        source: ./mysql/mysqld.conf
        target: /etc/mysql/mysql.conf.d/mysqld.cnf
        read_only: true
      - type: volume
        source: mysql
        target: /var/lib/mysql
        volume:
          nocopy: true
    environment:
      MYSQL_ALLOW_EMPTY_PASSWORD: 1
      MYSQL_DATABASE: time
      MYSQL_USER: time
      MYSQL_PASSWORD: time
volumes:
  mysql: {}
```

Рис. 3.5 Файл docker-compose.yml

depends_on: Вказує, від яких інших сервісів цей сервіс залежить (у цьому випадку від MySQL).

nginx: Сервіс для Nginx контейнера.

container_name: Ім'я Nginx контейнера.

image: Використовує готовий образ Nginx з Docker Hub.

volumes: Монтує файли та теки у контейнер.

ports: Відображає порти контейнера на зовнішні порти.

mysql: Сервіс для MySQL контейнера.

container_name: Ім'я MySQL контейнера.

image: Використовує готовий образ MySQL з Docker Hub.

ports: Відображає порти контейнера на зовнішні порти.

volumes: Монтує файли та теки у контейнер.

environment: Задає змінні середовища для налаштування MySQL.

volumes: Цей розділ визначає Docker volumes, які використовуються в сервісі MySQL.

Dockerfile визначає, як створити Docker образ для сервера PHP з додатковими налаштуваннями та встановленими PHP розширеннями.

Основні кроки та дії в моєму Dockerfile:

Використовується базовий образ php:8.1.23-fpm з Docker Hub.

Виконується оновлення пакетів та встановлення додаткових пакетів із списку.

Встановлюються PHP розширення та інші компоненти:

xdebug: Встановлюється через PECL і активується як розширення PHP.

intl, gd, mysqli, opcache, pdo_mysql, zip: Встановлюються PHP розширення через docker-php-ext-install.

Встановлюється Node.js і npm:

Встановлюється nvm (Node Version Manager).

Встановлюється Node.js (версія зазначена в змінній NODE_VERSION).

Налаштовується середовище для роботи з Node.js і npm.

Встановлюється Composer:

Завантажується Composer з офіційного репозиторію та копіюється в /usr/bin.

Надається право на виконання для Composer.

Встановлюється робочий каталог для PHP.

Відкривається порт 9000 для з'єднань з PHP-FPM.

Використовується команда CMD для запуску PHP-FPM та очікування доступності MySQL перед виконанням ./bin/install.sh.

```
FROM php:8.1.23-fpm

RUN apt-get update && \
    apt-get install -y --no-install-recommends acl default-mysql-client wait-
for-it less libicu-dev libpng-dev libzip-dev unzip

# Install PHP extensions
RUN pecl install xdebug-3.1.5
RUN docker-php-ext-enable xdebug
RUN docker-php-ext-install intl
RUN docker-php-ext-install gd
RUN docker-php-ext-install mysqli
RUN docker-php-ext-install opcache
RUN docker-php-ext-install pdo_mysql
RUN docker-php-ext-install zip

ENV NODE_VERSION=18.17.1
RUN apt install -y curl
RUN curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh |
bash
ENV NVM_DIR=/root/.nvm
RUN . "$NVM_DIR/nvm.sh" && nvm install ${NODE_VERSION}
RUN . "$NVM_DIR/nvm.sh" && nvm use v${NODE_VERSION}
RUN . "$NVM_DIR/nvm.sh" && nvm alias default v${NODE_VERSION}
ENV PATH="/root/.nvm/versions/node/v${NODE_VERSION}/bin/:${PATH}"
RUN node --version
RUN npm --version

# Install composer
RUN curl -gL
https://github.com/composer/composer/releases/download/2.4.0/composer.phar >
/usr/bin/composer
RUN chmod +x /usr/bin/composer

WORKDIR /var/www/app

EXPOSE 9000
CMD wait-for-it mysql:3306 -- bash ./bin/install.sh && php-fpm
```

Рис. 3.6 Файл Dockerfile

Цей Dockerfile дозволить створити Docker образ із налаштованим середовищем для веб-застосунку, включаючи PHP, Node.js, Composer та необхідні розширення.

Для того, щоб зібрати за запустити Docker контейнери потрібно у консолі перейти в каталог проєкту та виконати команду docker-compose up –build. Після завершення виконання команди Docker контейнери будуть доступні для подальшого використання.

3.4. Встановлення Symfony

Наступним кроком було встановлення Symfony Framework. Один із найпоширеніших способів встановлення Symfony – це використовувати

Composer, популярний пакетний менеджер PHP, який я завчасно встановив у Docker контейнер.

Для встановлення Symfony необхідно зайти в консоль PHP контейнера, перейти у каталог, де буде зберігатися проєкт та виконати команду composer create-project symfony/skeleton time, де time – це назва мого проєкту. Для того, щоб Symfony містила всі необхідні компоненти для веб-розробки необхідно виконати ще наступні команди: cd time, для переходу в каталог проєкту та composer require webapp для встановлення необхідних компонентів, використовуючи composer. Результатом буде вітальна сторінка Symfony, рис. 3.7.

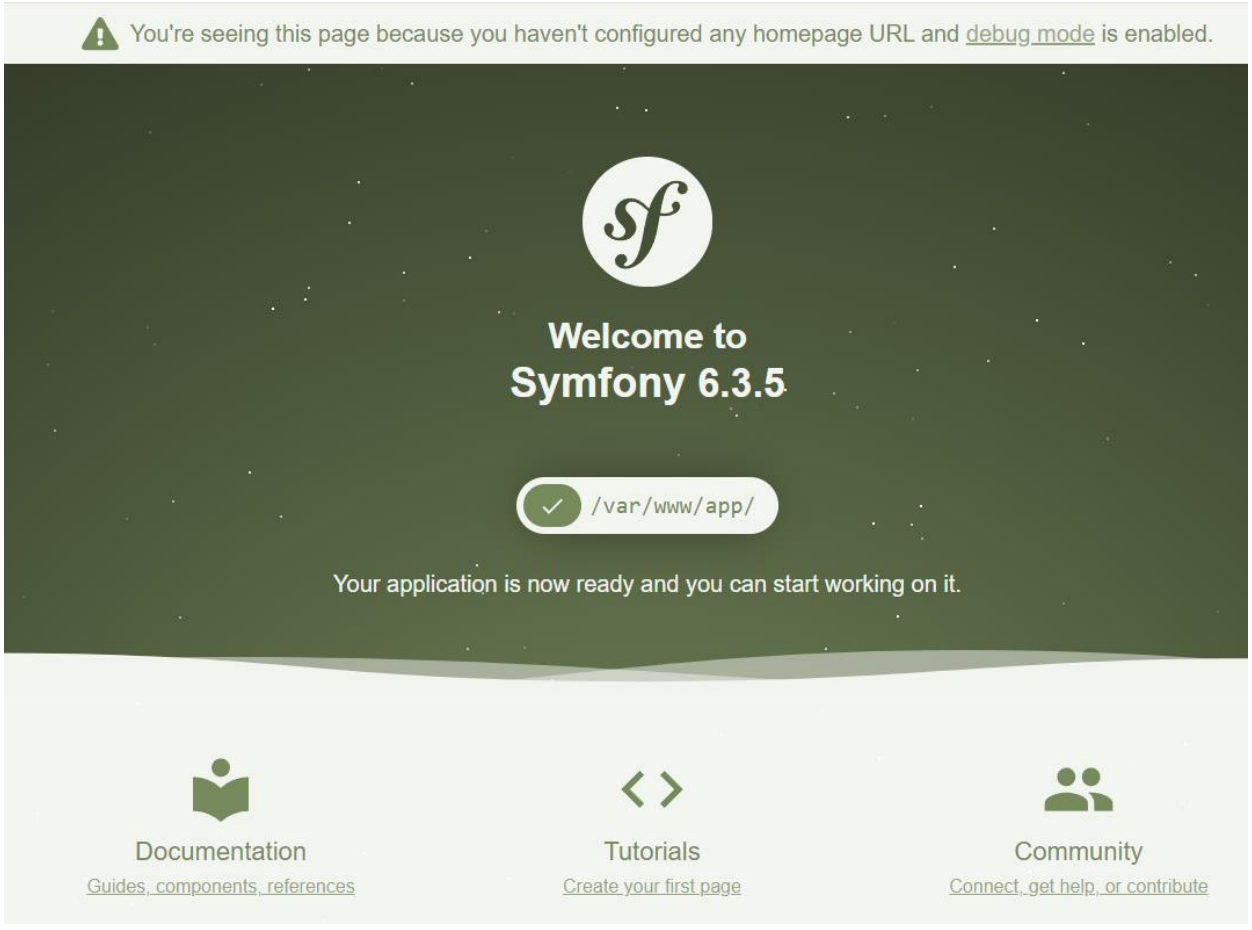


Рис. 3.7 Вітальна сторінка Symfony

3.5. Встановлення та налаштування WebPack

Для роботи з ресурсами, такими як CSS, JavaScript та зображеннями, я встановив WebPack.

Webpack - це потужний інструмент для збирання та управління ресурсами (такими як JavaScript, CSS, зображення) у веб-додатках. Він зазвичай використовується в сучасному веб-розробці для оптимізації та пакування ресурсів з метою зменшення завантаження сторінки та підвищення продуктивності. Webpack дозволяє об'єднувати JavaScript файли, CSS файли, зображення та інші ресурси в одну або декілька компільованих файлів. Це допомагає зменшити кількість запитів, які браузер повинен робити для завантаження сторінки. Крім цього він дозволяє використовувати модульний підхід до розробки, де можна організувати свій код на логічні модулі і імпортувати їх один в інший. Це полегшує управління залежностями та структурою проєкту. Webpack може автоматично оптимізувати та мінімізувати код, щоб зменшити його розмір і підвищити продуктивність завантаження.

Webpack використовує конфігураційний файл (зазвичай webpack.config.js), в якому визначаються правила для обробки та збору ресурсів. Розробники можуть налаштовувати цей файл відповідно до потреб свого проєкту.

Загальна ідея полягає в тому, щоб завантажувати лише ті ресурси, які потрібні на кожній сторінці та зменшувати час завантаження сторінки.

Webpack є незамінним інструментом у сучасному фронтенд-розробнику для забезпечення оптимальної продуктивності та структури проєкту.

Для встановлення Webpack у Symfony, необхідно налаштувати Webpack разом із Symfony Flex та відповідними плагінами для інтеграції з Symfony.

Для цього я виконав наступні кроки:

Встановив Symfony Flex за допомогою Composer.

composer require symfony/flex

Встановив Symfony Webpack Encore - бібліотеку, яка полегшує налаштування Webpack для Symfony проєктів.

composer require symfony/webpack-encore-bundle

Для подальшої роботи з Webpack я налаштував webpack.config.js, код якого зображено на рис. 3.8.

```
const Encore : Encore | {...}  = require('@symfony/webpack-encore');

if (!Encore.isRuntimeEnvironmentConfigured()) {
    Encore.configureRuntimeEnvironment( environment: process.env.NODE_ENV || 'dev');
}

Encore
    .setOutputPath('public/build/')
    .setPublicPath('/build')
    .addEntry( name: 'app', src: './assets/app.js')
    .enableStimulusBridge( controllerJsonPath: './assets/controllers.json')
    .splitEntryChunks()
    .enableSingleRuntimeChunk()
    .cleanupOutputBeforeBuild()
    .enableBuildNotifications()
    .enableSourceMaps(!Encore.isProduction())
    .enableVersioning(Encore.isProduction())
    .configureBabelPresetEnv( callback: (config) : void  => {
        config.useBuiltIns = 'usage';
        config.corejs = '3.23';
    })
    .enableSassLoader()

    .autoProvideVariables( variables: {
        $: 'jquery',
        jQuery: 'jquery',
        'window.jQuery': 'jquery',
        bootbox: 'bootbox',
    })
    .autoProvidejQuery()
;

module.exports = Encore.getWebpackConfig();
```

Рис. 3.8 Файл конфігурації Webpack webpack.config.js

Наступним кроком я встановив утиліту yarn, яка використовується для збору ресурсів:

yarn install

Тепер я можу використовувати наступні команду Symfony Console для збору ресурсів:

yarn dev – збирає ресурси для dev оточення, залишаючи код файлів зручним для читання, що полегшує відладку коду.

yarn build - збирає ресурси для prod оточення, мінімізую та оптимізує їх, що робить їх не зручними для читання, але покращую швидкість завантаження сторінок.

yarn watch – корисний режим роботи, коли файли ресурсів оброблюються в режимі реального часу, що робить розробку набагато комфортнішою.

Після виконання команди yarn build Webpack обробить файли ресурсів та підготую їх для подальшого використання.

3.6. Створення сутностей (Entity)

Наступним моїм кроком було налаштування сутностей (Models) та створення відповідної структури бази даних MySQL, за схемою (див. Рис. 2.7), яку було розроблено раніше.

Мною були створені всі необхідні сутності та забезпечені всі необхідні зв’язки між ними. Список цих сутностей зображено на рис. 3.9.

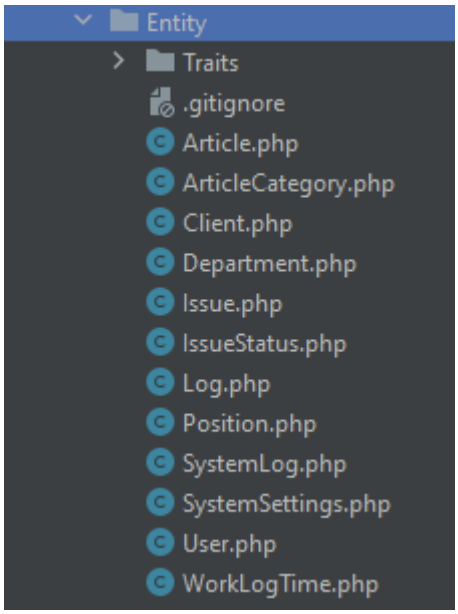


Рис. 3.9 Створені сутності

При створені сутностей я також використав однин з інструментів мови програмування PHP, який називається Traits.

Traits - це механізм множинної спадковості (multiple inheritance), який дозволяє використовувати функціональність одного класу в іншому класі без створення справжнього ланцюжка спадкування. Вони дозволяють включити методи і властивості одного класу в інший, що дозволяє поділити та перевикористовувати код між класами без створення складних ієрархій класів.

Наприклад, я створив трейт IdTrait, який звільнить він необхідності описувати поле ідентифікатора Id та його сетера (функція setId) та гетера

(функція getId) у всіх сутностях проєкту. Код цього трейту відображено на рис. 3.10.

```php
trait IdTrait
{
    #[ORM\Id]
    #[ORM\Column(name: 'id', type: Types::INTEGER, nullable: false)]
    #[ORM\GeneratedValue(strategy: 'AUTO')]
    protected ?int $id;

    no usages  Victor Kulagin
    public function getId(): int
    {
        return $this->id;
    }

    no usages  Victor Kulagin
    public function setId(?int $id): static
    {
        $this->id = $id;

        return $this;
    }
}
```

Рис. 3.10 Код IdTrait

Створення трейтів та подальше їх використання дозволяє не лише уникнути повторення одного і того коду у різних місцях програми, але і забезпечую гарантування того, що при зміні логіки у цьому трейті, вона буде змінена всюди, де використовується трейт. Крім IdTrait було створено ще декілька трейтів для інших, повторюваних у багатьох моделях, полів.

Приклад використання трейтів IdTrait, ActiveTrait та CreateUpdateTrait у сутності User показано на рис. 3.11.

```php
#[ORM\Entity(repositoryClass: UserRepository::class)]
#[UniqueEntity(fields: ['login'], message: 'There is already an account with this login')]
#[ORM\HasLifecycleCallbacks]
class User implements UserInterface, PasswordAuthenticatedUserInterface{
    use IdTrait;
    use ActiveTrait;
    use CreatedUpdatedTrait;

    2 usages
    #[ORM\Column(length: 255)]
    private ?string $firstname = null;

    2 usages
    #[ORM\Column(length: 255)]
    private ?string $lastname = null;

    2 usages
    #[ORM\Column(length: 255, nullable: true)]
    private ?string $middleName = null;

    2 usages
    #[ORM\Column(type: Types::JSON)]
    private array $roles = [];
```

Рис. 3.11 Використання IdTrait у класі сутності User

3.7. Система авторизації

Після цього приступив до реалізації системи авторизації. Я вирішив розробити авторизацію двома способами – вхід за допомогою логіну і паролю з локальної бази даних та за допомогою облікового запису Google.

Для налаштування входу через обліковий запис Google в Symfony, необхідно використовувати бібліотеку або пакет для автентифікації з OAuth2. В Symfony, однією з популярних бібліотек для роботи з OAuth2 є "HWIOAuthBundle".

Для того щоб її встановити я виконав наступну команду:

composer require hwi/oauth-bundle php-http/guzzle6-adapter

Я провів налаштування додатку в Google Developer Console. Цей процес включав створення нового проєкту та конфігурування OAuth-клієнта.

Спершу я створив новий проєкт у Google Developer Console. Проєкт - це інтерфейс для управління різними налаштуваннями та послугами, пов'язаними

з програмою або додатком. Інтерфейс Google Developer Console зображено на рис. 3.12.

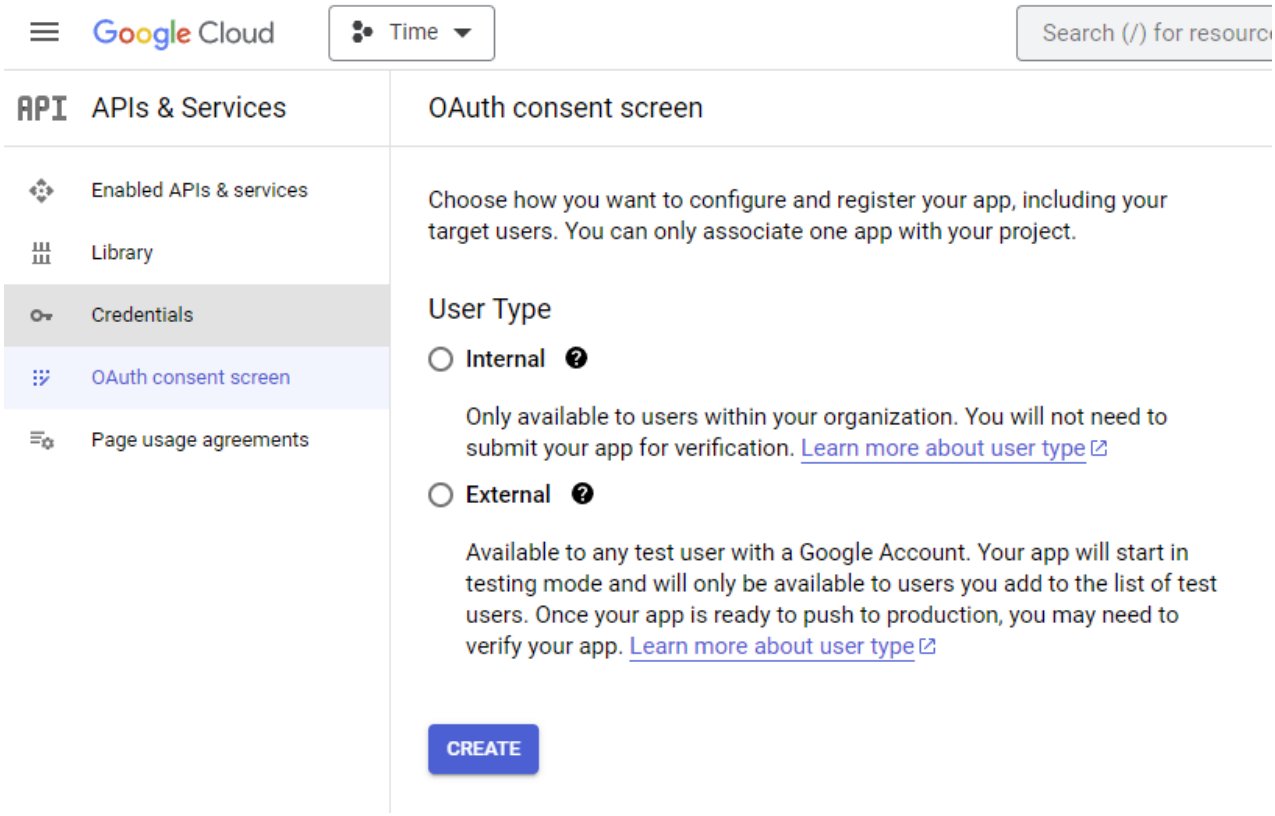


Рис. 3.12 Інтерфейс Google Developer Console

Далі я створив OAuth-клієнта для мого проєкту. OAuth - це протокол автентифікації, який дозволяє отримувати доступ до облікових даних користувача в інших системах або службах. OAuth-клієнт - це інструмент, який дозволяє взаємодіяти з іншими веб-сервісами чи додатками.

Після створення OAuth-клієнта мені було потрібно налаштувати його параметри. Це включало в себе визначення списку дозволів, які мій додаток запитуватиме у користувачів, а також надання URL-адрес для перенаправлення після успішної автентифікації.

Однією з важливих частин налаштування OAuth-клієнта є забезпечення безпеки. Це включає в себе збереження конфіденційних ключів та секретів безпеки, які використовуються для автентифікації.

Після налаштування OAuth-клієнта я використовував інструменти для відладки та тестування, щоб переконатися, що автентифікація працює належним чином та що система може взаємодіяти з Google автентифікацією.

Отримані client_id і client_secret я використав у налаштуваннях для автентифікації з Google, як показано на рис. 3.13.

```
hwi_oauth:
    firewall_name: main
    resource_owners:
        google:
            type: google
            client_id: %google_client_id%
            client_secret: %google_client_secret%
parameters:
    google_client_id: your_google_client_id
    google_client_secret: your_google_client_secret
```

Рис. 3.13 Налаштування HWIOAuthBundle

Наступним кроком було налаштування маршрутизації HWIOAuthBundle в app/config/routes/ hwi_oauth_routing.yaml.

Після налаштування маршрутизації необхідно змінити конфігурацію безпеки в app/config/packages/security.yml, додавши нову секцію oauth, яка вказую, як буде опрацьовуватись функціональність, пов’язана з авторизацією через Google, як показано на рис. 3.14.

```
    main:
        anonymous: ~
        oauth:
            resource_owners:
                google: "/login/check-google"
            login_path: /login
            use_forward: false
            failure_path: /login
            oauth_user_provider:
                service: hwi_oauth.user.provider.fosub_bridge
            success_handler:
                hwi_oauth.authentication.success_handler
```

Рис. 3.14 Налаштування app/config/packages/security.yml

Налаштував FOSUserBundle, а також зв'язав його з HWIOAuthBundle, щоб створити користувача в базі даних при вході через Google, якщо його ще не існувало в системі.

У шаблоні додав посилання на вхід через Google:

Login with Google

У разі успішної автентифікації користувача він буде перенаправлений у робочий простір працівника підприємства. Додатково провожу перевірку, що обліковий запис Google належить підприємству.

Зовнішній вигляд сторінки авторизації в системі зображено на рис. 3.15.

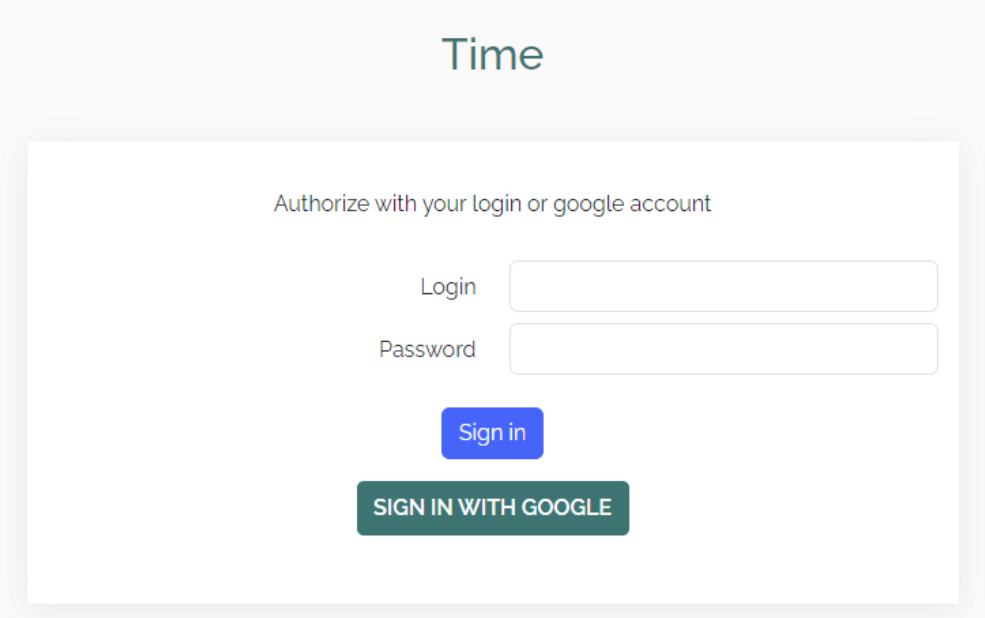


Рис. 3.15 Сторінка авторизації в системі використовуючи локальні облікові дані або за допомогою облікового запису Google

В разі входу через локальний обліковий запис використовується контролер SecurityController, код якого зображено на рис. 3.16. Цей контролер використовую стандартну для Symfony схему авторизації, описану у AuthenticationUtils та FormLoginAuthenticator. Вони забезпечують корисний функціонал, як наприклад валідацію форми входу та запам’ятовування останнього введеного логіну. При необхідності, цей функціонал можна замінити своїм, переписавши відповідні класи. Крім корисного функціоналу

FormLoginAuthenticator забезпечую головну функціональність, а саме перевірку введених облікових записів, співставлення їх з наявними записами у базі даних та авторизацію користувача у системі.

```
class SecurityController extends AbstractController
{
   public function __construct(
       private readonly FormLoginAuthenticator   $authenticator,
   ) {
}

   #[Route('/user/login', name: 'app_login')]
   public function login(
       AuthenticationUtils $authenticationUtils
   ): Response
   {
       if ($this->getUser()) {
           return $this->redirectToRoute('homepage');
       }

       $error = $authenticationUtils-
>getLastAuthenticationError();

       $lastUsername = $authenticationUtils->getLastUsername();

       return $this->render('security/login.html.twig', [
           'last_username' => $lastUsername,
           'error' => $error,
       ]);
   }

   #[Route('/logout', name: 'app_logout', methods: ['GET'])]
   public function logout(): mixed
   {
       throw new Exception('Don\'t forget to activate logout in
security.yaml');
   }
```

Рис. 3.16 Контролер SecurityController

3.8. Контролери (Controllers) та основні сторінки

Наступним кроком я створив все необхідні контролери. Кожен контролер відповідає за певну функціональність системи. Розглянемо детальніше кожен контролер.

UserWorkspaceController: Відповідає за дії, пов'язані з робочим простором працівника підприємства. Код контролера зображено на рис. 3.17.

create(): відповідає за можливість користувачів створювати нові записи про робочий час працівників підприємства.

show(): відповідає за перегляд деталей свого робочого часу, включаючи історію робочих записів.

edit(): редагування існуючих записи про робочий час, якщо це необхідно.

ReportController. Відповідає за дії, пов'язані з звітами. Такі як створення, перегляд, друк звітів.

create(): надає користувачам можливість створювати нові звіти. Наприклад, користувачі можуть створювати щоденні, тижневі або місячні звіти.

show(): слугує для перегляду звітів та їх вмісту.

print(): забезпечує можливість конвертувати звіти у формати XLS і PDF та друкувати їх.

SettingsController. Відповідає за дії, пов'язані з налаштуваннями робочого простору працівника підприємства, інтерфейс користувача.

index(): дана дія відповідає за відображення налаштувань робочого простору працівників підприємства та інтерфейсу користувача. Користувачі можуть налаштовувати різні параметри, які впливають на їхню робочу область.

index(): збереження налаштувань робочого простору працівників підприємства та інтерфейсу користувача.

ProfileController. Відповідає за дії, пов'язані з профілем користувача перегляд та редагування облікових даних користувача.

view(): використовується для перегляду свого користувацького профілю та особистої інформації.

edit(): ця дія надає можливість користувачам редагувати та зберігати свої особисті дані та профіль.

ArticleController. Цей контролер відповідає за довідковий підприємства. Він дозволяє користувачам переглядати та отримувати доступ до різних інформаційних розділів, які можуть бути корисними для роботи на підприємстві.

index(): відображає статті, до яких у користувача наявний доступ.

view(): відображає конкретну статтю, за умови, що у користувача наявний доступ до неї.

AdminRequestController. Відповідає за дії, пов'язані з запитами до адміністрації, створення, перегляд вже створених.

create(): дозволяє користувачам створювати нові запити до адміністрації підприємства.

index(): Користувачі можуть переглядати вже створені запити та їх статус.

Така структура контролерів дозволяє чітко розділити функціональність та полегшити розробку та підтримку проєкту. Кожен контролер має свою відповідальність, що сприяє читабельності та модульності коду.

```php
class UserWorkspaceController extends AbstractController
{

    /**
     * @param EntityManagerInterface $entityManager
     * @param UserInterface $user
     * @param null $date
     * @return Response
     */
    no usages  new *
    #[Route('/user/{date}', name: 'user_main')]
    public function userMainAction(
        EntityManagerInterface $entityManager,
        UserInterface $user,
        $date = null,
    ): Response
    {
        $workTimeLogs = $entityManager->getRepository(WorkLogTime::class)->findBy(
            [
                'date' => $date,
                'user' => $user,
                'active' => true,
            ],
        );

        return $this->render( view: '/base.html.twig', [
            'date' => $date,
            '$workTimeLogs' => $workTimeLogs,
        ]);
    }
```

Рис. 3.17 Контролер UserWorkspaceController

3.9. Шаблони (View)

Для відображення інформації та надання можливостей користувачу взаємодіяти з системою мною були розроблені всі необхідні шаблони (View). Як вже було описано вище, шаблони в Symfony в основному пишуться за допомогою шаблонізатора Twig.

Для того, щоб налаштувати зовнішній вигляд елементів я використовував мову декорування CSS. Крім того, я застосував фреймворк Bootstrap. Bootstrap - це відкритий фреймворк для розробки веб-сайтів і веб-додатків. Він надає набір готових стилів, компонентів, шаблонів та інших інструментів, що полегшують створення сучасних та адаптивних веб-інтерфейсів. Bootstrap був розроблений командою Twitter і став дуже популярним завдяки своїй простоті використання і широкому спектру можливостей.

Для забезпечення функціювання елементів користувацького інтерфейсу використовувалася мова програмування JavaScript та різноманітні JavaScript бібліотеки, як наприклад JQuery. Також, для того щоб додати інтерактивність веб-сторінкам, я використав JavaScript фреймворк Stimulus.

Stimulus - це JavaScript-фреймворк для розробки інтерактивних веб-інтерфейсів. Він був розроблений для додавання дійсної динаміки до веб-сторінок та додавання легковагової логіки до клієнтських додатків. Основна ідея Stimulus полягає в тому, щоб зосередитись на збереженні чистоти HTML, CSS та JavaScript, дозволяючи розробникам створювати ефективні інтерактивні компоненти на стороні клієнта.

```
import {Controller} from '@hotwired/stimulus';

no usages  Victor Kulagin *
export default class extends Controller {
    no usages  Victor Kulagin *
    connect() : void  {
        $(document).ready(function ($) : void  {
            $('.photo-box').on('click', function () : void  {
                $('#cdk-overlay-0').toggle();
            })
        });
    }
}
```

Рис. 3.18 Приклад Stimulus контролера

3.10. Панель адміністратора та її основні функції

Наступним етапом було створення адміністративної панелі управління.

При розробці адміністративної панелі використовувався бандл EasyAdmin.

EasyAdmin - це бібліотека для Symfony, яка надає готовий адміністративний інтерфейс для управління базою даних та адміністрування контенту в веб-додатках. Це спрощує процес створення адмін-панелі і дозволяє швидко налаштовувати та керувати моделями даних без необхідності писати багато коду вручну.

Основні можливості EasyAdmin включають:

- Готовий інтерфейс: EasyAdmin надає готовий інтерфейс адмін-панелі з можливістю додавання, редагування, видалення та перегляду даних.
- Конфігурування через YAML: налаштування адмін-панелі здійснюється за допомогою YAML-файлів, що робить процес налаштування простим та зрозумілим.
- Гнучка конфігурація полів: легко вказувати, які поля повинні бути відображені в адмін-панелі та як вони повинні бути представлені.
- Підтримка багатьох типів полів: EasyAdmin підтримує різні типи полів, такі як текстові поля, числа, дати, файли та інші.
- Фільтрація та пошук: користувачі можуть швидко фільтрувати та шукати дані за допомогою вбудованих функцій.
- Сортування та пагінація: можливість сортувати дані та розподіляти їх по сторінках для зручності навігації.
- Розширення за допомогою подій та власних контролерів: якщо потрібно реалізувати додаткову функціональність, EasyAdmin надає можливість використовувати дії та власні контролери.

Загалом, EasyAdmin допомагає значно прискорити розробку адмін-панелі для веб-додатків, спрощуючи багато звичайних завдань та дозволяючи фокусуватися на бізнес-логіці.

EasyAdmin у Symfony встановлюється за допомогою Composer, командою composer require easycorp/easyadmin-bundle.

Налаштувавши маршрутизацію та основні параметри EasyAdmin я приступив до генерації контролерів простору адміністратора системи.

EasyAdminBundle дозволяє генерувати контролери для адміністрування моделями даних. Щоб це зробити треба використати наступну команду:

php bin/console make:admin:crud

Після відповіді на ряд запитань генератора буде згенерований контролер, який треба доповнити необхідним функціоналом.

Мною були згенеровані контролери, показані на рис. 3.19.

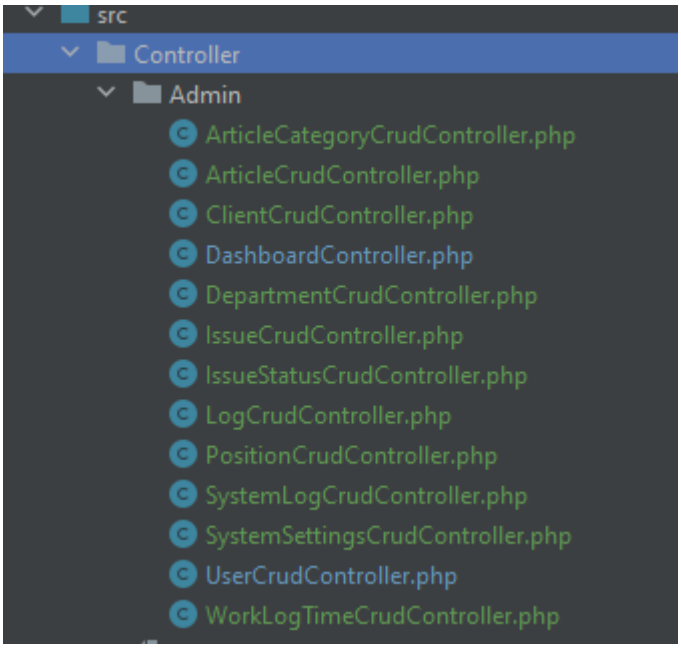


Рис. 3.19 Контролери робочого простору адміністратора системи

Наступним кроком було написання коду для кожного контролера, яким я описав які поля повинні відображатися при редагуванні сутності, які при перегляді конкретної сутності або списку всіх сутностей (див. Додаток А).

Також необхідно описати правила для збереження сутностей у базу даних. Наприклад, при створені або редагуванні користувача, необхідно задати йому пароль, який повинен бути захешований, рис 3.20.

```php
public function persistEntity(EntityManagerInterface $entityManager, $entityInstance): void
{
    $user = $entityInstance;

    $encodedPassword = $this->passwordHasher->hashPassword(
        $user,
        $user->getPassword(),
    );

    $user->setPassword($encodedPassword);

    parent::persistEntity($entityManager, $entityInstance);
}
```

Рис. 3.20 Код дії збереження інформації про користувача системи

Після завершення написання необхідного коду я отримав повністю функціональний простір адміністратора системи, який має всі розділи, необхідні для виконання поставлених задач.

Перелік розділів робочого простору адміністратора системи:

Користувачі (User). Адміністратор має можливість створювати нові облікові записи для працівників підприємства, вводячи їх особисті дані, такі як ім'я, прізвище, контактна інформація та інше.

Адміністратор також може змінювати інформацію про користувачів, включаючи їхні права доступу та ролі в системі.

У разі потреби адміністратор може видаляти або блокувати облікові записи користувачів, наприклад, якщо вони більше не працюють на підприємстві або з інших причин.

Адміністратор також має можливість надавати та керувати правами доступу різних користувачів до різних функціональних частин системи.

Звіти (Report). Адміністратор має можливість створювати різноманітні звіти щодо робочого часу працівників, включаючи інформацію про відгули, відпустки, лікарняні, переробки та інші аспекти.

Згенеровані звіти доступні для перегляду в інтерфейсі. Адміністратор може аналізувати статистику робочого часу працівників та приймати управлінські рішення, користуючись цими даними.

Звіти можуть бути відправлені по електронній пошті або надруковані для подальшого зберігання та розповсюдження.

Клієнти (Client). Адміністратор має можливість включати до системи нових клієнтів та компанії. Це сприяє ефективному управлінню замовленнями та проєктами, які пов'язані з цими клієнтами.

Задачі (Issue). Адміністратор має можливість створювати, оновлювати та видаляти інформацію про завдання. Він також може призначати користувачам завдання для їх виконання.

Записи про виконану роботу (WorkLogTime). Адміністратор має можливість оглядати звіти щодо виконаної роботи працівниками організації. Це дозволяє відстежувати продуктивність і результати їхньої діяльності.

Довідкова система підприємства (Article). Адміністратор може створити статті у розділі інформації, де розміщується важлива для працівників підприємства інформація, така як корпоративні правила та політика компанії. Також адміністратор надає доступ до ключової інформації про підприємство, яка може бути корисною для користувачів системи.

Логи системи (Log, System Log). Адміністратор може використовувати функцію перегляду журналів системи для моніторингу дій користувачів та подій, що відбуваються в системі.

Системні налаштування (SystemSettings). Адміністратор має можливість налаштовувати різноманітні системні параметри, такі як часові зони, робочі графіки, налаштування електронної пошти та інші налаштування.

Зовнішній вигляд робочого простору адміністратора системи зображено на рис. 3.21.

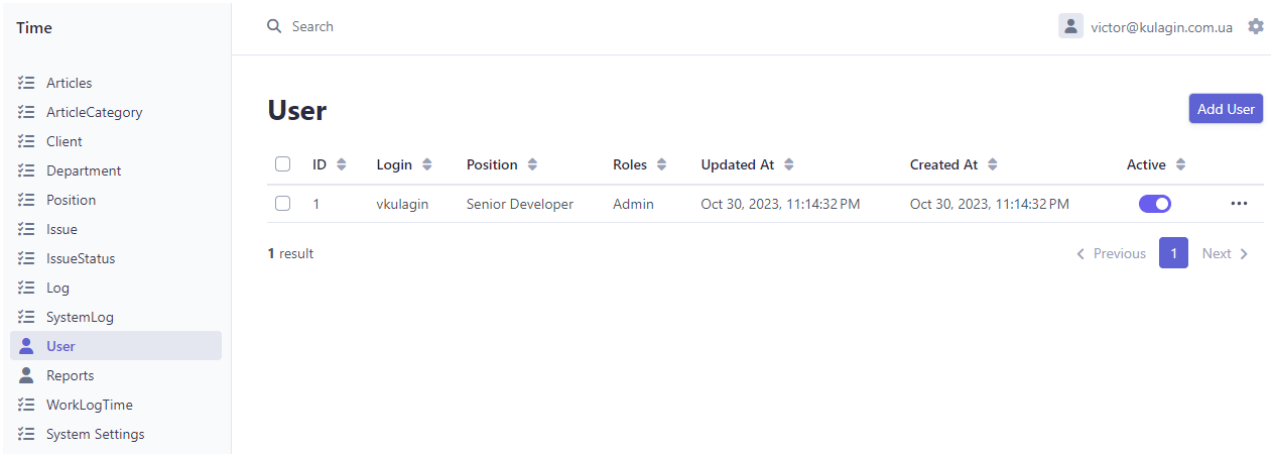


Рис. 3.21 Робочий простір адміністратора системи

3.11. Головна сторінка простору працівника підприємства, основні функції простору працівника підприємства.

Після впровадження системи авторизації була створена основна сторінка інтерфейсу. При розробці цієї сторінки було використано шаблонізатор Twig, а також мови програмування PHP, JS, CSS і HTML.

Інтерфейс програми надає можливість працівнику вносити дані про виконану роботу та час, затрачений на її виконання, переглядати історію робочого часу, вносити корективи у свої записи про робочий час. Надає можливість обрати дату, з якою будуть проводитися дії (перегляд, створення та редагування записів). Змінювати налаштування інтерфейсу для кращого адаптування до своїх потреб. Система надає можливість генерувати звіти на основі внесених даних, такі як звіти про робочий час працівника, про лікарняні, відпустку, відгули, відпрацювання тощо. Крім того у працівника підприємства є можливість переглядати інформацію з інформаційного розділу підприємства та створювати запити до адміністрації. Користувач може переглядати та редагувати власний профіль, включаючи особисті дані, контактну інформацію тощо.

Для досягнення цієї функціональності були створені відповідні моделі, контролери та шаблони, про які було згадано вище. Також були створені відповідні таблиці в базі даних для збереження цієї інформації.

Ці функції та можливості робочого простору системи дозволяють користувачам зручно та ефективно управляти своїм робочим часом, отримувати необхідну інформацію та взаємодіяти з іншими аспектами діяльності підприємства.

Загальний дизайн сторінки відповідає попередньому проектуванню, як з візуальної, так і з функціональної точки зору.

Інтерфейс робочого простору працівника підприємства зображено на рис. 3.22.

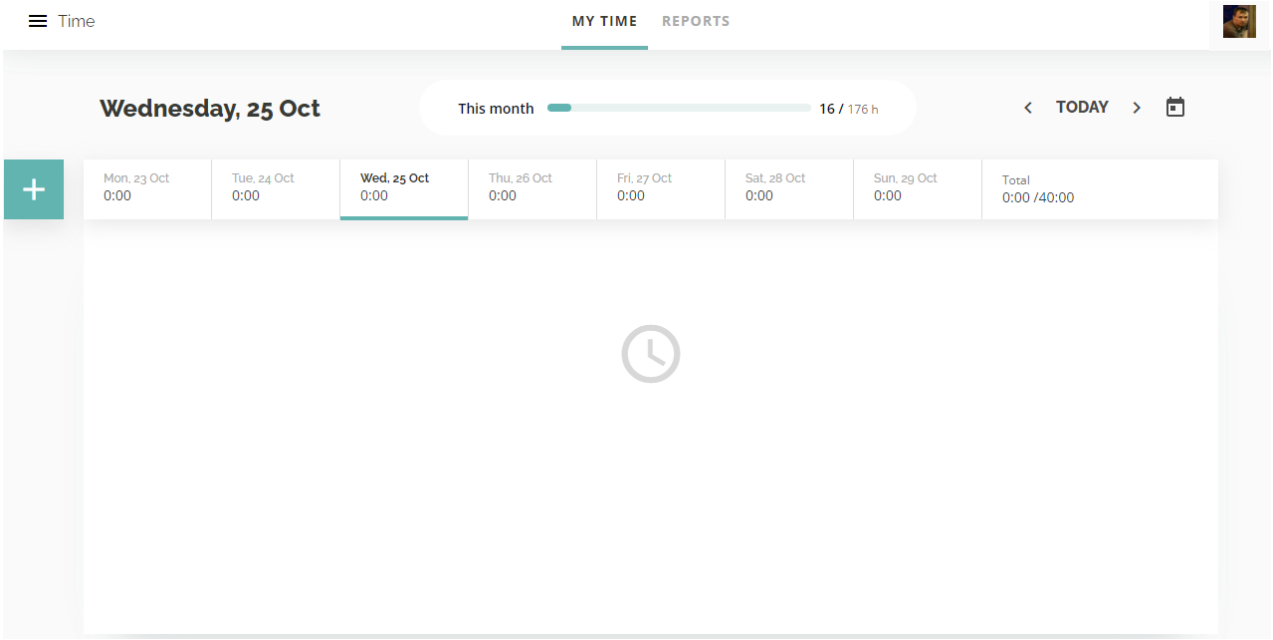


Рис. 3.22 Робочий простір працівника підприємства

3.12. Додаткова функціональність

Автоматизована система обліку робочого часу працівників на підприємстві розроблена з урахуванням можливостей для масштабування та розширення функціоналу в майбутньому.

В рамках магістерської роботи були реалізовані такі додаткові функції, як інформаційний розділ компанії для доступ до додаткової інформації про підприємство та його процеси та створення запиту до адміністрації для можливість користувачів звертатися до адміністрації з питаннями, пропозиціями або проханнями.

Інші додаткові функції, як то інтеграція та взаємодія з іншими системами, що дозволяє обмін даними та координацію між різними системами, такими як Jira, календар Google тощо, імпорт даних про користувачів з друкованих форм для завантаження інформації про працівників підприємства із паперових форм або інших документів у систему автоматизованої системи, підрахунок заробітної плати та соціального пакету, перегляд інформації про інших працівників та їх контактні дані тощо, можуть бути легко інтегровані в розроблену систему.

Висновки до розділу 3

В даному розділі було розроблено автоматизовану систему обліку робочого часу працівників на підприємстві.

Було досліджено систему версій BitBucket і створено проєкт у цій системі, що надає численні переваги, такі як зберігання коду в хмарі, доступ до нього з будь-якого місця, полегшення спільної роботи команди розробників, контроль версій програмного забезпечення та інше.

Досліджено інтегроване середовище розробки PHPStorm, використання якого надає багато переваг при розробці, описаних вище.

Досліджено та налаштовано оточення розробки у Docker. Використання Docker полегшує не лише розробку, але й розгортання проєкту на інших робочих станціях та серверах.

Встановлено фрейворк Symfony.

Досліджено та налаштовано WebPack, інструмент для збирання та управління ресурсами. Використання WebPack дозволить ефективно управляти CSS та JavaScript кодом, прискорить розробку та оптимізуватиме швидкість завантаження сторінок системи.

Розроблено систему авторизації та досліджено інтеграцію системи з сервісами Google, завдяки чому користувачі системи зможуть проходити автентифікацію не лише використовуючи локальні облікові записи, а і обліковий запис Google.

Створені всі необхідні складові частини системи. Розроблено робочий простір працівника підприємства та адміністратора системи. Описані необхідні моделі, контролери, шаблони, сервіси тощо.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці автоматизованої системи обліку робочого часу працівників на підприємстві. Досліджено основні причини використання фреймворків загалом та фреймворку Symfony зокрема, розглянуто переваги та недоліки різних фреймворків.

В ході цієї роботи було проведено дослідження принципів роботи фреймворку Symfony та основи розробки на його платформі. Досліджено архітектуру Model-View-Controller (MVC) та кожної складової частини цієї архітектури (модель, перегляд, контролер). Також було досліджено основні структурні компоненти фреймворку Symfony, такі як валідатори, форми, сервіси, слухачі, події, перекладач тощо. Також були досліджені та використані при розробці популярні та ефективні інструменти розробки, як то система версій BitBucket, інтегроване середовище розробки PHPStorm, оточення розробки Docker, інструмент для збирання та управління ресурсами WebPack тощо. Було досліджено та використано на практиці інтеграцію системи з сервісами Google.

Розроблена автоматизована система обліку робочого часу працівників на підприємстві відповідає поставленим задачам та має наступний фунціонал:

- облік працівників підприємства;
- облік робочого часу кожного працівника підприємства;
- облік відпусток, відгулів та лікарняних працівників підприємства;
- має особистий кабінет для кожного працівника підприємства;
- кожен працівник має роль в системі та відповідні права та функціонал;
- формує звітність про робочий час працівника: загальна кількість робочих годин, кількість не робочих годин, кількість годин витрачених на кожну конкретну задачу, кількість годин витрачених на кожного окремого клієнта;
- має додаткову функціональність, як то інформаційна система підприємства, а також модуль взаємодії з адміністрацією;
- система є легкою в обслуговуванні та масштабованою;

Автоматизована система обліку робочого часу робітників підприємства має простий та зрозумілий інтерфейс та надає широкий спектр можливостей для використання. В подальшому, автоматизована система може легко бути модифікована та модернізована під нові вимоги підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robin Nixon “ Learning PHP, MySQL & JavaScript. A Step-by-Step Guide to Creating Dynamic Websites.”, 6th Ed. (2021), 826 pages.
2. Васильєв О.М. “Програмування мовою PHP” (2022), 368 стор.
3. Fabien Potencier, “Symfony 5: The Fast Track” (2019), 324 pages
4. Mike McGrath, “ PHP and MySQL in Easy Steps” (2018), 192 pages
5. Symfony — Вікіпедія [Електронний ресурс]. – Посилання на ресурс: https://uk.wikipedia.org/wiki/Symfony
6. Laravel 10.x - The PHP Framework For Web Artisans [Електронний ресурс]. – Посилання на ресурс: https://laravel.com/docs/10.x
7. CodeIgniter4 User Guide [Електронний ресурс]. – Посилання на ресурс: https://codeigniter.com/user_guide/index.html
8. The Definitive Guide to Yii 2.0 [Електронний ресурс]. – Посилання на ресурс: https://www.yiiframework.com/doc/guide/2.0/en
9. Zend Documentation [Електронний ресурс]. – Посилання на ресурс: https://help.zend.com/
10. Phalcon Documentation - Introduction [Електронний ресурс]. – Посилання на ресурс: https://docs.phalcon.io/5.0/en/introduction
11. CakePHP Welcome - 5.x [Електронний ресурс]. – Посилання на ресурс: https://book.cakephp.org/5/en/index.html
12. Переваги та недоліки PHP фреймворків [Електронний ресурс]. – Посилання на ресурс: https://foxminded.ua/php-freimvorky/
13. Фреймворки мови PHP [Електронний ресурс]. – Посилання на ресурс: https://spacelab.ua/articles/3-samych-populyarnych-frejmvorka-php/
14. Найкращі PHP-фреймворки, які спрощують розробку [Електронний ресурс]. – Посилання на ресурс: https://wezom.com.ua/ua/blog/luchshie-php-frejmvorki-kotorye-uproshchayut-razrabotku-v-2019-godu

15. Чому варто використовувати PHP у 2023 році? Основні переваги та недоліки [Електронний ресурс]. – Посилання на ресурс: https://yevhenvolen.com/chomu-varto-vikoristovuvati-php-u-2023
16. Який вибрати PHP-фреймворк для розробки проекту у 2022 році [Електронний ресурс]. – Посилання на ресурс: https://icstudio.online/post/yakij-vibrati-php-frejmvork-dlya-rozrobki-proektu-u-2022-roci
17. Creating Symfony Applications [Електронний ресурс]. – Посилання на ресурс: https://symfony.com/doc/current/setup.html#creating-symfony-applications
18. Symfony Routing [Електронний ресурс]. – Посилання на ресурс: https://symfony.com/doc/current/routing.html
19. MySQL Documentation [Електронний ресурс]. – Посилання на ресурс: https://dev.mysql.com/doc/
20. Learn Git with Bitbucket Cloud [Електронний ресурс]. – Посилання на ресурс: https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
21. Better code with Bitbucket: 4 starting steps [Електронний ресурс]. – Посилання на ресурс: https://bitbucket.org/product/guides/basics/four-starting-steps#step-1-put-your-code-in-bitbucket
22. Git and Bitbucket Introduction [Електронний ресурс]. – Посилання на ресурс: https://www.w3schools.com/git/git_intro.asp?remote=bitbucket
23. PhpStorm Tutorials [Електронний ресурс]. – Посилання на ресурс: https://www.jetbrains.com/phpstorm/documentation/
24. Explore PhpStorm features [Електронний ресурс]. – Посилання на ресурс: https://www.jetbrains.com/help/phpstorm/quick-start-guide-phpstorm.html
25. Developers bring their ideas to life with Docker [Електронний ресурс]. – Посилання на ресурс: https://www.docker.com/why-docker/
26. Get Started with Docker [Електронний ресурс]. – Посилання на ресурс: https://www.docker.com/get-started/

27. What is Docker? [Електронний ресурс]. – Посилання на ресурс: https://aws.amazon.com/docker/
28. Webpack concepts [Електронний ресурс]. – Посилання на ресурс: https://webpack.js.org/concepts/
29. Webpack: Getting Started [Електронний ресурс]. – Посилання на ресурс: https://webpack.js.org/guides/getting-started/
30. An intro to Webpack: what it is and how to use it [Електронний ресурс]. – Посилання на ресурс: https://www.freecodecamp.org/news/an-intro-to-webpack-what-it-is-and-how-to-use-it-8304ecdc3c60/
31. hwi/HWIOAuthBundle [Електронний ресурс]. – Посилання на ресурс: https://github.com/hwi/HWIOAuthBundle
32. Integrating Google Sign-In into your web app [Електронний ресурс]. – Посилання на ресурс: https://developers.google.com/identity/sign-in/web/sign-in
33. Symfony Documentation [Електронний ресурс]. – Посилання на ресурс: https://symfony.com/doc/current/index.html

ДОДАТКИ

Додаток А

А.1 Код шаблону форми авторизації

```
<body>
{% block body %}
        <div class="auth-container">
                <div class="auth-content">
                        <div class="auth-name">Time</div>
                        <div class="auth-box">
                                <div class="auth-text"> Authorize with your login or google account</div>
                                <form method="post">
                                {% if error %}
                                        <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
                                {% endif %}
                                {% if app.user %}
                                        <div class="mb-3">
                                                You are logged in as {{ app.user.userIdentifier }}, <a href="{{ path('app_logout') }}">Logout</a>
                                        </div>
                                {% endif %}
                                <div class="row mb-2">
                                        <div class="col-sm text-end pt-2">

                                                <label for="inputEmail">Login</label>
                                        </div>
                                        <div class="col-sm">
```

Продовження Додатку А1

```
                                    <input type="text" value="{{
last_username }}"                                    name="email"
id="inputEmail" class="form-control" autocomplete="email" required autofocus>
                                    </div>
                                </div>
                                <div class="row mb-4">
        <div class="col-sm text-end pt-2">
                                    <label
for="inputPassword">Password</label>
                                    </div>
                                    <div class="col-sm">
                                        <input type="password"
name="password" id="inputPassword" class="form-control"
autocomplete="current-password" required>
                                    </div>
                                </div>
                                <div class="row">
                                    <input type="hidden" name="_csrf_token"
value="{{ csrf_token('authenticate') }}">
                                </div>
                                <button class="btn btn-primary mb-3"
type="submit">
                                    Sign in
                                </button>
                                </form>
                                <a href="{{ path('hwi_oauth_service_redirect',
                            {'service': 'google' }) }}"    class="auth-button">
                                    SIGN IN WITH GOOGLE
```

```
                    </a>
                </div>
            </div>
        </div>
{% endblock %}
</body>
</html>
```

А.2 Код сутності Issue (Задача)

```
#[ORM\Entity(repositoryClass: IssueRepository::class)]
#[ORM\HasLifecycleCallbacks]
class Issue
{
    use IdTrait;
    use ActiveTrait;
    use CreatedUpdatedTrait;
    #[ORM\Column(length: 255)]
    private ?string $name = null;
    #[ORM\Column(length: 255)]
    private ?string $description = null;
    #[ORM\ManyToOne(inversedBy: 'issues')]
    #[ORM\JoinColumn(nullable: false)]
    private ?Client $client = null;
    #[ORM\ManyToOne(targetEntity: User::class, inversedBy: 'issues')]
    private Collection $user;
    #[ORM\ManyToOne(inversedBy: 'issues')]
    #[ORM\JoinColumn(nullable: false)]
    private ?IssueStatus $status = null;
```

```
    #[ORM\OneToMany(mappedBy: 'issue', targetEntity:
WorkLogTime::class)]
    private Collection $workLogTimes;
    public function __construct() {
        $this->user = new ArrayCollection();
        $this->workLogTimes = new ArrayCollection();
    }
    public function getName(): ?string {
        return $this->name;
    }
    public function setName(string $name): static {
        $this->name = $name;
        return $this;
    }
    public function getDescription(): ?string
    {
        return $this->description;
    }
    public function setDescription(string $description): static
    {
        $this->description = $description;
        return $this;
    }
    public function getClient(): ?Client
    {
        return $this->client;
    }
    public function setClient(?Client $client): static
```

```
    {
            $this->client = $client;
        return $this;
    }
    public function getUser(): Collection
    {
        return $this->user;
    }
    public function getStatus(): ?IssueStatus
    {
        return $this->status;
    }
    public function setStatus(?IssueStatus $status): static
    {
        $this->status = $status;

        return $this;
    }

    /**
     * @return Collection<int, WorkLogTime>
     */
    public function getWorkLogTimes(): Collection
    {
        return $this->workLogTimes;
    }
}
```

А.3 Код опису полів для відображення контролеру робочого простору адміністратора системи

```
public function configureFields(string $pageName): iterable
{
    $user = $this->getContext()->getEntity()->getInstance();
    yield IdField::new('id')
        ->hideOnForm();
    yield TextField::new('login');
    yield TextField::new('email')->hideOnIndex();
    yield TextField::new('phone')->hideOnIndex();
    yield AssociationField::new('position')
        ->setFieldFqcn(Position::class)
        ->setFormTypeOption('class', Position::class);
    if (!$user || !$user->getId()) {
        yield TextField::new('password')->hideOnIndex();
    }
    yield TextField::new('firstName', 'First Name')->hideOnIndex();
    yield TextField::new('lastName', 'Last Name')->hideOnIndex();
    yield TextField::new('middleName', 'Middle    Name')->hideOnIndex();
    yield TextField::new('middleName', 'Middle    Name')->hideOnIndex();
    yield ChoiceField::new('roles', 'Roles')
        ->allowMultipleChoices()
        ->autocomplete()
        ->setChoices([
            'User' => User::ROLE_USER,
            'Admin' => User::ROLE_ADMIN,
        ]);
    yield DateTimeField::new('updatedAt')
        ->hideOnForm()
        ->setRequired(false)
```

```
            ->setColumns(3);
        yield DateTimeField::new('createdAt')
            ->setRequired(false)
            ->hideOnForm()
            ->setColumns(3);
        yield BooleanField::new('active');
    }
```

Додаток Б

Копії публікації з темою магістерської роботи

КУЛАГІН В.П., ДЕМКІВСЬКА Т.І.

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ РОБОЧОГО ЧАСУ РОБІТНИКІВ ПІДПРИЄМСТВА

KULAHIN V.P., DEMKIVSKA T.I.

DEVELOPMENT OF SOFTWARE FOR THE AUTOMATED SYSTEM OF RECORDING THE WORKING TIME OF THE WORKERS IN ENTERPRISE

Annotation. The article presents the analysis and characteristics of the automated system of recording the working time of the workers in enterprise.

The paper version of data entry and storage is no longer relevant, so modernity needs to change. Modern technologies allow the use of electronic databases in enterprises and store all important data in them.

The development of an automated system of recording the working time of the workers in enterprise is a topical issue today, because it facilitates the work of the employees and provides information about workers efficiency. Also, this method is more reliable and practical to use and is to maintain the confidentiality of information.

Keywords: Automated workplace, Database, Information system, Accounting system, Graphical interface.

Вступ

На сьогоднішній день паперовий варіант обліку робочого часу працівників підприємства вже дуже застарілий і не актуальний. Він потребує багато часу для його ведення та для формування різноманітної звітності. Крім того виникають складності з конфіденційністю інформації.

Автоматизація різноманітних процесів на підприємстві дозволяє полегшити роботу працівників та надає можливості для швидкого формування різноманітних звітів. Це дає змогу контролювати ефективність кожного працівника та полегшує багато внутрішніх процесів підприємства. Створюється програмний продукт, який допоможе вести облік працівників, вести облік робочого часу кожного працівника, облік часу

витраченого на кожну конкретну задачу, формувати звітність про кількість витраченого часу на кожного клієнта та формувати звіти про витрачений час. Даний програмний продукт буде вести облік відгулів, лікарняних, відпусток та формувати різноманітні звіти.

Постановка завдання

Метою даного дослідження є розробка програмного забезпечення для автоматизованої системи обліку робочого часу працівників на підприємстві. Автоматизована система має серверну та клієнтську частини.

Програмний продукт, що розробляється повинен виконувати наступні основні задачі:

- облік працівників підприємства;
- облік робочого часу кожного працівника підприємства;
- облік відпусток, відгулів та лікарняних працівників підприємства;
- мати особистий кабінет для кожного працівника підприємства;
- кожен працівник повинен мати роль в системі та відповідні права та функціонал;
- формувати звітність про робочий час працівника: загальна кількість робочих годин, кількість не робочих годин, кількість годин витрачених на кожну конкретну задачу, кількість годин витрачених на кожного окремого клієнта;
- додаткова функціональність;

Основна частина

Автоматизована система обліку робочого часу працівників на підприємстві складається з двох основних частин: серверної (бекенд) та користувацької (фронтенд). Серверна частина відповідає за обробку та зберігання даних, а користувацька - за взаємодію користувача системи з цими даними.

Для розробки серверної частини даного програмного забезпечення обрано мову програмування PHP та фреймворк Symfony.

PHP - це одна з найпопулярніших мов програмування на сьогоднішній день, особливо серед веб-розробників. Це мова програмування з відкритими вихідними кодами, над розвитком якої працюють програмісти з усього світу. За статистикою, кожен шостий

програмний продукт створено на PHP. Мова програмування PHP постійно розвивається та отримує оновлення.

Symfony - це популярний веб-фреймворк PHP з відкритими вихідними кодами який відповідає шаблону Model, View, Controller (MVC).

На сьогоднішній день це один з найпопулярніших фреймворків для розробки веб-проектів. Він надає можливість пришвидшити розробку та відповідає усім сучасним критеріям безпеки.

Найважливішим компонентом у Symfony є компонент Ядро. Це основний клас управління навколишнім середовищем і відповідає за обробку запитів http.

Компонент HttpFoundation допомагає зрозуміти HTTP. Він надає об'єкт запиту та відповіді для інших компонентів.

Крім того, Symfony надає безліч функцій. Він використовує Doctrine для об'єктного реляційного відображення (ORM) та гілку як механізм шаблонів. Більше того, Symfony використовує систему шаблонів Twig.

Клієнтська частина написана з використанням шаблонизатора Twig, мови розмітки гіпертекстових документів HTML, мови стилів CSS та додаткових бібліотек та інструментів.

Для зберігання даних використовується система управління реляційними базами даних MySQL. У реляційній базі даних дані зберігаються в окремих таблицях, завдяки чому досягається виграш у швидкості й гнучкості. Таблиці зв'язуються між собою за допомогою відносин, завдяки чому забезпечується можливість поєднувати при виконанні запиту дані з декількох таблиць.

В базі даних зберігаються облікові дані кожного працівника підприємства, облікові дані клієнтів а також інформація про задачі, які виконуються працівниками підприємства. Всі ці дані мають зв'язки між собою. Клієнт має зв'язок з задачами, які поставлені ним. Працівник має зв'язок з задачами, які він виконує. Клієнт і працівник також мають зв'язок через задачі.

Автоматизована система виконує наступні основні задачі:

- облік працівників підприємства;
- облік робочого часу кожного працівника підприємства;
- облік відпусток, відгулів та лікарняних працівників підприємства;
- має особистий кабінет для кожного працівника підприємства;
- кожен працівник має роль в системі та відповідні права та функціонал;

- формує звітність про робочий та неробочий час працівника, кількість годин витрачених на кожну конкретну задачу, кількість годин витрачених на кожного окремого клієнта;

Автоматизована система має систему авторизації та реалізує різні рівні доступу для користувачів.

Працівник підприємства вносить дані про робочі часи, витрачені на виконання поставленої задачі. Також вносить запити на лікарняні, відпустку, відгули.

Менеджер створює записи про працівників, клієнтів та поставлені задачі. Назначає працівника(ів) на ці задачі. Формує звіти для клієнтів про витрачений час на задачі. Формує звіти про загальний робочий та неробочий час працівника. Підтверджує запити на лікарняні, відпустку, відгули.

Адміністратор системи має права всіх інших груп. Має права для створення користувачів автоматизованої системи та керує правами користувачів. Має можливість змінювати різноманітні системні налаштування.

Система також надає можливість швидко знайти необхідну інформацію:

- облікові дані працівника;
- дані про задачу (хто виконував та скільки часу було витрачено)
- дані про клієнта (облікові дані, задачі, кількість витраченого часу)
- тощо.

Автоматична система надає можливість друку звітної інформації у форматі А4.

Висновки

Досліджено задачі керування обліком робочого часу працівників підприємства а також методи автоматизації розв’язання цих задач за допомогою програмного забезпечення.

Проведено дослідження PHP фреймворку Symfony. Досліджено шаблон Model, View, Controller (MVC) та об'єктно-реляційний проектор (ORM) Doctrine.

Розроблено автоматизовану інформаційну систему обліку робочого часу працівників підприємства.

Розроблена база даних для збереження інформації про працівників підприємства, клієнтів та виконувані задачі.

Розроблена автоматизована інформаційна система дозволяє спростити облік робочого часу працівників підприємства та формування звітності. Використання створеного програмного забезпечення підвищує конфіденційність інформації.

Додаток В

Презентація кваліфікаційної роботи

Розробка програмного забезпечення для автоматизованої системи обліку робочого часу робітників підприємства

Виконавець: Віктор Кулагін
Науковий керівник: Тетяна Демківська

" Time is money. "
– Benjamin Franklin

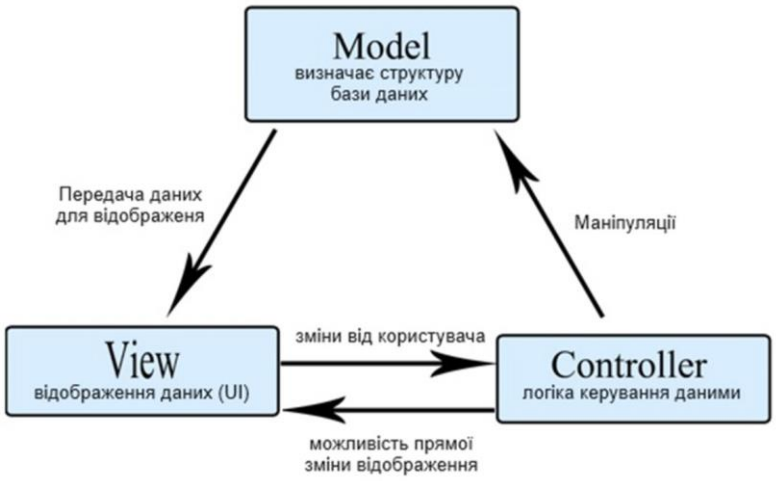
Мета роботи

Розробка автоматизованої системи обліку робочого часу робітників підприємства на базі Symfony, проведення аналізу існуючих аналогічних систем та вивчення можливостей Symfony для розробки власної системи, яка буде відповідати специфічним вимогам підприємства.

Навіщо потрібні фреймворки

- *"інвестуванні у завдання, а не у технологію"*
- *відповідність бізнес-правилам*
- *чітка структура*
- *легкість підтримки та оновлення*
- *заощадження часу*
- *взаємодія з іншими бібліотеками PHP*

Архітектура Model-View-Controller (MVC)



Популярні фреймворки

- *Symfony*
- *Laravel*
- *CodeIgniter*
- *Yii*
- *Zend*
- *Phalcon*
- *CakePHP*

Робочий простір працівника підприємства

- **інтерфейс вибору дати**
- **записи про робочий час**
- **звіти**
- **налаштування інтерфейсу та системні налаштування**
- **профіль користувача**
- **додаткові функції**

Робочий простір адміністратора системи

- **користувачі (працівники)**
- **звіти**
- **клієнти, компанії, задачі**
- **записи про виконану роботу**
- **довідкова система підприємства**
- **логи дій користувачів та системні логи**
- **системні налаштування**
- **додаткові функції**

Висновки

В ході цієї роботи було проведено дослідження принципів роботи фреймворку Symfony та основи розробки на його платформі.

Досліджені та використані при розробці популярні та ефективні інструменти розробки.

Розроблена автоматизована система обліку робочого часу працівників на підприємстві, яка відповідає поставленим задачам, має простий і зрозумілий інтерфейс та надає широкий спектр можливостей для використання