Хмельницький національний університет
Факультет програмування та комп'ютерних і телекомунікаційних систем
Кафедра кібербезпеки та комп'ютерних систем і мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення

Назва теми

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

КРМКІ.190190.19.01.11 ПЗ

Виконав: студент 2 курсу, група КІ1м-19-1 _____ Мілер В.П.
Підпис

Керівник доц., к. т. н, доцент кафедри КбКСМ _____ Орленко В.С.
Підпис

Нормоконтролер доц., к. т. н, доцент кафедри КбКСМ _____ Муляр І.В.
Підпис

До захисту допускаю:
Зав. кафедри КбКСМ, к.т.н., доц _____ Кльоц Ю.П.
Підпис

4 12 2020 р.

Хмельницький, 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ПРОГРАМУВАННЯ ТА КОМП'ЮТЕРНИХ І ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

Кафедра КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА ПІДГОТОВКИ МАГІСТРА

ЗАТВЕРДЖУЮ

Зав. кафедри Ю.П. Кльоц

"4" 09 2020 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мілеру В.П..

Прізвище, ім'я, по батькові студента

Тема проекту (роботи) Вдосконалення методу проєктування вебдодатків на основі об'єктно реляційного перетворення

1. Керівник проекту (роботи) к.т.н., доц. Орленко В.С.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом № 118 ректора університету додаток №23 від 01.09.2020

2. Строк подання студентом проєкту (роботи) на кафедру 1.12.2020

3. Вихідні дані до проекту (роботи) Методи структурного синтезу, слабоструктуровані дані, , фреймворки ,об'єктно реляційне перетворення, веб-додатки на платформі LAMP

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Сформувати критерії якості вебдодатків, і провести аналіз існуючих методів розробки, зберігання і взаємодії вебзастосунків Провести синтез ефективної архітектури і алгоритмів функціонування вебдодатків. Визначення рівнів абстракції в вебдодатку, їх взаємодії і життєвого циклу. Розробити метод взаємного об'єктно-реляційного перетворення для організації взаємодії вебдодатку з базою даних і зберігання даних, а також шаблону проектування на його основі. Рішення практичних завдань розробки вебдодатків на базі запропонованих методів, в тому числі методів взаємодії вебзастосунків

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Тема, мета магістерської роботи, задачі дослідження, наукова новизна, практична цінність,. Методи розробки вебдодатків. Існуючі моделі архітектур вебдодатків. Запропонована модель архітектури. Метод об'єктно-реляційного перевторення. Практична реалізація методики кешування. Порівняльний аналіз розробленої системи. Висновки

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання пр
Відповідальний за оформлення ДП	Муляр І.В.		

7. Дата видачі завдання «1» лютого 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи	Термін виконання етапів проекту (роботи)	При
1	Вибір напряму дослідження та узгодження тематики ДРМ з керівником	2.02.2020	
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	2.03.2020	
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	1.04.2020	
4	Робота над розділом 2 – розробка моделей і методів для вирішення поставленої задачі	1.05.2020	
5	Робота над науковою статтею	1.06.2020	
6	Робота над розділом 3 – розробка алгоритмів та технологій, їх аналіз	1.09.2020	
7	Робота над розділом 4 – проектування ПЗ для вирішення поставленої задачі	1.10.2020	
8	Узгодження отриманих; оформлення пояснювальної записки згідно вимог	1.11.2020	
9	Оформлення графічної частини	11.11.2020	
10	Попередній захист КРМ	15.11.2020	
11	Захист КРМ на засіданні ЕК	10.12.2020	

Студент _____ Підпис В.П. Міляр Ініціали, прізвище

Керівник проекту (роботи) _____ Підпис Орленко В.С. Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення.

Автор роботи: Мілер В.П.

Керівник роботи: к.т.н., доц. Орленко В.С.

Пояснювальна записка: 94 с., 22 рис., 3 табл., 3 дод., 39 джерел.

ФРЕЙМВОРКИ, ДИНАМІЧНА ІНТЕГРАЦІЯ, ОБ'ЄКТНО-РЕЛЯЦІЙНЕ ПЕРЕТВОРЕННЯ, ВЕБДОДАТОК.

Мета магістерської роботи полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу.

Дана кваліфікаційна робота присвячена розробці моделі «ОРП», особливістю якої є те, що *n* кількість відношень, які використовуються для відображення може бути більше 1. Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність, На основі моделі вдосконалено метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти.

Дата 03.12.2020 Підпис студента

ANNOTATION

a master's degree work of Miler Vadym

entitled «Improving the method of designing web applications based on object-relational transformation».

Mentor: Viktoriia Orlenko

Total volume of work: 94 pages, 22 figures, 3 tables, 3 appendices, 39 references.

FRAMEWORKS, DYNAMIC INTEGRATION, OBJECT-RELATIONAL TRANSFORMATION, WEB APPLICATION.

The purpose of the master's thesis is to increase the efficiency of development and interaction of components of web applications, by reducing time and number of errors at the stage of structural synthesis.

This thesis is devoted to the development of the model "ORP", the feature of which is that n the number of relationships used to display can be more than 1. The application of the proposed technique reduces memory costs (no need to create n models, everything is in one) and Increases productivity. Based on the model, the object-relational transformation method has been improved for the synthesis and development of web applications, which provides the conversion of database server tables into software components.

Date 03.12.2020 Signature

ЗМІСТ

Вступ 6
1 Аналіз методів проєктування вебзастосунків 9
1.1 Особливості проєктування веборієнтованих систем 9
1.2 Архітектура вебдодатків 13
1.3 Принципи розробки вебдодатків 26
1.4 Постановка задачі 38
2 Моделювання процесу проєктування та створення вебдодатку 41
2.1 Модель запропонованої архітектури 41
2.2 Проєктування на рівні контролерів 42
2.3 Методи розробки рівня моделі 61
2.4 Висновки 74
3 Метод розробки вебдодатків на основі об'єктно-реляційного перетворення 75
3.1 Алгоритм роботи зі сторонніми бібліотеками 75
3.2 Вдосконалений метод модульної збірки вебзастосунків на основі ОРП 77
3.3 Висновки 79
4 Застосування методів розробки вебзастосунків 80
4.1 Проєктування вебзастосунку 80
4.3 Висновки 83
Висновки 88
Перелік джерел посилань 90
Додаток А. Експертні оцінки якості 95
Додаток Б Копії наукових праць 96
Додаток В Презентація. 101

ВСТУП

На теперішній час існує широкий спектр програмного та апаратного забезпечення для побудови вебсистем. Однак подальший розвиток вебтехнологій стає неможливим без побудови формально-математичної бази, насамперед без створення офіційної моделі вебсистеми. Без такої бази важко розробити методи її проектування. Таким чином, сьогодні не існує розширень для вебтехнологій усталених методів та засобів аналізу та проєктування інформаційних вебсистем (зокрема, структурних методологій). Відсутність математичної моделі вебсистеми унеможливлює розробку інтелектуальних інструментів їх адміністрування, які базуються на алгоритмах її оптимізації та прогнозування поведінки в часі.

Необхідність моделювання багаторівневої вебсистеми глобального характеру призводить до потреби побудови їх формальної моделі та алгоритмів їх оптимізації як основи для якісного розвитку ефективних вебсистем, незалежно від їх складності та характеру.

Сьогодні найбільш перспективними є системи інтеграції даних, які працюють із використанням технології Mashup. Mashup - це технологія вебдизайну, яка дозволяє користувачам поєднувати різні типи даних з декількох джерел в один інтегрований інструмент. Розробка технології динамічної інтеграції слабоструктурованих даних з урахуванням змісту даних у різних вебсистемах і здатна враховувати особливості різних вхідних інформаційних систем з урахуванням таких факторів, як: відсутність теоретичного обґрунтування методів семантичного системи обробки даних засоби динамічної обробки інформаційних ресурсів інтегрованих систем Mashup. Практичний фактор обробки даних різної структури та форматів подання в різних вебсистемах з їх динамічною інтеграцією пов'язаний з вирішенням проблем поліпшення пошуку інформації в Інтернеті в сучасну еру стрімкого зростання інформаційних даних різного характеру та змісту, інтеграції різнорідних інформаційні системи, що містять різного роду та зміст інформаційні ресурси та величезний попит на динамічні системи інтеграції

даних, засновані на технології Mashup, відкриваючи нові та широкі можливості для використання інформаційних ресурсів.

Завдання створення набору технологічних інструментів на основі теоретичного обґрунтування методів, моделей та принципів семантичної обробки даних різних структур та форматів презентацій, які знаходяться на різних вебсайтах з їх динамічною інтеграцією, є актуальним.

Мета магістерської роботи полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу.

Об’єкт дослідження: процес створення вебзастосунків

Предмет дослідження: методи структурного синтезу та розробки вебзастосунків.

Задачі досліджень у роботі формулюються наступним чином:

1. Провести аналіз існуючих підходів до синтезу та розробки веборієнтованих систем.
2. Визначити можливості підвищення ефективності структурного синтезу, взаємодії вебзастосунків і їх зберігання.
3. Проектування семантичної моделі процесів і розробки вебзастосунків.
4. Розробити ефективну архітектуру та алгоритм функціонування вебзастосунків.
5. Розробити метод об'єктно-реляційного перетворення для синтезу та розробки вебзастосунків, а також шаблону проектування на його основі.
6. Реалізація розроблених алгоритмів структурного синтезу вебзастосунків на базі запропонованого методу.

Наукова новизна роботи полягає в наступному:

1. Вдосконалена модель архітектури вебзастосунку, в якій враховуються індивідуальні і системні властивості процесів і модулів.

2. Запропоновано метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти.

Методика проведення досліджень: методи теорії системного аналізу, теорії множин, теорії реляційних баз даних, теорії алгоритмів теорії графів, і методів імітаційного моделювання

Практична цінність. Практична реалізація розроблених методик дозволила створити новий клас адаптивних автоматизованих систем, що суттєво підвищують ефективність процесу збірки вебзастосунків і покращені структурності, повторюваності, модифікованості.

Публікації. По темі магістерської роботи опубліковано 1 стаття у нефаховому журналі (збірник НПК МНІС ІП-2020), 1 - теза доповідей на всеукраїнській конференції (Тези доповідей XVI Міжнародної науково-практичної конференції "Військова освіта і наука: сьогодення та майбутнє" [Текст] / за заг. редакцією Ігоря Толока.– К. : ВІКНУ, 2020)

1 АНАЛІЗ МЕТОДІВ ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКІВ

1.1 Особливості проєктування веборієнтованих систем

Інформація, особливо її автоматизована обробка, як і раніше залишається важливим фактором підвищення ефективності будь-якої організації. Важливими у використанні інформації є методи її реєстрації, обробки, накопичення та передачі; систематизоване зберігання інформації та її вихід у необхідній формі; виробництво нової числової, графічної та іншої інформації.

Підвищити ефективність інформаційних систем завдяки наскрізній структурі та сумісності інформаційних систем, що дозволяє усунути дублювання та забезпечити багаторазове використання інформації, встановити певні інтеграційні зв'язки, обмежити кількість показників, зменшити інформаційні потоки, збільшити використання інформації . Інформаційна система повинна підтримувати такі функції, як надання інформації (наприклад, необхідної користувачам для вирішення науково-виробничих проблем) та створення найбільш зручних умов для її розповсюдження (наприклад, проведення адміністративної, організаційної, дослідницької та виробничої діяльності, що забезпечує її ефективну роботу розподіл) [24].

Інтернет впевнено увійшов у наше життя, зокрема громадське. Паралельно з процесом збільшення кількості активних користувачів мережі, якість послуг провайдерів послуг Інтернету постійно вдосконалюється, перш за все, це стосується регулярного зменшення вартості пропускної здатності з точки зору окремих кінцевих користувачів..

На момент масового впровадження графічного інтерфейсу користувача на платформі персонального комп’ютера ця концепція призвела до вибухового поширення Інтернету у світі. Web 2.0 також слід розглядати як сукупність технологій, а не як єдину технологію. Водночас на перший план виходить ідея

співпраці, колективної творчості, миттєвого розповсюдження та отримання відгуків від споживачів інформації.

Сьогодні стає все більш очевидним, що сучасні інформаційні системи, зокрема веб-орієнтовані системи, повинні все більше покладатися на семантику предметної області, на знання про неї. Якщо базою знань системи є онтологія, то система, заснована на знаннях, повинна бути онтологічно орієнтована [11].

Базова модель автоматизованої системи не акцентує уваги на характеристиках відкритості, тому вона була доповнена узагальненою структурою інформаційної системи (рис 1.1)

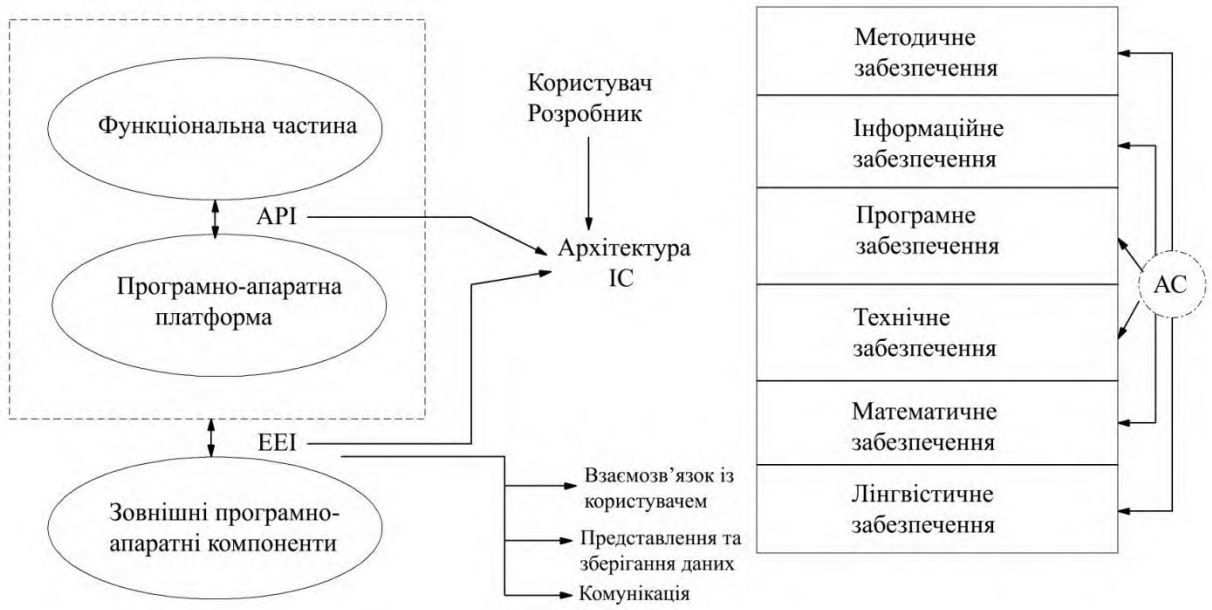


Рисунок 1.1 - Загальна структура відкритої автоматизованої інформаційної системи

Загальна структура відкритої автоматизованої системи представлена у вигляді інтерфейсів API , внутрішніх і міжрівневих протоколів та служб, які згруповані за двома внутрішніми рівнями системи і одному зовнішньому.

Функціональність будь-яких систем залежить від організаційно-управлінської структури організації, технології управління документами, тощо. На сучасному рівні розробки програмного забезпечення існує багато різного програмного забезпечення для обробки інформації, написаного різними мовами

програмування вищезазначеними методами. Різноманітність програмних засобів пов’язана зі специфікою кожної галузі, в якій здійснюється обробка інформації.

У той же час характерною особливістю веборієнтованих інформаційних систем, зокрема систем управління вмістом вебсайтів, є наявність великої кількості документальної інформації, і самі ці документи часто формуються динамічно, на основі певних процедур. Тому вебінформаційна система, що базується на знаннях, повинна бути зосереджена на роботі як з онтологіями, так і з документами та з кількома документами [17].

Переважно, під час розробки веборієнтованих інформаційних систем, які будуть використовуватися в Інтернеті або в локальній мережі певної установи, виникає завдання створення структурно складних систем.

Завдання підходів до інтеграції даних полягає в враховуванні значних проявів неоднорідності, властивої інформаційним системам, які створювалися без уніфікованим ставлення до даних [16]. Дослідник з Цюріхського університету Клаус Діттріх запропонував схему класифікації технологій інтеграції даних з шести рівнів [16] (рис. 1.2):

Рисунок 1.2 - Класифікація рівнів інтеграції даних

1) Інтеграція вручну. Користувач сам інтегрує дані, використовуючи різні типи інтерфейсів та мови запитів. Користувачеві необхідно детально розібратися в семантиці отриманих даних та їх структурі.

2) Загальний інтерфейс користувача. Користувачеві надається єдиний користувальницький інтерфейс, такий як браузер, який забезпечує доступ до різнорідних даних. Однак самі дані залишаються неоднорідними, і їх структурування та інтеграція повинні впроваджуватися користувачем.

3) Інтеграція за програмами. Підхід заснований на використанні програм, які отримують доступ до різних неоднорідних джерел даних та забезпечують користувачеві інтегрований результат [27].

4) Інтеграція за допомогою проміжного програмного забезпечення. Таке програмне забезпечення - це функціональність, яку можуть використовувати самі програми. Однак такий підхід не повністю знімає завдання інтеграції з додатків, а лише полегшує, вирішити проблему в повному обсязі можна лише взаємодіючи з додатком. Крім усього іншого, для створення загальної системи інтеграції зазвичай потрібен цілий набір додаткових інструментів та проміжного програмного забезпечення.

5) Єдиний доступ до даних. На цьому рівні досягається логічна інтеграція даних, яка досягається на рівні доступу до даних. Різні програми отримують однакове віртуальне бачення фізично розподілених та неоднорідних даних, тоді як самі фізичні дані не використовуються. Цей тип віртуалізації даних має свої незаперечні переваги, але структурування та об’єднання даних вимагає часу та значних ресурсів.

6) Загальне зберігання даних. На цьому рівні інтеграція досягається поєднанням даних з різних систем зберігання в одну. Цей підхід сприяє швидкості роботи єдиної системи джерел даних, але вимагає додаткового часу для передачі даних, і такий підхід може призвести до дублювання інформації.

Отже, модель інформаційної бази веборієнтованої системи повинна базуватися на двох компонентах вузла: онтології предметної області та набору документів, а також взаємозв’язках між цими компонентами [13].

1.2 Архітектура вебдодатків

Створюючи вебдодаток, слід мати на увазі три основні принципи. З точки зору замовника, програма повинна бути простою, естетично приємною і відповідати більшості їхніх проблем. З ділового аспекту веб-програма повинна залишатися узгодженою з відповідністю товару / ринку . З точки зору інженера-програміста, веб-додаток повинен бути масштабованим, функціональним та здатним витримувати великі навантаження на транспорт.

Усі ці проблеми розглядаються в архітектурі вебпрограми. Ми розглянемо основні поняття будь-якого сучасного веб-додатку та пояснимо, як можуть відрізнятися шаблони архітектури залежно від програми, яку ви створюєте.

Основне визначення вебпрограми - це програма, яка працює на браузері. Це не веб-сайт, але межа між ними нечітка [23]. Щоб відрізнити вебдодаток від вебсайту потрібно врахувати три формальні характеристики. Вебпрограма:

- вирішує певну проблему, навіть якщо це просто пошук якоїсь інформації;
- настільки ж інтерактивний, як і настільний додаток;
- має систему управління вмістом.

Під вебсайтом традиційно розуміють просто поєднання статичних сторінок. [18]. Але сьогодні більшість веб-сайтів складаються як із статичних, так і з динамічних сторінок, що робить майже всі сучасні веб-сайти - ви вже здогадалися! - веб-додатки. У цій статті ми будемо використовувати терміни як взаємозамінні.

Комп'ютер, смартфон або будь-який інший пристрій, з яким ви переглядаєте, називається клієнтом. Іншу половину веб-рівняння називають

сервером, оскільки він обслуговує дані, які ви вимагаєте. Їх спілкування називається модель клієнт-сервер, основною проблемою якої є отримання вашого запиту та повернення відповіді назад.

Архітектура вебдодатків - це механізм, який визначає спосіб взаємодії компонентів програми між собою [31]. Або, іншими словами, спосіб підключення клієнта та сервера встановлюється архітектурою веб-додатків.

Щоб зрозуміти компоненти архітектури вебдодатків, нам потрібно зрозуміти, як вони використовуються при виконанні найосновніших дій - отримання та відповіді на вебзапит.

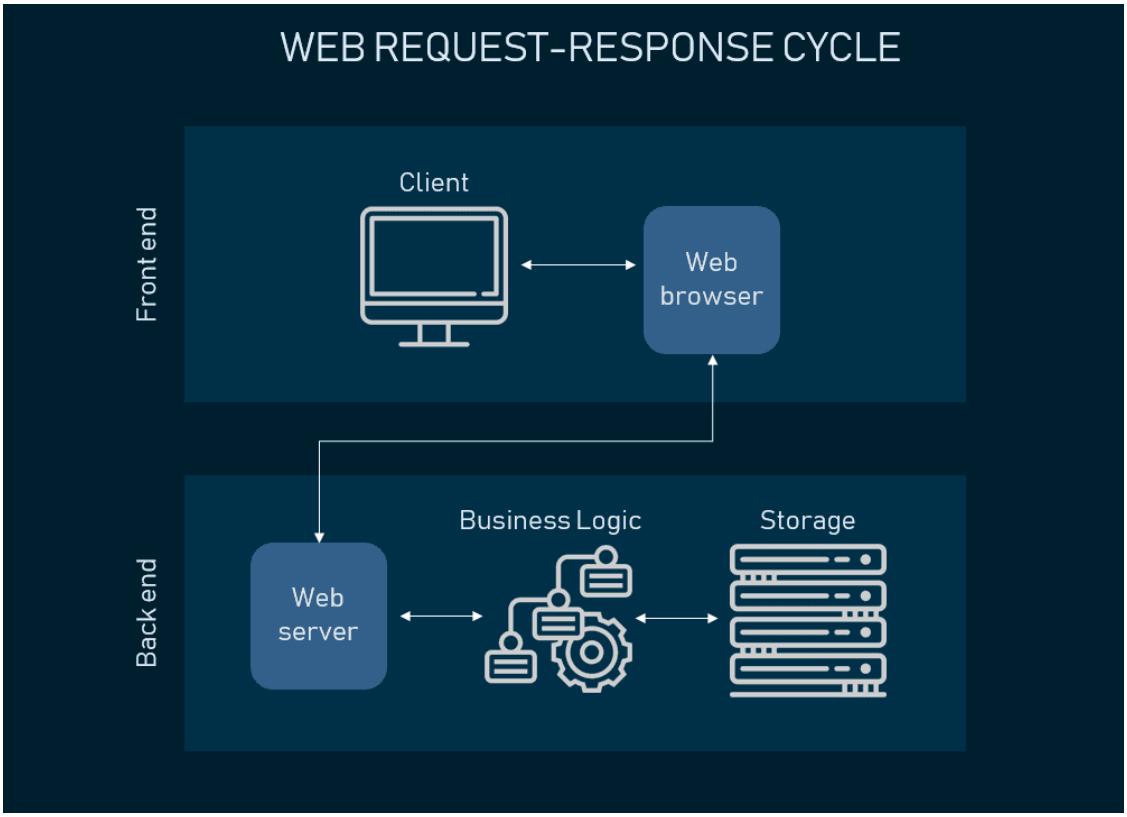


Рисунок 1.3 - Цикл запиту-відповіді

Давайте подивимося на Amazon.com, щоб проілюструвати наше пояснення.

Спочатку ви відвідуєте amazon.com. Ви вводите URL-адресу, і після натискання клавіші Enter ваш браузер готується розпізнати цю URL-адресу, оскільки вона повинна знати адресу сервера, де знаходиться сторінка. Тому він надсилає ваш запит до Центру імен доменів (DNS), сховища доменних імен та їх

IP-адрес [8]. Якщо ви вже відвідували Amazon з того самого браузера, він витягне адресу з кешу. Потім браузер надсилає запит на знайдену IP-адресу за допомогою протоколу HTTPS [25].

По-друге, вебсервер обробляє запит . Вебсервер, на якому знаходиться Amazon.com, ловить запит і відправляє його в область зберігання, щоб знайти сторінку та всі наступні дані. Але його маршрут проходить через бізнес-логіку (також називається логікою домену та логікою додатків). Бізнес-логіка керує способом доступу до кожного фрагмента даних і визначає цей робочий процес спеціально для кожної програми. Коли бізнес-логіка обробляє запит, він надсилає його до сховища для пошуку шуканих даних.

По-третє, ви отримуєте свої дані . Ваша відповідь повертається до вас, і ви бачите вміст вебсторінки на своєму дисплеї. Графічний інтерфейс, який ви бачите під час прокрутки Amazon або будь-якого іншого вебсайту, називається інтерфейсом програми - він відображає всі компоненти UX та UI, щоб користувач мав доступ до інформації, яку шукав [18]. Використовуються наступні критерії оцінки.

Критерії користувача:

- чуйність / зручність використання (Оновлення даних на сторінках, перемикання між сторінками (час відгуку); насиченість та інтуїтивність);
- пов’язуваність (Можливість зберігати закладки та посилання на різні розділи веб-сайту)
- робота в автономному режимі

Критерії розробника:

- швидкість розвитку (Впровадження нових функцій, рефакторинг, розпаралелювання процесу розробки програмного забезпечення);
- продуктивність (Максимальна швидкість відповіді сервера з мінімальним споживанням обчислювальної потужності);
- масштабованість (Можливість збільшити обчислювальну потужність або простір на диску при збільшенні обсягу інформації та / або

кількості користувачів. Якщо використовується виділена масштабована система, потрібно забезпечити узгодженість даних, доступність та допуск на розділи. випробуваність (Можливість і простота автоматизованого модульного тестування):

Критерії власника програмного продукту:

- функціональна розширюваність (Нова функціональність за мінімальний час та бюджет);
- SEO (Користувачі повинні мати можливість знайти програму через будь-яку пошукову систему);
- підтримка (Окрім власне розробки програмного забезпечення, існують додаткові витрати: обладнання, мережева інфраструктура, обслуговування);
- безпека (Власник програмного забезпечення повинен бути впевнений, що як ділові дані, так і інформація про користувачів захищаються. В якості основного критерію безпеки ми розглянемо можливість зміни функціональних можливостей поведінки додатків на стороні клієнта та всі пов'язані з цим ризики. Стандартні небезпеки однакові для порівняних архітектур. Ми не розглядаємо безпеку на каналі сервер-клієнт, оскільки всі ці архітектури однаково схильні до злому. Цей канал може бути однаковим);
- перетворення: веб-сайт - мобільний або настільний додаток (Перетворення на мобільний або настільний додаток з мінімальними додатковими витратами).

Давайте окреслимо три основні типи вебзастосунків відповідно до ролей, які виконують сервер та клієнтський браузер.

Більшість веб-додатків розробляються шляхом розділення його основних функцій на рівні або рівні. Це дозволяє легко замінити та оновити кожен шар самостійно (рис. 1.4).

Презентаційний рівень доступний користувачам через браузер і складається з компонентів інтерфейсу користувача та компонентів процесів інтерфейсу

користувача, які підтримують взаємодію з системою. Розробники використовують такі фреймворки JavaScript, як Angular та React, щоб зробити вміст сторінки динамічним [25].

Бізнес рівень, який також називають бізнес-логікою, або логікою домену, або рівень додатків, приймає запити користувачів від браузера, обробляє їх і визначає маршрути, через які буде здійснюватися доступ до даних.

Рівень зберігання або доступу до даних, рівень стійкості - це централізоване місце, яке приймає всі дзвінки даних і забезпечує доступ до постійного зберігання програми. Рівень стійкості тісно пов'язаний з бізнес-рівнем, тому логіка знає, з якою базою даних говорити, а процес отримання даних є більш оптимізованим [27].

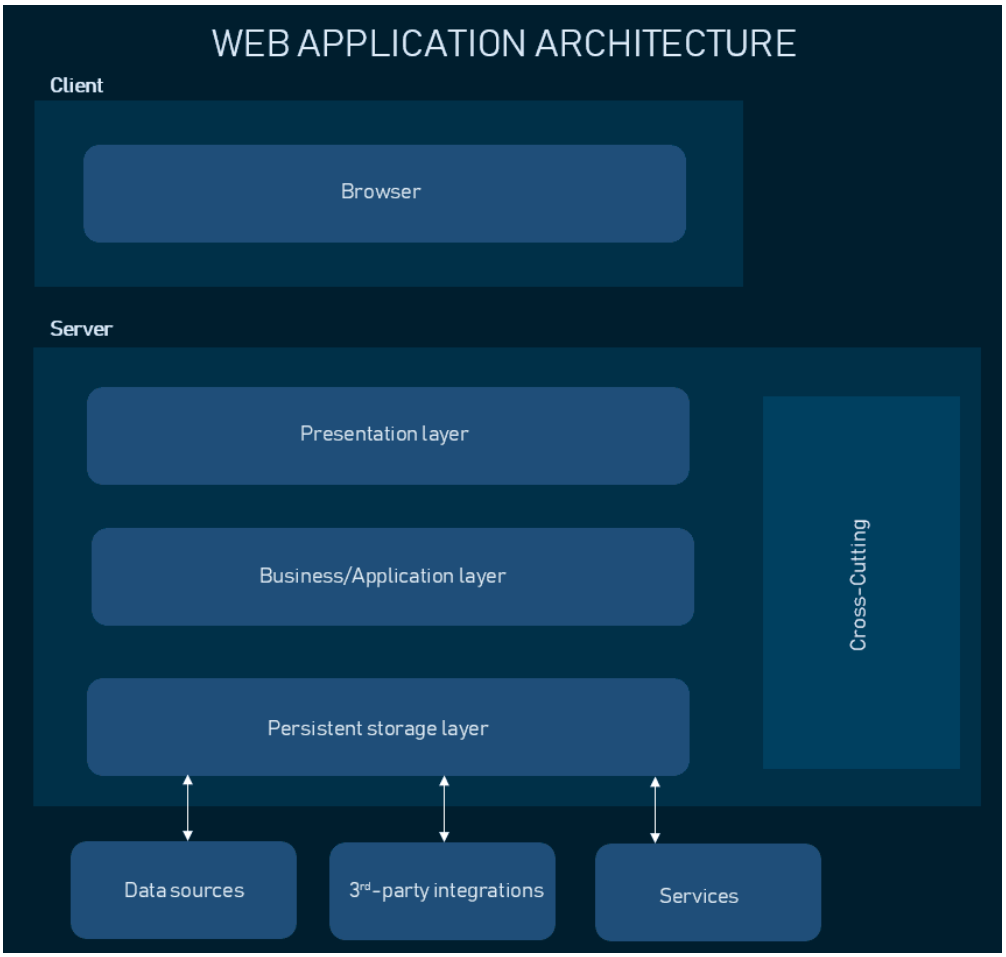


Рисунок 1.4 Архітектура веб-додатків за трирівневим зразком

Інфраструктура зберігання даних включає сервер та систему управління базами даних, програмне забезпечення для зв’язку з самою базою даних, програми та користувальницькі інтерфейси для отримання даних та їх синтаксичного аналізу. Зазвичай ви можете зберігати свої дані або на власних апаратних серверах, або в хмарі - це означає, що ви купуєте послуги управління та обслуговування центру обробки даних, отримуючи доступ до сховища практично. Користуючись послугами постачальників хмарних технологій, таких як Amazon, Google або Microsoft, ви можете використовувати підходи до інфраструктури як послуги, платформи як послуги або безсерверні підходи до управління хмарою [18].

Є також компоненти, які зазвичай існують у всіх вебпрограмах, але відокремлені від основних шарів:

Наскрізний код. Цей компонент вирішує інші проблеми, пов’язані з додатками, такі як зв’язок, управління операцією та безпека. Це впливає на всі частини системи, але ніколи не повинно змішуватися з ними.

Сторонні інтеграції. Платіжні шлюзи, соціальні логіни, GDS на веб-сайтах для подорожей - це все інтеграції, підключені до задньої частини програми за допомогою фрагментів коду, які називаються API. Вони дозволяють вашому програмному забезпеченню отримувати дані з іншого програмного забезпечення та розширювати ваші функціональні можливості, не кодуючи їх з нуля.

Найпоширеніша архітектура вебзастосунків. Сервер генерує HTML-вміст і відправляє його клієнту як повноцінну HTML-сторінку. Іноді цю архітектуру називають "Web 1.0", оскільки вона з'явилася першою і в даний час домінує у сфері веб-розробки.

Чуйність / зручність використання: 1/5. Найменш оптимальне значення серед цих прикладів архітектури. Величезна кількість даних передається між сервером і клієнтом. Неможливо надіслати миттєві оновлення даних або зміни в режимі реального часу. Якщо ми розглянемо можливість оновлення в режимі реального часу за допомогою генерації готових фрагментів вмісту на стороні

сервера та оновлень клієнта (через AJAX, WebSockets), а також дизайну з частковими змінами на сторінці, ми вийдемо за рамки цієї архітектури.

Пов’язуваність: 5/5. Найвища з трьох, оскільки це найпростіша реалізація. Це пов’язано з тим, що за замовчуванням одна URL-адреса отримує певний HTML-вміст на сервері.

SEO: 5/5. Досить легко реалізується, подібно до попереднього критерію. Зміст відомий заздалегідь.

Швидкість розвитку: 5/5. Це найстаріша архітектура веб-розробки, тому можна вибрати будь-яку мову сервера та структуру для певних потреб.

Масштабованість: 4/5. Якщо ми подивимось на генерацію HTML, під зростаючим навантаженням настає момент, коли потрібен баланс навантаження. Ситуація з масштабуванням баз даних набагато складніша, але це завдання є однаковим для цих трьох прикладів архітектури програмного забезпечення.

Продуктивність: 3/5. Тісно пов’язаний з чуйністю та масштабованістю. Продуктивність порівняно низька, оскільки потрібно передати велику кількість даних, що містить HTML, дизайн та бізнес-дані. Тому необхідно генерувати дані для всієї сторінки (не тільки для змінених бізнес-даних), а також усієї супровідної інформації (наприклад, дизайну).

Випробуваність: 4/5. Хороша річ полягає в тому, що немає необхідності в спеціальних інструментах, які підтримують інтерпретацію JavaScript, для тестування інтерфейсу, а вміст статичний.

Безпека: 4/5. Логіка поведінки додатків знаходиться на стороні сервера. Однак дані передаються відкрито, тому може знадобитися захищений канал (що в основному є історією будь-якої архітектури, що стосується сервера). Вся функціональність захисту знаходиться на стороні сервера.

Перетворення: веб-сайт - мобільний або настільний додаток: 0/5. У більшості випадків це просто неможливо. Таким чином, можна обернути додаток у node-webkit або аналогічним способом.

Офлайн робота: 2/5. Реалізовано з маніфестом на сервері, який вводиться до специфікацій HTML5. Якщо браузер підтримує таку специфікацію, усі сторінки програми будуть кешовані: у випадку, якщо з'єднання вимкнено, користувач побачить кешовану сторінку.

Проаналізуємо, як трирівнева архітектура реалізована у різних типах вебдодатків.

Приклад №1. Динамічні веб-сторінки, SPA та MPA [17]

Інтерфейс програми може обслуговувати як статичний, так і динамічний вміст. У більшості випадків це поєднання обох. Поєднання динамічного та статичного вмісту складає веб-додаток. Найпростішим прикладом веб-програми з динамічним вмістом є програма з однією сторінкою.

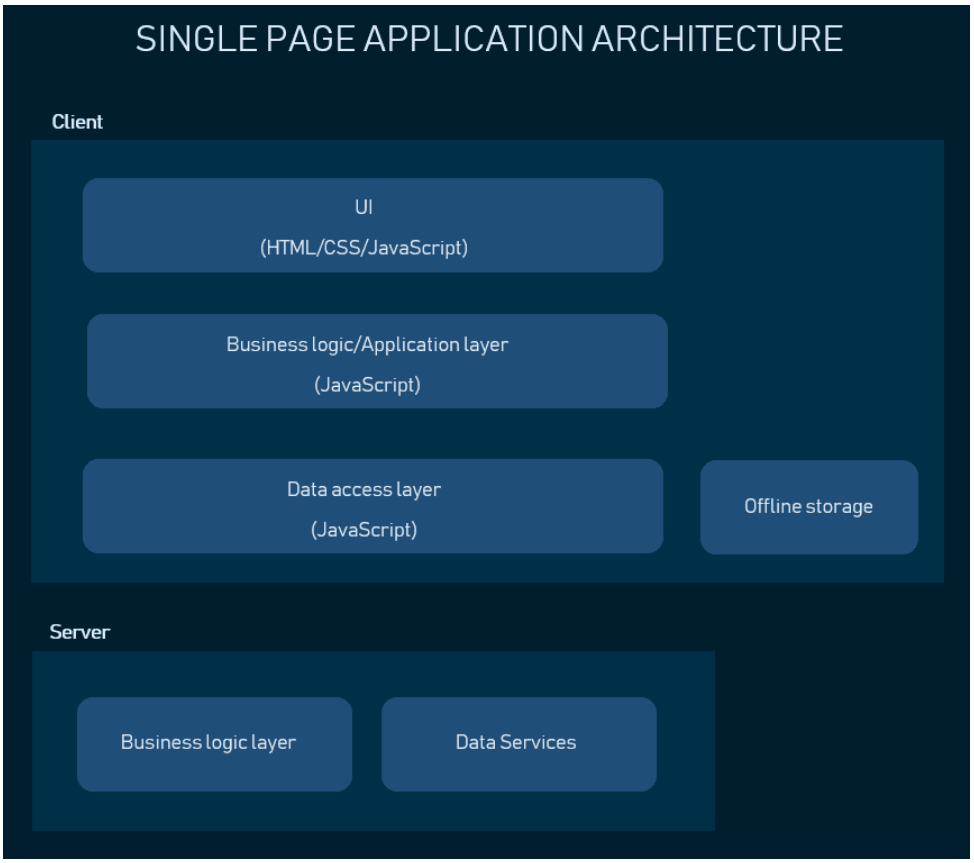


Рисунок 1.5 - Архітектура додатків для однієї сторінки

Основне призначення SPA - це можливість отримати доступ до всієї інформації з однієї HTML-сторінки. Перемістивши логіку програми на сторону

клієнта та використовуючи сторону сервера лише як сховище даних, розробники можуть пришвидшити роботу веб-сайту та полегшити навантаження з сервера. Інтерфейс, окрім HTML та CSS, написаний на єдиному фреймворку, який динамічно генерує вміст і передає його користувачеві (згадайте канал Facebook або ваш Gmail). Залежності між компонентами жорсткі. Це означає, що для внесення змін до одного з елементів UX або UI потрібно переписати весь зовнішній код.

Оскільки SPA переносять логіку на сторону клієнта, їх потрібно писати за допомогою сценаріїв на стороні клієнта. Якщо ви використовуєте технології сценаріїв на стороні клієнта , ви в основному створюєте шаблони, тому, коли користувач запитує вміст, сервер просто передає ці дані назад у браузер, який відображає їх відповідно до шаблонів.

Суть цієї архітектури полягає в тому, що HTML-сторінка завантажується з сервера. Ця сторінка є контейнером для коду JavaScript, який звертається до певної веб-служби та отримує лише ділові дані. Дані використовуються додатком JavaScript, який генерує HTML-вміст сторінки. Ця архітектура є самодостатнім і досить складним додатком JavaScript, де частина функціональних можливостей переходить на сторону клієнта. Для порівняння, архітектура другого типу не може показувати велику кількість взаємопов’язаних та структурованих функцій.

У сучасній веб-розробці програми, що працюють у режимі офлайн, є рідкістю (за невеликими винятками, наприклад, rad-js.com). Цей підхід дозволяє легко здійснити зворотне перетворення: опублікувати існуючу програму в Інтернеті.

Чуйність / зручність використання: 5/5. Обсяг даних, що передаються для оновлень, мінімальний. Тому чуйність на найвищому рівні. Інтерфейс створюється за допомогою JavaScript, можна реалізувати будь-які необхідні варіанти. Є проблема з багатопотоковістю в JavaScript: у цьому конкретному

випадку обробка великих обсягів бізнес-даних повинна бути перенесена на веб-службу.

Пов’язуваність: 1/5. Потрібні спеціальні інструменти та механізми, а також рамки, які можуть використовувати, наприклад, механізм Хеш-Банга.

SEO: 1/5. Найважче просувати. Якщо просування цілої програми здійснюється безпосередньо, немає проблем: можна просувати контейнер програми. Якщо це потрібно для частини програми, для цього буде потрібний спеціальний механізм. Кожна більш-менш велика пошукова система пропонує свої методи стандартизації для цього процесу.

Швидкість розвитку: 2/5. Потрібно розробити веб-сервіс і застосувати більш спеціалізовані фреймворки JavaScript, які будують архітектуру додатків. Оскільки архітектура є відносно новою, не так багато фахівців, здатних створити якісний сайт / систему на основі цього підходу. Існує не так багато перевірених часом інструментів, рамок та підходів.

Продуктивність: 5/5. На цей критерій найменш впливає сторона сервера. Сервер повинен лише передати браузеру програму JavaScript. З боку клієнта, продуктивність та тип браузера мають найбільше значення.

Масштабованість: 5/5. Веб-логіка стоїть на стороні клієнта. На сервері не генерується вміст. Коли кількість користувачів збільшується, потрібно масштабувати лише веб-служби, які надають дані про бізнес.

Випробуваність: 3/5. Це потрібно для тестування веб-служб та коду JavaScript клієнта.

Безпека: 0/5. Логіка перенесена на клієнтський JavaScript, який зловмисник може відносно легко змінити. Для захищених систем потрібно розробити превентивну архітектуру, яка враховує особливості програм з відкритим кодом.

Перетворення: вебсайт - мобільний або настільний додаток: 5/5. Веб-сайт стає додатком за допомогою PhoneGap або подібної платформи.

Офлайн робота: 5/5. Ця архітектура є повноцінним додатком; можна зберігати окремі дані, а також частини програми за допомогою будь-якого сховища (наприклад, локального сховища). Ще однією перевагою є можливість перевести зберігання даних та керування ними в автономний режим. Для порівняння, дві вищезазначені архітектури функціонують лише частково в автономному режимі.

Ми не можемо говорити про SPA, не згадуючи більш традиційну модель - багатосторінкові програми.

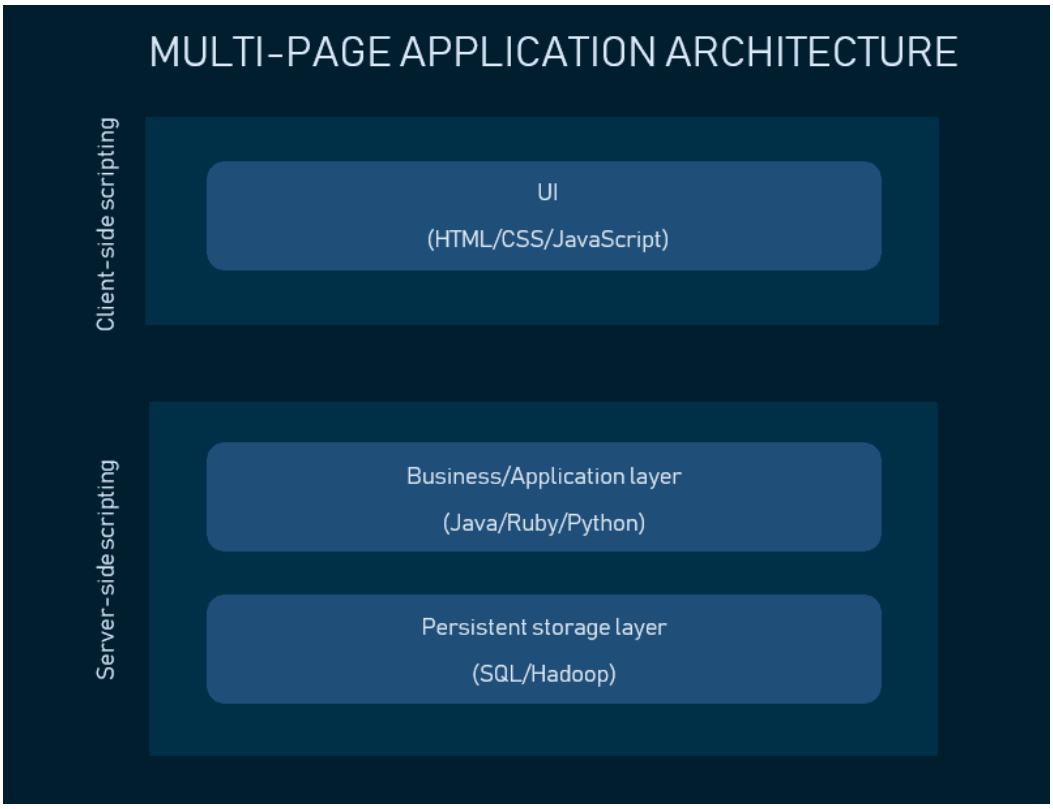


Рисунок 1.6 - Архітектура MPA

У багатосторінкових програмах деякі запити вмісту потребують отримання абсолютно нової веб-сторінки для отримання із сервера [3]. Це масивні програми з багатошаровим інтерфейсом. Технологія AJAX вирішує труднощі складних додатків, які передають величезну кількість даних між сервером та браузером, оновлюючи лише окремі елементи програми. У той же час, даний підхід додає складності таблиці, яку важче розробити порівняно з SPA.

На відміну від сценаріїв на стороні клієнта SPA, традиційні програми пишуться як на мові клієнта, так і на стороні сервера. Серверні сценарії означають, що всі операції виконуються на кінці сервера, тому, коли ви запитуєте вміст, він обробляє сценарій, отримує дані із сховища та обирає вміст для відображення. Мови сценаріїв сервера, з якими ви повинні бути знайомими, включають PHP, Java , Python, Ruby, C # та багато іншого [29].

Найголовніша перевага полягає в тому, що оновлення з сервера надходять лише для тієї частини сторінки, яку вимагає клієнт. Також добре, що віджети розділені функціонально. Певний віджет відповідає за частину сторінки; часткові зміни не вплинуть на всю сторінку.

Чуйність / зручність використання: 3/5.Обсяг переданих даних для частини сторінки менший, ніж для цілої сторінки, тому чуйність вища. Але оскільки сторінка є набором віджетів, застосовні шаблони інтерфейсу користувача у веб-програмі обмежені вибраною структурою інтерфейсу. Холодний запуск (перше повне завантаження) такої сторінки займе трохи більше часу. Вміст, який повністю генерується та кешований на сервері, може негайно відображатися на клієнті; тут витрачається час на отримання даних для віджету і, як правило, на шаблонування. Під час першого відвідування веб-сайт завантажуватиметься не так швидко, але надалі він буде набагато приємнішим у використанні, якщо порівнювати з сайтами, заснованими на архітектурі першого типу.

Пов’язуваність: 2/5. Тут потрібні спеціальні інструменти та механізми. Як правило, застосовується механізм Хеш-Банга.

SEO: 2/5. Для цих завдань існують спеціальні механізми. Швидкість розвитку: 3/5. Потрібно знати серверні технології веб-розробки та використовувати фреймворки JavaScript на стороні клієнта. Це також потрібно для реалізації веб-служб на стороні сервера.

Продуктивність: 4/5. Час і ресурси, витрачені на створення вмісту HTML, порівняно незначні, якщо порівнювати з часом, витраченим

програмою на отримання даних із баз даних та на їх обробку перед створенням шаблону.

Масштабованість: 4/5. Те саме, що і для архітектури першого типу.

Випробуваність: 1/5. Потрібно протестувати сторону сервера, код клієнта та веб-службу, яка повертає дані для оновлення віджетів.

Безпека: 4/5. Частина логіки переноситься на клієнтський JavaScript, який може бути змінений зловмисником.

Перетворення: веб-сайт - мобільний або настільний додаток: 0/5. Те саме, що і для архітектури першого типу.

Офлайн робота: 1/5. У цьому випадку механізм маніфесту працює, але існує проблема з оновленням або кешуванням даних, що відображаються на віджеті.

Приклад №2. Корпоративні програми.

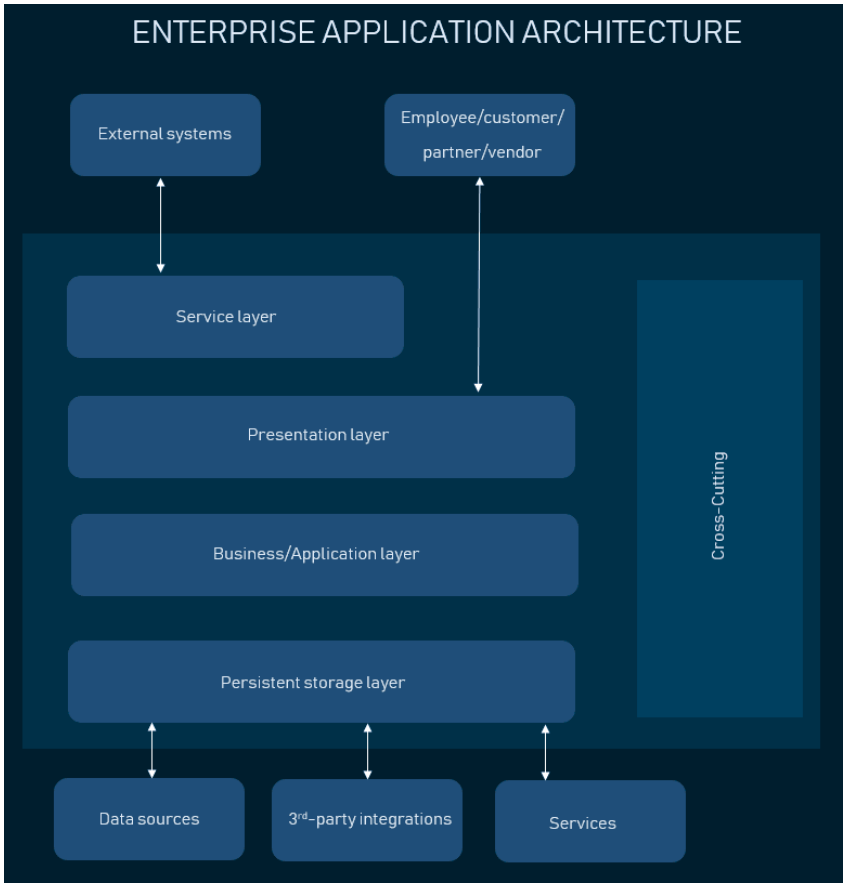


Рисунок 1.7 - Архітектура корпоративних додатків

Корпоративний додаток - це програмне забезпечення, яке легко настроюється, розроблене спеціально для потреб конкретної організації [33]. Зазвичай він має кілька інструментів, орієнтованих на бізнес, інтегрованих під єдиним інтерфейсом. Корпоративні системи також безпосередньо підключаються до існуючого робочого процесу компанії. Вони надійні, надають доступ багатьом користувачам одночасно та обмінюються інтерфейсами з різними іншими інструментами підприємства. Якщо ви хочете дізнатися більше про процес розробки корпоративної архітектури , ознайомтесь зі статтею.

Дві основні відмінності архітектури корпоративних додатків від звичайної веб-програми - це додавання іншого рівня до класичного зразка - рівня обслуговування.

Рівень послуг є ще однією абстракції між презентації та бізнес - логіки. Це інтеграційний шлюз, який дозволяє іншому програмному забезпеченню отримувати доступ до вашої бізнес-логіки та ресурсів, не взаємодіючи безпосередньо з цими ресурсами. Він працює, передаючи повідомлення через окремий інтерфейс і працює як API.

Окрім додаткового рівня, корпоративні програми мають доступ до джерел даних з інших програм в організації, що робить її мережею програмних рішень, пов'язаних API. Крім того, існує більше груп користувачів, які мають доступ до різних функціональних компонентів - вони можуть бути діловими партнерами, клієнтами, адміністраторами та кількома групами персоналу. Іноді рівні презентації окремі для всіх, тому ви можете розгорнути програму як інтрамережу або екстрамережу.

1.3 Принципи розробки вебдодатків

Сьогодні все більше і більше додатків переходять в Інтернет. Без обмежень платформи та вимог до встановлення модель програмного забезпечення як послуги виглядає дуже привабливо. Дизайн інтерфейсу вебзастосунків за своєю

суттю - веб-дизайн; однак основна увага приділяється функції. Щоб конкурувати з настільними програмами, веб-програми повинні пропонувати прості, інтуїтивно зрозумілі та чуйні користувальницькі інтерфейси, які дозволять своїм користувачам виконувати справи з меншими зусиллями та часом.

Наступні рекомендації ґрунтуються на методології The Twelve-Factor App [28].

Обговорені тут найкращі практики розробки вебзастосунків можна застосувати до будь-якої моделі програмного забезпечення як послуги (SaaS). Він охоплює все: від внутрішньої розробки на локальній машині до хмарної архітектури [34].

SaaS - це модель надання ліцензії на програмне забезпечення за передплатою. SaaS (software as a service) - програмне забезпечення як послуга, також зустрічається переклад «інформація як сервіс». SaaS найчастіше - це хмарне рішення.

Користувач отримує доступ до сервісу через браузер або по API. При цьому його підтримкою цілком займається постачальник послуги. Спрощено кажучи, модель SaaS це коли клієнт працює з готовим рішенням онлайн. Оплачує доступ і максимально швидко отримує на руки готовий інструмент. Саме за такою моделлю працює наш сервіс коллтрекінга .

Найпростіший приклад SaaS - це Google Docs, безкоштовний сервіс для роботи з документами. Ніяких носіїв, драйверів і установок. Заводите аккаунт Google, переходите по посиланню і працюєте з текстами, таблицями та презентаціями прямо в браузері. Причому в документах одночасно з вами можуть працювати і інші колеги. Виїхавши у відрядження, можна зайти в свій аккаунт з будь-якого пристрою і продовжити друкувати потрібний документ. При цьому зберігати потрібно тільки змінити установки, інші дані зберігаються автоматично.

По суті, SaaS-сервіс це єдине програмне ядро, яке надається в користування клієнтам. Доступ до системи вони отримують через мережу і можуть змінювати

налаштування на свій розсуд. Обслуговуванням сервісу цілком займається провайдер послуги, а користувач тільки працює в ній.

Популярність цієї моделі щорічно зростає. Тільки в минулому році ринок SaaS збільшився на 21,7%. За прогнозами експертів, така тенденція збережеться і в найближчі роки. Це й не дивно, адже так користувачі отримують в розпорядження сучасні технології практично без зусиль зі свого боку.

При використанні SaaS у вас на руках не буде нічого, що можна «помацати»: настановних файлів, папок і документів на комп'ютері. Всі дані розміщуються на серверах постачальника послуги - в так званому «хмарному» сховище в мережі. Ви ж отримуєте віддалений доступ до програми і можете працювати з нею в браузері з будь-якого місця на планеті, де є інтернет. У той же час, існують SaaS, які можна купити і встановити до себе на сервер.

Схеми оплати за SaaS досить різноманітні. Спочатку розглянемо найбільш популярну. У цьому випадку користувач не набуває дорогу ліцензію ПО цілком, як при купівлі диска або при скачуванні настановних файлів. І не оплачує значну вартість розробки, як при замовленні індивідуального рішення. Він ніби бере сервіс в оренду, оплачуючи аналог абонплати за доступ до функціонала на місяць або довше.

Головний плюс SaaS в тому, що ця модель дозволяє зосередитися на роботі, не відволікаючись на завдання, пов'язані з софтом. У випадку з традиційним ПО неполадки доводиться усувати своїми силами або запрошувати системного адміністратора. Оновлення необхідно купувати на новому диску, знову встановлювати, вирішувати проблеми з настройками і т. д.

У SaaS все інакше. Постачальник відповідає за настройку сервісу зі свого боку. Він же несе відповідальність за збереження даних, коректну роботу ПО і випуск оновлень. Техпідтримка сервісу вирішує будь-які питання і часто надає користувачам матеріали, навчальні роботі з рішенням.

Переваги SaaS

1. У стандартній моделі відносини покупця і постачальника найчастіше закінчуються відразу після покупки ПО. А далі, як то кажуть, крутитесь, як вмієте. Винятки становлять хіба що «гіганти» рівня Windows і Adobe. Коли ж мова йде про SaaS, користувач в будь-який момент може звернутися до постачальника за підтримкою. Всім клієнтам сервісу доступні регулярні оновлення.

2. Для великої компанії це економія на ПО. Програму не потрібно купувати і окремо встановлювати на кожен комп'ютер, щоб в ній могло працювати кілька користувачів. Головне мати доступ до інтернету, а також логін і пароль.

3. Деякі сервіси просто неможливо реалізувати на власному сервері компанії. Наприклад, продукти аналогічні Serpstat або Datanyze . Вони не можуть існувати в якості традиційного ПО, а тільки як SaaS. Таке рішення при великому бажанні можна розробити самостійно. Але для цього потрібні значні ресурси.

З іншого боку, для власника SaaS така модель - джерело регулярного доходу і потенціал для зростання. Користувачі оплачують сервіс не одноразово, а постійно, і це дозволяє розробити довгострокову стратегію розвитку.

Вони також базуються на внесках ширшої спільноти інженерів програмного забезпечення, які мають значний досвід у веб-розробці підприємств.

Контроль доступу на основі атрибутів mobidev. Вибір технології: фреймворк Symfony PHP.

Цей шлях до найкращих практик розробки вебзастосунків може бути використаний як посібник для проектування внутрішньої архітектури тощо. Ви також можете зробити це, використовуючи будь-яку мову програмування, яка вам подобається.

Однак у цій публікації ми зупинимось на прикладах, які використовують фреймворк Symfony PHP [16] . Це цілком логічно, оскільки цей фреймворк часто використовується для створення середніх хмарних продуктів SaaS.

Сьогодні система відповідає рекомендаціям PHP Standards і може похвалитися принципами твердого дизайну для веброзробки. Тож інженери

програмного забезпечення зможуть безперешкодно дотримуватися цих найкращих практик [17].

1. Одна кодова база може вмістити кілька розгортань протягом циклу веб-розробки.

Якщо ви розробляєте веб-додаток, а не розподілене програмне забезпечення, настійно рекомендую дотримуватися однієї кодової бази. Найкращий спосіб це зробити - зберегти свій код у сховищі Git (або будь-якому іншому контрольованому версією рішення).

Цикл розробки вебзастосунків для нових функцій буде виглядати (більш-менш) так:

- Інженери-програмісти працюють з однаковою базою кодів на своїх локальних комп'ютерах.
- Після того, як кожна функція розроблена, вони надсилатимуть код до сховища Git.
- Потім оновлена кодова база розгортається та тестується на проміжному сервері командою забезпечення якості.
- Потім той самий код розгортається у виробництві, але лише після того, як його перевірять на проміжному рівні, щоб переконатися, що він працює належним чином.

2. Явне оголошення та ізоляція залежностей для програмного забезпечення.

Золотим правилом є спочатку скласти список усіх зовнішніх бібліотек та служб, які використовувались. Вони повинні бути окремими від коду і швидко вмикатись / вимикатись або змінювати за необхідності. Це дозволяє зробити більше за мінімальний вплив на загальний код.

Наприклад, для управління бібліотеками використовуйте Composer . Це чудовий спосіб додавати та керувати бібліотеками, які вимагає ваш веб-додаток.

Зберігайте composer.lock у своєму сховищі Git і виключайте папку постачальника з Git. Цей підхід гарантуватиме, що кожне розгортання (для

іншого розробника, для проміжних або виробничих серверів) отримає однакові версії бібліотек після команди запуску “composer install”.

3. Окремі файли конфігурації та база програмного коду.

Згідно з принципами програми Дванадцять факторів, «конфігурація програми - це все, що, можливо, відрізнятиметься від розгортань».

Часто параметри конфігурації програмного забезпечення можна змінювати між середовищем розробки, інтерактивним та виробничим середовищем. Однак, згідно з принципами розробки додатків, зберігати їх у кодовій базі як конфігури або константи заборонено.

Нижче наведено приклади налаштувань, які не слід зберігати в коді:

- Налаштування підключення до бази даних та кешу
- Повноваження сторонніх служб
- API та платіжні шлюзи
- Все, що змінюється залежно від стадії розгортання

Як правило, щось завжди ламається через зміни, внесені в конфігурацію в коді. Однак веброзробка Symfony використовує бібліотеку DotEnv для роботи зі змінними середовища. Ось приклад різних змінних середовища для розробки, постановки та виробництва.

Development Config	Staging Config	Production Config
APP_ENV=dev APP_DEBUG=true APP_DEFAULT_LOCALE=en DATABASE_URL=”postgresq://db user:ohzahquae9raiJoh@postgres: 5432/appdb” GOOGLE_API_KEY=GOOGLE-API -KEY-FOR-DEV	APP_ENV=prod APP_DEBUG=true APP_DEFAULT_LOCALE=en DATABASE_URL="postgresql:// dbuser:Pu7lan6feiNaecha@host. yyyy.us-east-1.rds.amazonaws. com:5432/appdb GOOGLE_API_KEY=GOOGLE-API-KEY-FOR-STAGING	APP_ENV=prod APP_DEBUG=false APP_DEFAULT_LOCALE=en DATABASE_URL="postgresql:// dbuser:ohzahquae9raiJoh@host. xxxx.us-east-1.rds.amazonaws. com:5432/appdb" GOOGLE_API_KEY=GOOGLE-API -KEY-FOR-PROD

mobidev

Рисунок 1.8 - Змінні середовища для розробки

Тож не має значення, де розгорнуто вебдодаток, оскільки не потрібно буде змінювати код. Потрібно встановлювати лише належні змінні середовища. Якщо ви використовуєте Docker , запустіть різні контейнери Docker з одного зображення з різними змінними середовища для різних розгортань [10].

4. Усі різноманітні послуги, які використовує ваш веб-додаток, повинні залишатися відокремленими

Немає значення, чи використовує додаток якісь сторонні внутрішні служби, API або вбудовані компоненти, такі як база даних MySQL. Найкращою практикою за цим сценарієм буде розглядати їх усіх як приєднані ресурси.

Увімкніть доступ до програмного забезпечення через URL-адресу або скористайтеся будь-яким іншим методом, що зберігається у конфігурації. Цей підхід дозволить команді розробників управляти компонентами, не вносячи змін до коду.

Розробка Symfony полягає у дотриманні принципів SOLID. Наприклад, щоб змінити сховище бази даних із MySQL, розміщеного на екземплярі EC2, на Amazon RDS, вам просто потрібно змінити URL-адресу стилю DSN у конфігурації середовища, не вносячи жодних змін у код.

Ви навіть можете змінити статичне сховище з Amazon S3 на будь-яке інше хмарне сховище, просто змінивши змінні середовища, не змінюючи код.

Деякі бібліотеки підтримують різні типи сховищ з реалізованим шаром абстракції. Наприклад , Flysystem підтримує різні типи хмарних сховищ. Доктрина - ще один приклад, але вона абстрагована навколо сховища бази даних. Таким чином, ви можете перейти з MySQL на PostgreSQL після внесення незначних змін у код [37].

5. Створення, випуск і запуск кожного етапу розробки програмного забезпечення окремо.

Щоб забезпечити стабільність на всіх етапах збірки, ви можете використовувати інструменти автоматизації, такі як Gitlab CI та Jenkins [12] .

На етапі збірки - код повинен бути встановлений, залежно від бібліотек PHP та ресурсів CSS / JS, які потрібно підготувати. Нарешті, контейнер Docker з тегом версії повинен бути побудований і введений до реєстру Docker. Ви також матимете можливість виконувати модульні тести та sniffer коду.

На етапі випуску - поєднайте контейнер Docker (створений на етапі збірки) з конфігурацією для середовища (проміжне та / або виробниче), де ви будете запускати свою збірку.

На етапі запуску - запустіть програму у вибраному середовищі з відповідною конфігурацією середовища. Це буде базуватися на етапі випуску.

6. Зробіть процеси без громадянства та зберігайте дані поза веб-програмою.

Провідні практики веб-розробки рекомендують зберігати постійні дані в окремих службах. Це може бути будь-яка реляційна база даних, наприклад, як Redis, Amazon S3 тощо.

Однак, якщо ви працюєте з Docker, вам не потрібно зберігати всі дані всередині контейнера. Це пов’язано з тим, що ваш додаток має бути без громадянства

У майбутньому такий підхід буде критично важливим для масштабованості. Якщо вам потрібна авторизація у вашому API, ви можете використовувати веб-токени JSON без громадянства або сесії, але зберігати їх у Redis [25].

Структура Symfony може працювати в обидві сторони, і такі методи дозволяють отримати переваги від масштабованості. Наприклад, ви можете запустити один контейнер або сотні контейнерів, які використовують той самий код і працюють з одними і тими ж службами.

Ви можете перезапустити свій кластер (або купу контейнерів) або запустити новий із новою версією без втрати даних.

У певний момент протягом циклу зростання вам знадобиться більше одного сервера. На цьому етапі програмне забезпечення почне працювати на кількох серверах за допомогою балансира навантаження.

Це найкращий підхід до організації виконання функцій та різних запитів. Це саме той момент, коли всі процеси повинні бути без громадянства.

Користувачі будуть авторизовані на одному сервері, але їх запити будуть спрямовані на інший сервер, який буде виконано. Якщо служба авторизації не має без громадянства (і не може підтримувати обидва сервери), користувач не зможе продовжувати запускати програму через відсутність авторизації на другому сервері.

7. Зберігайте програмне забезпечення самостійним та експортуйте послуги за допомогою прив’язки портів

Усі вебпрограми мають бути доступними без залежностей. Якщо програма використовує кілька служб, кожна служба повинна бути доступною через окремі порти.

Наприклад, після запуску Docker Compose на локальній машині розробника додаток стане доступним за адресою http: // localhost: 8085. У той же час окремий графічний інтерфейс для служб управління базами даних буде доступний на http: // localhost: 8084.

Такий підхід дозволяє нам запускати багато контейнерів Docker та використовувати балансування навантаження (наприклад, HAProxy та Amazon Elastic Load Balancing.)

Якщо команда розробників відмовиться слідувати такому підходу, це ускладнить процес запуску. Усі непотрібні (додаткові) кроки можуть створити додаткові помилки в системі.

8. Застосовуйте модель процесу без розподілу.

Коли вебдодаток розробляється, у програмі буде виконуватися безліч процесів. Однак важливо тримати всі ці процеси окремо. Ви навіть можете

розділити ці процеси на окремі, якщо деякі з них стають «занадто важкими» для управління.

Тож між різними процесами нічого не слід ділити. Це найкращий спосіб зробити програму простою та стабільною. Чудовим у цьому підході до архітектури вебзастосунків є той факт, що він полегшує масштабування:

- Додаток може запускати стільки контейнерів PHP, скільки потрібно для масштабування (вгору або вниз).
- Це може працювати для стільки клієнтів, скільки потрібно.
- Якщо нам потрібно обробити більше запитів HTTP, ми можемо просто додати більше запущених контейнерів PHP.
- Якщо нам потрібно обробити більше фонових завдань, ми можемо додати більше контейнерів із фоновими працівниками, щоб вирішити ці потреби.

9. Залишайтеся стабільними завдяки швидкому запуску та вимкненню процесів

Усі вебпрограми повинні бути розроблені таким чином, щоб забезпечити масштабування (вгору або вниз) за бажанням. Він повинен мати змогу запускати 100 контейнерів, наприклад, у робочий час (коли користувачі програми найбільш активні), і запускати десять контейнерів вночі (для економії грошей, виділених на інфраструктуру).

Цей підхід можна застосувати до вашого програмного продукту, містячи процеси окремо. Однак ви повинні дозволити їх швидке запуск або припинення.

Ось чому мінімізація часу запуску процесів є обов’язковою. Якщо ви використовуєте Docker, ви можете запустити більше PHP-контейнерів.

Зображення контейнера повинно бути готовим до запуску. Не компілюйте під час запуску та не запускайте PHP Composer під час запуску. Час для запуску завжди повинен бути якомога коротшим.

Коли ви хочете зменшити масштаб, все, що вам потрібно зробити, це закрити деякі контейнери. Також критично важливо закрити контейнери, не порушуючи роботу системи.

Користувачі не хочуть бачити час очікування запиту або повідомлення про помилки. Тому використовуйте належні сигнали завершення, надсилаючи демони, як на більшості серверів черг.

RabbitMQ та Beanstalkd є чудовими прикладами, які ефективно працюють у фоновому режимі та виставляють завдання на чергу у випадку відмови. Вам також слід застосовувати транзакції до Реляційної бази даних [17].

10. Зберігайте етапи розробки, постановки та виробництва якомога подібнішими

Середовища розробки, постановки та виробництва повинні бути якомога подібнішими. Дотримання цього правила допоможе командам розробників вебзастосунків уникати таких ситуацій, як збій виробничого програмного забезпечення (що часто трапляється, коли веб-додаток розроблявся за інших умов).

Є деякі випадки, коли розробник може запустити додаток на локальній машині з Docker і використовувати ті самі контейнери Docker для індексування та виробництва. Щоб використовувати щось, що працює лише в хмарі, краще скористатися шаблоном адаптера.

Наприклад, виробничий кластер Docker використовує ті самі контейнери, які були протестовані на етапі, але з різними змінними середовища. Версії PHP і PostgreSQL будуть однаковими (і без змін), але остання працюватиме на Amazon RDS [32].

Рівень журналу відрізняється, оскільки користувачі не бачать подробиць про помилки, подібні до тих, що спостерігаються на машині розробника або на проміжному сервері. У будь-якому випадку, ми все одно отримуватимемо звіти про помилки (з подробицями помилок) на нашому сервері агрегатора

журналів помилок. Таким чином, ви дізнаєтесь про помилку, перш ніж користувачі повідомлять про неї.

11. Передовою практикою збору журналів програм є сприйняття їх як потоків подій

Інженери-програмісти повинні підходити до журналів як до потоків подій, а не до постійного потоку даних, що спрямовується на зберігання. Це оптимальний спосіб досягти високої видимості під час запуску програми.

Зберігання журналів як окремих файлів для кожного контейнера Docker (або окремих служб) не повинно бути доступним для вашої команди.

За замовчуванням додаток Symfony PHP записує журнали у файли журналів, але ви можете налаштувати вихід журналів у потік stdout / stderr Linux. Це можна зробити, встановивши рядок “php: // stderr” у шляху журналу.

Після цього ви зможете побачити всі активні журнали з консолі. У будь-якому випадку, такий підхід буде корисним для розвитку, але недостатньо для виробництва. Ви можете бачити журнали, поки ваш контейнер Docker працює, але не після того, як контейнер буде припинено.

Для виробництва потрібно використовувати служби агрегації журналів. Чудові приклади - Sentry , Graylog , Logstash або Splunk [23] .

Такі служби можуть об'єднувати журнали з усіх запущених контейнерів і дозволяти аналіз з централізованої платформи. Ви також можете використовувати сповіщення, щоб отримувати інформацію про виробничі проблеми та уникати втрати журналів даних після аварії.

12. Управління діяльністю адміністратора як одноразового процесу

Діяльність з адміністрування та управління відіграє важливу роль у розробці та впровадженні програмних продуктів.

Міграції баз даних, операції з кешем, створення нових користувачів системи та резервні копії - це лише деякі приклади. Всі ці принципи веб-розробки повинні застосовуватися і до цих видів діяльності.

Symfony надає команду bin / console для обробки завдань адміністратора / управління. Найкраще запускати такі команди як разовий процес в тому ж ідентичному середовищі, що і звичайні тривалі процеси програми. Це означає, що нам потрібно запустити новий контейнер лише для одного завдання, а не використовувати існуючий контейнер.

Ми використовуємо Docker run, а не Docker execute . Ця команда працює з уже запущеним контейнером. Після завершення міграції з контейнера Docker можна вийти. Це буде одноразовий процес.

Swift - це новий, принциповий підхід до створення веб-додатків, які надійно захищені конструкцією [33]. У сучасних веб-додатках деякі функції додатків зазвичай реалізуються як клієнтський код, написаний на JavaScript. Переміщення коду та даних до клієнта може створити вразливі місця безпеки, але в даний час не існує хороших методів вирішення, коли це безпечно робити. Swift автоматично розділяє код програми, забезпечуючи при цьому надійність та ефективність розміщення. Код програми пишеться як Java-подібний код, анотований політиками потоку інформації, що визначають конфіденційність та цілісність інформації веб-додатків. Усі вищезазначені принципи розробки вебзастосунків повинні виступати базовим для команди розробників програмного забезпечення. Такі впровадження також слід робити розумно та узгоджувати з пріоритетами бізнесу.

Вони повинні бути технічно обґрунтованими та легко застосовуватися до конкретних випадків використання. Якщо ви вирішите ігнорувати одну або кілька з цих найкращих практик, для цього має бути чудова причина.

1.4 Постановка задачі

Розвиток вебтехнологій дає кращі можливості для побудови інтерактивного інтерфейсу, сучасних середовищ для швидкого розвитку вебзастосунків,

можливої взаємодії з існуючими системами та сервісами. Звичайно, якою б досконалою не була технологія, вона сама не здатна вирішити 100% проблем, і Інтернет не є винятком. Але якщо вебрішення задовольняє вимогам проєкту, то робота з цією технологією не розчарує користувача.

У першому розділі розглядаються основні теоретичні положення створення мобільних веб-додатків, сучасні підходи до даної проблеми, наводиться склад і зміст технологічних операцій проєктування на різних рівнях ієрархії управління об’єктом, наявні засоби проектування, методи формалізації процесу проектування та методи управління проектуванням ІС. Проведено аналітичний огляд сучасного стану методів і засобів проектування веборієнтованих інформаційних систем. Показано, що задача забезпечення інтероперабельності веборієнтованих систем і їх модулів є актуальною на даному етапі і в цьому напрямку працюють провідні вчені світу.

Для подальшого дослідження була сформульована мета магістерської роботи, яка полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу.

Виходячи з вищезазначеного матеріалу, виникає питання: які технології та яка архітектура інформаційних вебсистем будуть відповідати таким умовам: простота обслуговування, використання, поширеність. Це визначає мету, завдання та актуальність цього питання.

Для досягнення поставленої мети потрібно вирішити наступні завдання:

1. Провести аналіз існуючих підходів до синтезу та розробки веборієнтованих систем.
2. Визначити можливості підвищення ефективності структурного синтезу, взаємодії вебзастосунків і їх зберігання.
3. Проектування семантичної моделі процесів і розробки вебзастосунків.
4. Розробити ефективну архітектуру та алгоритм функціонування вебзастосунків.

5. Розробити метод об'єктно-реляційного перетворення для синтезу та розробки вебзастосунків, а також шаблону проектування на його основі.
6. Реалізація розроблених алгоритмів структурного синтезу вебзастосунків на базі запропонованого методу.

2 МОДЕЛЮВАННЯ ПРОЦЕСУ ПРОЄКТУВАННЯ ТА СТВОРЕННЯ ВЕБДОДАТКУ

2.1. Модель запропонованої архітектури

Model-View-Controller (MVC) є архітектурним шаблон, який відокремлює додаток на три основні логічні компоненти: модель, вид і контролер. Кожен із цих компонентів створений для обробки конкретних аспектів розробки програми. MVC - одна з найбільш часто використовуваних галузевих стандартів веб-розробки для створення масштабованих та розширюваних проектів (рис. 2.1).

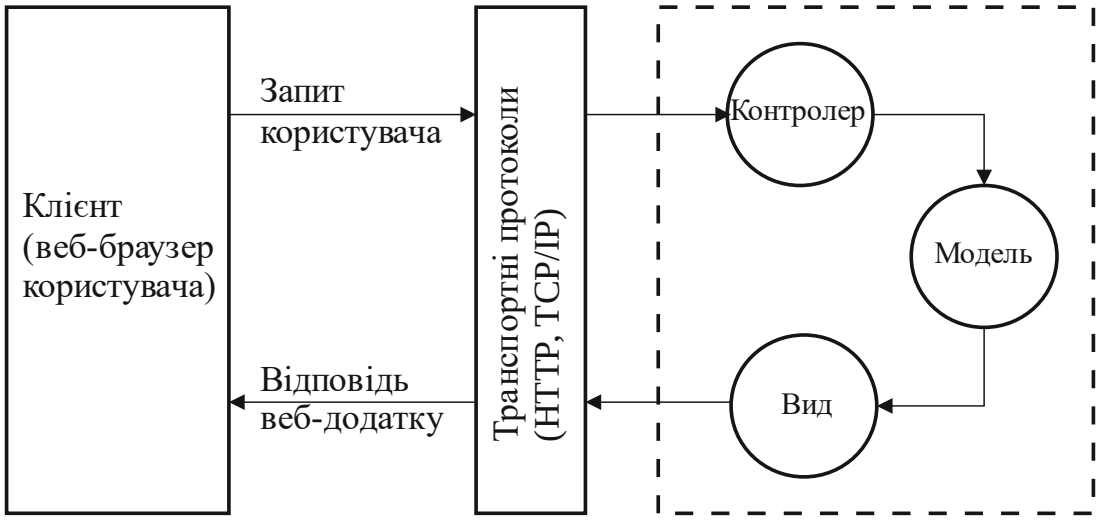


Рисунок 2.1 - Шаблон MVC

Архітектура MVC - це програмний архітектурний шаблон, в якому логіка програми розділена на три компоненти на основі функціональних можливостей. Ці компоненти називаються:

- Моделі - представляють, як дані зберігаються в базі даних
- Представлення - компоненти, видимі користувачеві, такі як вихідні дані або графічний інтерфейс
- Контролери - компоненти, які виконують роль інтерфейсу між моделями та видами

Компонент Model відповідає всій логіці даних, з якою працює користувач. Це може представляти або дані, що передаються між компонентами View та Controller, або будь-які інші дані, пов’язані з бізнес-логікою. Наприклад, об'єкт "Клієнт" буде отримувати інформацію про клієнта з бази даних, маніпулювати нею та оновлювати дані назад у базу даних або використовувати її для рендерингу даних.

Компонент View використовується для всієї логіки інтерфейсу програми. Наприклад, подання Клієнт включатиме всі компоненти інтерфейсу, такі як текстові поля, випадаючі меню тощо, з якими взаємодіє кінцевий користувач.

Контролери виступають як інтерфейс між компонентами Model і View для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою компонента Model та взаємодії з поданнями для надання кінцевого результату. Наприклад, контролер замовника буде обробляти всі взаємодії та входи з перегляду замовника та оновлювати базу даних за допомогою моделі замовника. Той самий контролер буде використовуватися для перегляду даних клієнта.

При даному підході розроблений програмний код розділяється на три непересічних множини: код етапу бізнес-логіки предметної області (рівень контролерів *K*), коду етапу роботи з даними (рівень моделі – *M*), код етапу зовнішнього представлення (рівень видів – *B*).

У даному магістерському досліджені пропонується модифікована модель архітектури MVC (рис. 2.2) і методи розробки, що застосовуються разом з нею. Запрпонована модель заснована на архітектурі вебзастосунку з двоєроковим завантаженням [26]. Відмінністю від оригінальної архітектури MVC, у запропонованому підході розширено всі три рівні архітектури (рис. 2.1).

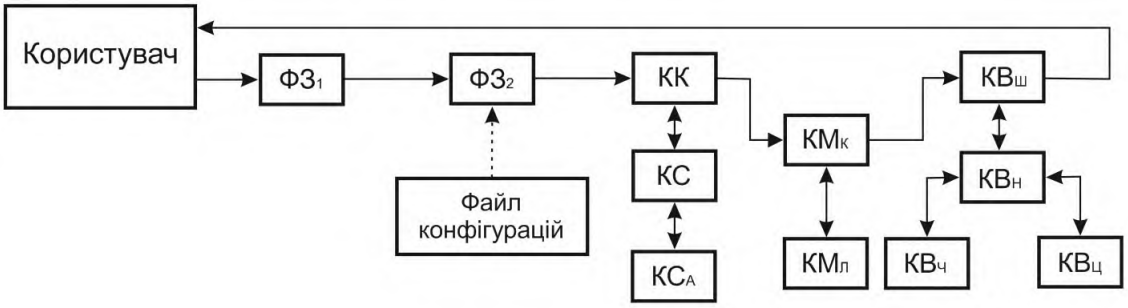


Рисунок 2.2 – Модифікована модель архітектури вебзастосунку

Рівень контролерів розбивається на три рівні: власне *К* (в якому залишаються «тонкі» контролери), *С* (сервіси) і C_A-рівень (сервіс-адаптери).

Сенс терміну "тонкий контролер" в тому, що, як правило, в ваших класах контролера ви розміщуєте тільки код, який [21]:

- звертається до даних запиту користувача($_GET, $_POST, $_FILES та інші PHP-змінні);
- перевіряє правильність вхідних даних;
- (Опціонально) робить базову підготовку даних;
- передає дані моделі (ям) і витягує результат, що повертається моделлю (ями);
- і нарешті, повертає вихідні дані як частина контейнера ViewModel.

Класу контролера не слід:

- містити складну бізнес-логіку, яку краще зберігати в класах моделей;
- містити будь-яку HTML-розмітку, яку краще зберігати в шаблонах уявлення.

Зробимо декомпозицію рівня моделі на концептуальну ($М_К$) і логічну модель ($М_Л$).

Сутності предметної області у вебзастосунку пропонується зображати спеціальними класами, які називаються «моделями». Їхня множина реалізує шар доступу до даних. До класичного підходу MVC додається сервісний шар.

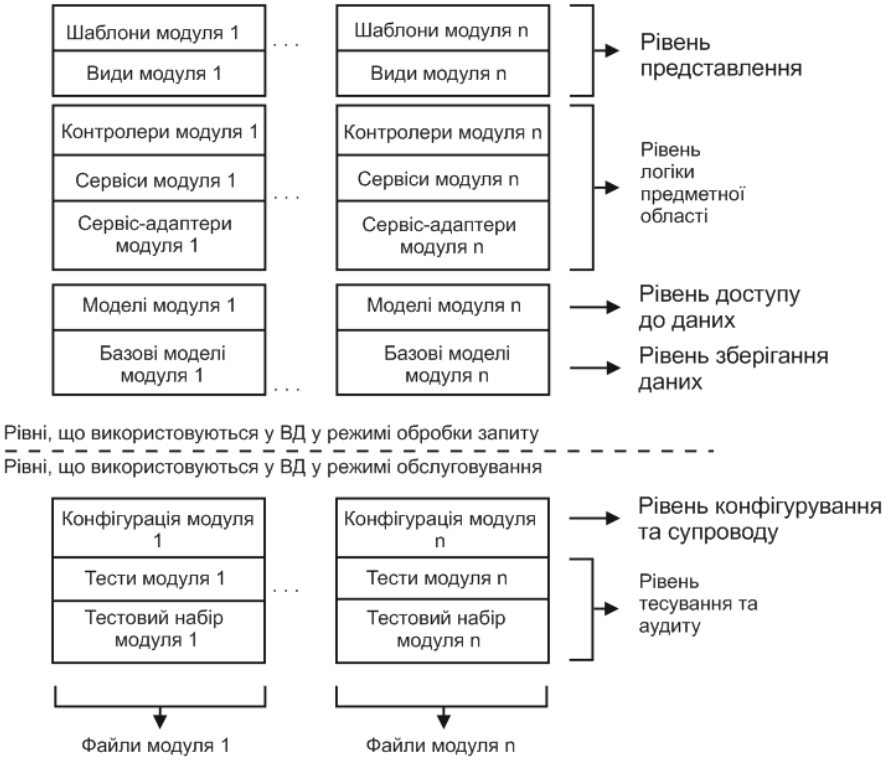


Рисунок 2.3 – Рівні абстракції в розробленій моделі

Пропонується поділити рівні на виконувані, і невиконувані, які використовуються для службових потреб (рис. 2.3).

На рівні зберігання даних відображаються сутності, з якими працює програма («базові моделі даних»).

Рівень доступу до даних містить функції роботи з екземпляром моделей які успадковуються від базових.

Бізнес-логіка складається із множини класів, (контролерів), функції яких називаються діями.

Сервісні функції, такі як взаємодія з групою моделей, пошук, модифікація знаходяться на рівні сервісів. Для реалізації поведінкового шаблону проектування, призначеного для визначення сімейства алгоритмів, інкапсуляції забезпечення їх взаємозамінності використовується шаблон «стратегія» , а класи, які в ньому застосовуються, називаються сервіс-адаптерами.

Цей шаблон дозволяє змінювати обраний алгоритм незалежно від об'єктів-клієнтів, які його використовують. Адаптери баз даних використовуються для інкапсуляції особливостей доступу до різних баз даних.

На рівні представлення знаходяться каркаси і їх представлення (види). Змінні представлення мають локальну область видимості (інкапсуляція даних). Види можуть включати в себе результат рендеринга інших видів. На практиці рекомендується застосовувати не більше 3-х рівнів рендеринга через витрати процесорного часу на їх відображення.

Завдання супроводу програмного коду, налаштування параметрів роботи покладається на рівень конфігурації і супроводу. Він містить конфігураційні файли.

Пропонується додати невиконуваний рівень тестування та аудиту, призначений для верифікації вебзастосункуів. Він служить для самотестування програмного коду. Тут знаходяться тестові контролери, журнал виконання і журнал тестування

Під модулем будемо розуміти множину шаблонів і видів, контролерів, моделей даних і базових моделей, сервісів, конфігурацій і тестів, що стосуються частини предметної області вебзастосунку. Запропонована в магістерському досліджені архітектура може бути модифікована, для вирішення задачі функціонального тестування (рис. 2.4).

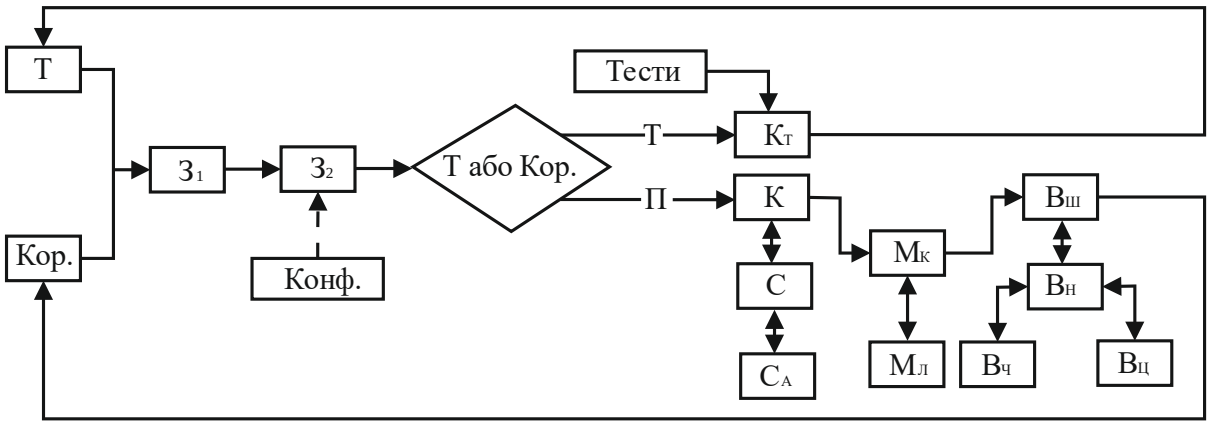


Рисунок 2.3 – Модифікована архітектура з функціональним тестуванням

Коли джерелом запиту є система тестування, основним завантажником $З_2$ підключається попередньо тестуючий контролер ($К_m$). Він є надбудовою над контролером вебзастосунку, який тестується, і має можливість підключати ті ж моделі, що і інші контролери.

Для розподілу модулів на початковому етапі рекомендується створити два модулі. Перший модуль за замовчуванням, другий - адміністративний модуль.

2.2 Проєктування на рівні контролерів

К - рівень є другим за об’ємом програмного коду. На меперішній час існує багато методів розробки класів на цьому рівні, але всі вони мають певнії недоліки. Вагомим недоліком є те, що проектувальнику потрібно вибрати частину прототипу вебдодатку, на яку слід звернути особливу увагу при проектуванні та кодуванні рівня контролера.

Створюючи вебдодаток, дуже корисно мати розуміння та уявлення про основні частини та їх взаємодію з високого рівня. Вебпрограма розділена на два основні «розділи»: front end і back end.

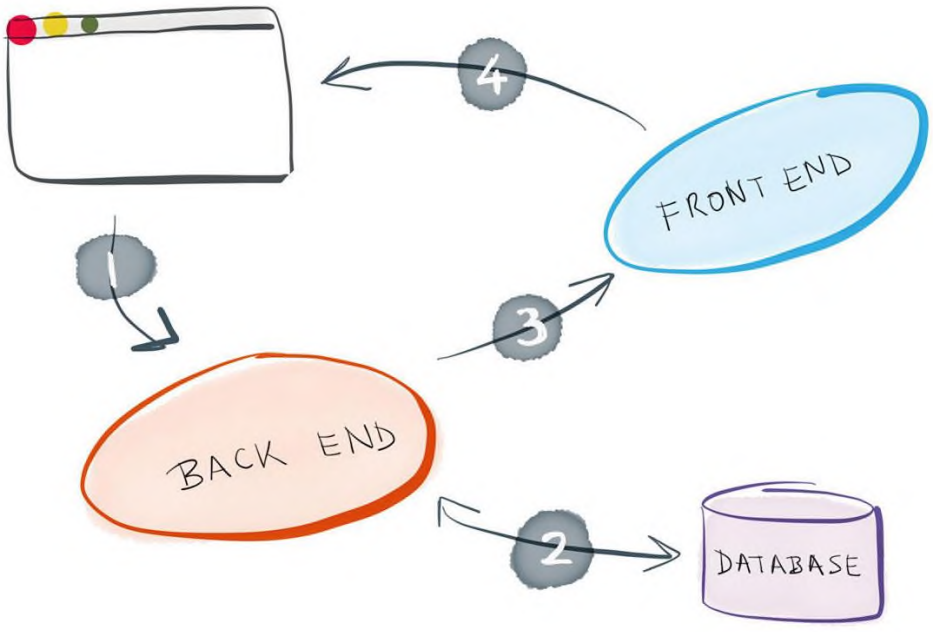


Рисунок 2.4 - Потік виконання запиту, зробленого до веб-програми

Розбиття back end: контролери.

На більш детальному рівні, у back end вебпрограми, контролери відповідають за обробку запитів користувачів - отримання та упорядкування всієї необхідної інформації, необхідної для виконання запиту, а потім надсилання її на передній кінець для відображення сторінки .

Таким чином, ви можете побачити, як отримати назву «контролер» - вони спрямовують трафік у вашій програмі. Ви створюєте єдиний контролер для кожного типу об’єкта у вашій програмі . І коли "надсилають цю інформацію на передній кінець", вони насправді надсилають її на зовнішній компонент, який називається видом .

Розбиття front end: види. Перегляди відповідають за надання сторінок, запитаних користувачем. Ви створюєте єдине подання для кожної окремої сторінки у вашій програмі. На рис.2.5 зображено повний цикл з контролерами та поданнями:

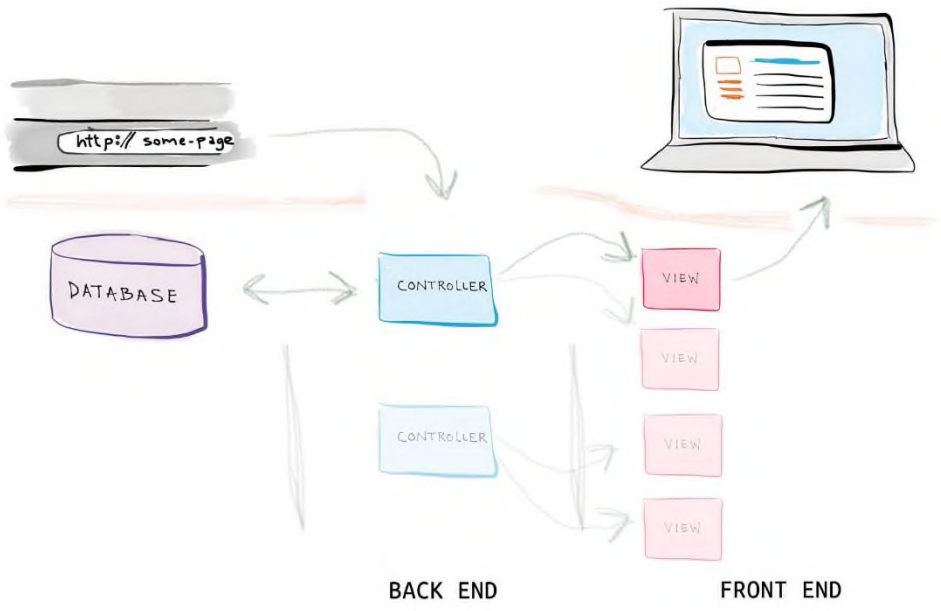


Рисунок 2.5 - Потік виконання запиту на веб-програму з контролерами та поданнями

Повний цикл виконання запиту.

Коли користувач вводить у своєму браузері URL-адресу, яка вказує на веб-програму, він робить запит . Цей запит надсилається на back end , точніше, на призначений контролер . Контролер взаємодіє з базою даних для отримання необхідної інформації. Потім вона надсилає цю інформацію до інтерфейсу, точніше, до призначеного представлення, яке робить сторінку запиту для перегляду користувачем у своєму браузері.

Моделі виступають посередниками між вашими контролерами та базою даних і використовуються контролерами для взаємодії з базою даних.

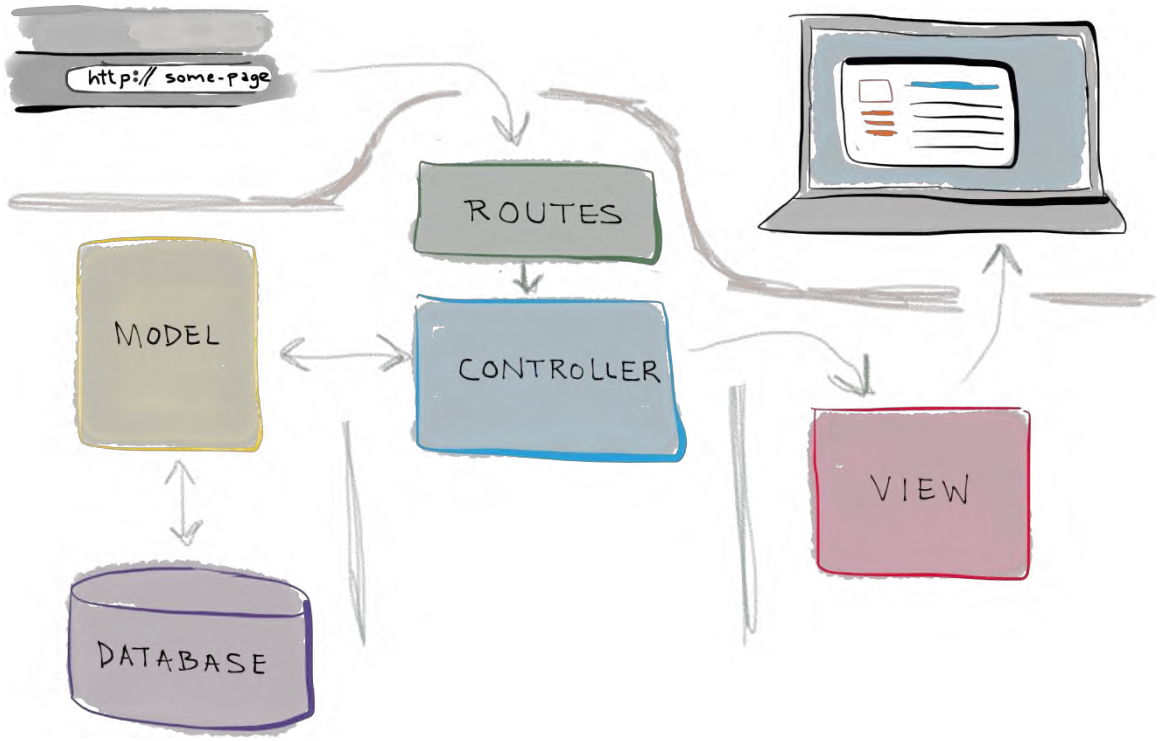


Рисунок 2.6 - Повна веб-програма MVC, що використовує маршрути для виявлення запитів на основі URL; контролери для обробки цих запитів; моделі для отримання даних з бази даних; та перегляди для відтворення сторінок

У вашому додатку в кінцевому підсумку має бути одна модель для кожного типу об’єкта у вашій програмі .

Наприклад, якщо ви створювали додаток на основі автомобілів, у вас можуть бути два різні типи об’єктів:

- автомобілів
- виробників

У такому випадку ви б створили дві моделі:

- Car модель
- Manufacturer модель

І кожен із них буде відповідальним за взаємодію з базою даних для отримання будь-якої інформації для відповідних типів об’єктів - наприклад, Car модель отримуватиме всю інформацію, пов’язану з автомобілем.

Тож давайте розглянемо повний цикл виконання запитів із включеними моделями.

Повний цикл виконання запитів MVC

Користувач вводить у свій браузер URL-адресу, яка вказує на веб-програму, таким чином роблячи запит.

Ця URL-адреса узгоджується з маршрутом , який пов'язує URL-адресу із призначеною дією контролера (тобто функцією в контролері).

Контролер використовує необхідні моделі - які взаємодіють з базою даних - для отримання необхідної інформації.

Потім вона надсилає цю інформацію у подання , яке надає сторінку запиту для перегляду користувачем у своєму браузері.

Контролери є частиною архітектури MVC . Вони є об'єктами класів, що розповсюджуються від yii \ base \ Controller, і відповідають за обробку запитів та генерацію відповідей. Зокрема, після переходу управління від додатків , контролери будуть аналізувати вхідні дані запитів, передавати їх моделям , вводити результати моделей у подання і, нарешті, генерувати вихідні відповіді.

Контролери складаються з дій, які є найосновнішими одиницями, до яких кінцеві користувачі можуть звертатися та вимагати виконання. Контролер може мати одну або кілька дій.

У наступному прикладі показано postконтролер з двома діями: viewі create:

```
namespace app\controllers;
use Yii;
use app\models\Post;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
class PostController extends Controller
{
    public function actionView($id)
    {
        $model = Post::findOne($id);
        if ($model === null) {
            throw new NotFoundHttpException;
        }
        return $this->render('view', [
            'model' => $model,
        ]);
    }
    public function actionCreate()
    {
        $model = new Post;
        if ($model->load(Yii::$app->request->post()) && $model->save()) {
            return $this->redirect(['view', 'id' => $model->id]);
        } else {
            return $this->render('create', [
                'model' => $model,
```

```
            ]);
        }
    }
}
```

Під час viewдії (визначеної actionView() методом) код спочатку завантажує модель відповідно до запитуваного ідентифікатора моделі; Якщо модель успішно завантажена, вона відобразить її за допомогою подання з іменем view. В іншому випадку це призведе до винятку.

У createдії (визначеної actionCreate() методом) код подібний. Спочатку він намагається заповнити новий екземпляр моделі за допомогою даних запиту та зберегти модель. Якщо це вдасться, він перенаправить браузер на viewдію з ідентифікатором нещодавно створеної моделі. В іншому випадку на екрані відобразиться createподання, за допомогою якого користувачі можуть надати необхідні дані.Маршрути

Кінцеві користувачі звертаються до дій за допомогою так званих маршрутів . Маршрут - це рядок, що складається з таких частин:

- ідентифікатор модуля: він існує лише в тому випадку, якщо контролер належить модулю , що не застосовується ;
- контролер ID : рядок , яка однозначно ідентифікує контролер між усіма контролерами в межах однієї програми (або той же модуля , якщо контролер належить до модуля);
- дію ID : рядок , яка однозначно ідентифікує дію серед усіх дій в рамках того ж контролера.

Маршрути приймають такий формат: ControllerID/ActionID.

Або наступний формат, якщо контролер належить модулю: ModuleID/ControllerID/ActionID

Отже, якщо користувач запитує URL-адресу http://hostname/index.php?r=site/index, indexдія в siteконтролері буде виконана.

Докладніше про те, як маршрути перетворюються на дії, див. У розділі Маршрутизація та створення URL-адрес .

Створення контролерів

У вебпрограмах контролери повинні поширюватися на yii \ web \ Controller або його дочірні класи. Подібним чином у консольних додатках контролери повинні поширюватися від yii \ console \ Controller або його дочірніх класів. Наступний код визначає sitеконтролер:

```
namespace app\controllers;
use yii\web\Controller;
class SiteController extends Controller
{}
```

Ідентифікатори контролера. Зазвичай контролер призначений для обробки запитів щодо певного типу ресурсу. З цієї причини ідентифікатори контролерів часто є іменниками, що стосуються типів ресурсів, якими вони обробляються. Наприклад, ви можете використовувати articleяк ідентифікатор контролера, який обробляє дані статті.

За замовчуванням ідентифікатори контролера повинні містити лише такі символи: англійські літери з малої літери, цифри, підкреслення, дефіси та косі риски. Так , наприклад, articlei post-commentобидва є дійсними ідентифікатори контролера, в той час як article?, PostComment, admin\postне є.

Ідентифікатор контролера може також містити префікс підкаталогу. Наприклад, admin/articleозначає articleконтролер у adminпідкаталозі в просторі імен контролера . Дійсні символи для префіксів підкаталогів включають: англійські літери з нижнього та верхнього регістру, цифри, підкреслення та прямі риски, де прямі риски використовуються як роздільники для багаторівневих підкаталогів (наприклад panels/admin).

Іменування класу контролера

Назви класів контролерів можна отримати з ідентифікаторів контролерів згідно з наступною процедурою:

1. Першу букву в кожному слові, розділену дефісами, перетворіть у верхній регістр. Зверніть увагу, що якщо ідентифікатор контролера містить косу риску, це правило застосовується лише до частини після останньої косої риски в ідентифікаторі.
2. Видаліть дефіси та будь-які скісні риски замініть назад.
3. Додайте суфікс Controller.
4. Додайте простір імен контролера .

Нижче наведено кілька прикладів, якщо припустимо, що простір імен контролера приймає значення за замовчуванням app\controllers:

- articlectає app\controllers\ArticleController;
- post-commentctає app\controllers\PostCommentController;
- admin/post-commentctає app\controllers\admin\PostCommentController;
- adminPanels/post-commentctає app\controllers\adminPanels\PostCommentController.

Класи контролера повинні завантажуватися автоматично . З цієї причини у наведених вище прикладах articleклас контролера слід зберігати у файлі, псевдонім якого @app/controllers/ArticleController.php; поки admin/post-commentконтролер повинен бути в @app/controllers/admin/PostCommentController.php.

Інформація: Останній приклад admin/post-commentпоказує, як можна помістити контролер у підкаталог простору імен контролера . Це корисно, коли ви хочете впорядкувати свої контролери за кількома категоріями, і ви не хочете використовувати модулі.

Карта контролера. Ви можете налаштувати карту контролера для подолання обмежень ідентифікаторів контролера та назв класів, описаних вище. Це в основному корисно, коли ви використовуєте сторонні контролери і у вас немає контролю над їхніми іменами класів.

Ви можете налаштувати карту контролера в конфігурації програми . Наприклад:

```
[    'controllerMap' => [
        // declares "account" controller using a class name
        'account' => 'app\controllers\UserController',

        // declares "article" controller using a configuration array
        'article' => [
            'class' => 'app\controllers\PostController',
            'enableCsrfValidation' => false,
        ], ],]
```

Контролер за замовчуванням.

Кожна програма має контролер за замовчуванням, вказаний через властивість yii \ base \ Application :: $ defaultRoute . Коли в запиті не вказано маршрут , буде використаний маршрут, вказаний цією властивістю. Для веб-додатків його значення дорівнює 'site', тоді як для консольних додатків - це help. Отже, якщо URL-адреса є http://hostname/index.php, тоді sitеконтролер обробляє запит.

Ви можете змінити контролер за замовчуванням із наступною конфігурацією програми :

```
[
    'defaultRoute' => 'main',
]
```

Створення дій.

Створення дій може бути таким простим, як визначення так званих методів дій у класі контролера. Метод дії - це загальнодоступний метод, назва якого починається зі слова action. Повернене значення методу дії представляє дані відповіді, що надсилаються кінцевим користувачам. Наступний код визначає дві дії, indexі hello-world:

```
namespace app\controllers;
use yii\web\Controller;
```

```
class SiteController extends Controller
{    public function actionIndex()
   {       return $this->render('index');   }
   public function actionHelloWorld()
   {       return 'Hello World';    }}
```

Ідентифікатори дій.

Дія часто призначена для здійснення певної маніпуляції з ресурсом. З цієї причини, ідентифікатори дій, як правило , дієслова, такі , як view, updatei т.д.

За замовчуванням ідентифікатори дій повинні містити лише такі символи: англійські літери малими літерами, цифри, підкреслення та дефіси (ви можете використовувати дефіси для розділення слів). Так , наприклад, view, update2i comment-postвсі дійсні ідентифікатори дій, в той час як view?i Updateнемає.

Ви можете створювати дії двома способами: вбудованими та самостійними. Вбудована дія визначається як метод у класі контролера, тоді як автономна дія - це клас, що поширюється на yii \ base \ Action або його дочірні класи. Вбудовані дії вимагають менше зусиль для створення, і їх часто надають перевагу, якщо ви не збираєтеся повторно використовувати ці дії. З іншого боку, автономні дії в основному створюються для використання в різних контролерах або для їх перерозподілу як розширення.

Вбудовані дії.

Вбудовані дії відносяться до дій, які визначені з точки зору методів дій, як ми щойно описали.

Назви методів дії походять від ідентифікаторів дій згідно з такою процедурою:

1. Перетворіть першу літеру в кожному слові ідентифікатора дії у верхній регістр.
2. Видалити дефіси.
3. Додайте префікс action.

Наприклад, indexстає actionIndex, і hello-worldстає actionHelloWorld.

Примітка: Назви методів дій чутливі до регістру . Якщо у вас є метод з іменем ActionIndex, він не буде розглядатися як метод дії, і в результаті запит на indexдію призведе до винятку. Також зверніть увагу, що методи дій повинні бути загальнодоступними. Приватний або захищений метод НЕ визначає вбудовану дію.

Вбудовані дії - це найбільш часто визначаються дії, оскільки для їх створення потрібно мало зусиль. Однак, якщо ви плануєте повторно використовувати одну і ту ж дію в різних місцях або якщо ви хочете перерозподілити дію, вам слід розглянути можливість визначення її як самостійної дії .

Автономні дії. Самостійні дії визначаються в термінах класів дій, що поширюються на yii \ base \ Action або його дочірні класи. Наприклад, у випусках Yii є yii \ web \ ViewAction та yii \ web \ ErrorAction , які є самостійними діями.

Щоб використовувати автономну дію, слід оголосити її на карті дій , замінивши метод yii \ base \ Controller :: actions () у своїх класах контролерів.

Результати дій. Повернене значення методу дії або run()методу самостійної дії є значним. Це означає результат відповідної дії.

Повернене значення може бути об'єктом відповіді, який буде надіслано кінцевому користувачеві як відповідь.

- Для вебдодатків значенням , що повертається, також можуть бути деякі довільні дані, які будуть присвоєні yii \ web \ Response :: $ data і будуть далі перетворені у рядок, що представляє тіло відповіді.
- Для консольних програм повернене значення також може бути цілим числом, що представляє статус виходу виконання команди.

У наведених вище прикладах результатами дії є всі рядки, які будуть розглядатися як тіло відповіді, яке буде надіслано кінцевим користувачам.

Параметри дії. Методи дій для вбудованих дій та run()методи самостійних дій можуть приймати параметри, які називаються параметрами дій . Їх значення отримуються із запитів. Для веб-додатків значення кожного параметра дії

отримується з $_GETвикористанням імені параметра як ключа; для консольних програм вони відповідають аргументам командного рядка.

У наступному прикладі view дія (вбудована дія) оголосила два параметри: $idi $version.

```
namespace app\controllers;
use yii\web\Controller;
class PostController extends Controller
{
   public function actionView($id, $version = null)
   {    // ..    }}
```

Параметри дії будуть заповнюватися наступним чином для різних запитів:

- http://hostname/index.php?r=post/view&id=123: $idпараметр буде заповнений значенням '123', в той час $versionяк все ще, nullоскільки versionпараметр запиту відсутній .
- http://hostname/index.php?r=post/view&id=123&version=2: Te $idi $versionпараметри будуть заповнені '123'і '2', відповідно.
- http://hostname/index.php?r=post/view: буде викинуто виняток yii \ web \ BadRequestHttpException , оскільки необхідний $idпараметр не вказаний у запиті.
- http://hostname/index.php?r=post/view&id[]=123: буде викинуто виняток yii \ web \ BadRequestHttpException , оскільки $idпараметр отримує несподіване значення масиву ['123'].

Тепер, якщо запит є http://hostname/index.php?r=post/view&id[]=123, $idпараметр прийме значення ['123']. Якщо запит є http://hostname/index.php?r=post/view&id=123, $idпараметр все одно отримає те саме значення масиву, оскільки скалярне значення '123'буде автоматично перетворено в масив.

Дія за замовчуванням. Кожен контролер має дію за замовчуванням, вказану за допомогою властивості yii \ base \ Controller :: $ defaultAction . Коли маршрут

містить лише ідентифікатор контролера, це означає, що запитується дія за замовчуванням зазначеного контролера.

Дію за замовчуванням встановлено як index. Якщо ви хочете змінити значення за замовчуванням, просто перевизначте цю властивість у класі контролера, як показано нижче:

```
namespace app\controllers;
use yii\web\Controller;
class SiteController extends Controller
   public $defaultAction = 'home';
   public function actionHome()
   {       return $this->render('home');    }}
```

Життєвий цикл контролера. Під час обробки запиту програма створює контролер на основі запитуваного маршруту . Потім контролер пройде такий життєвий цикл для виконання запиту:

1. Після створення та налаштування контролера викликається метод yii \ base \ Controller :: init () .
2. Контролер створює об'єкт дії на основі запитуваного ідентифікатора дії:
   - Якщо ідентифікатор дії не вказаний, буде використаний ідентифікатор дії за замовчуванням .
   - Якщо ідентифікатор дії буде знайдений на карті дій, буде створена автономна дія;
   - Якщо виявлено, що ідентифікатор дії відповідає методу дії, буде створено вбудовану дію;
   - В іншому випадку буде викинуто виняток yii \ base \ InvalidRouteException .
3. Контролер послідовно викликає beforeAction()метод програми, модуль (якщо контролер належить модулю) та контролер.

- Якщо один із викликів повернеться false, решта не викликаних beforeAction() методів буде пропущено, а виконання дії буде скасовано.
- За замовчуванням кожен beforeAction() виклик методу ініціює beforeAction подію, до якої ви можете приєднати обробник.

4. Контролер запускає дію.

- Параметри дії будуть проаналізовані та заповнені з даних запиту.

5. Контролер послідовно викликає afterAction()метод контролера, модуль (якщо контролер належить модулю) та додаток.

- За замовчуванням кожен afterAction()виклик методу ініціює afterActionподію, до якої ви можете приєднати обробник.

6. Додаток прийме результат дії та призначить його відповіді .

Кращі практики. У добре розробленому додатку контролери часто дуже тонкі, кожна дія містить лише кілька рядків коду. Якщо ваш контролер досить складний, це, як правило, вказує на те, що вам слід його переформатувати і перенести якийсь код в інші класи.

Ось деякі конкретні найкращі практики. Контролери

- може отримати доступ до даних запиту ;
- може викликати методи моделей та інших компонентів служби із даними запиту;
- може використовувати подання для складання відповідей;
- Не слід обробляти дані запиту - це слід робити на рівні моделі ;
- слід уникати вбудовування HTML або іншого презентаційного коду - це краще робити у поданнях .

В магістерській роботі запропоновано «ресурсний» метод організації контролерів вебзастосунків (рис. 2.7). Головна ідея методу полягає в тому, що розробка рівня контролерів проводиться, з врахуванням не моделей, і їх представлень, а на сутностях, над якими будуть проводитись операції. Ці сутності

виступають ресурсами вебзастосунку, а зовнішній ПЗ, яке забезпечує інтерфейс до ресурсу буде називатися контролером ресурсу.

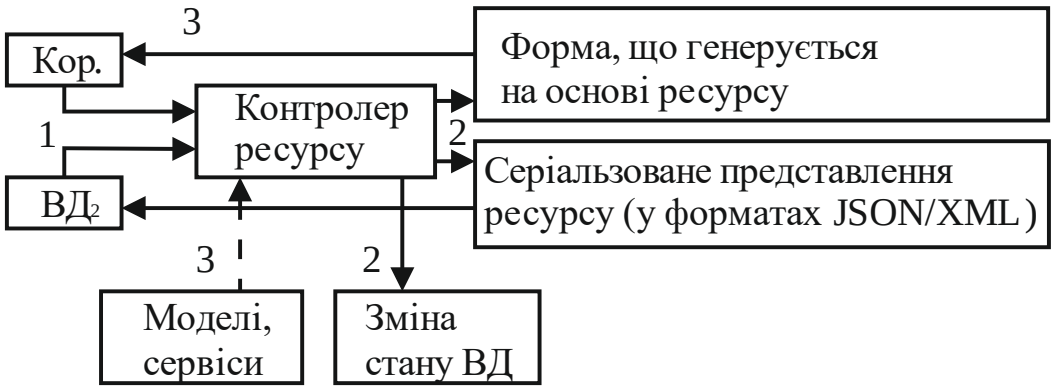


Рисунок 2.7 – Контролер ресурсу вебзастосунку

Ресурсом може бути будь-що: модель, представлення або ж сутність (наприклад, сесії користувача), яка потребує зберігання в БД. Також ресурсом може виступати група сутностей, або ж група ресурсів. Це дає можливість рівню контролерів надавати різне представлення ресурсу в залежності отриманого запиту. Наприклад, для одного користувача повертати HTML-представлення, а для його серіалізоване представлення в одному із стандартних форматів (JSON, XML).

1) Імена модулів кодуються шляхом присвоєння кожному модулю послідовного унікального числового ідентифікатора.

2) Далі відбувається кодування імен контролерів, аналогічно до п. 1. Якщо буде знайдено контролер з іменем, вихідний код якого вже відомий, йому відповідно присвоюється знайдений код (без генерування нового).

3) Аналогічним чином відбувається кодування дій (публічних функції) класу контролера.

4) Формується асоціативний масив глибини 3, в який записано відповідність кодів модулів, контролерів та їх дій.

5) На основі отриманих даних будується тривимірний точковий графік (рис. 2.8).

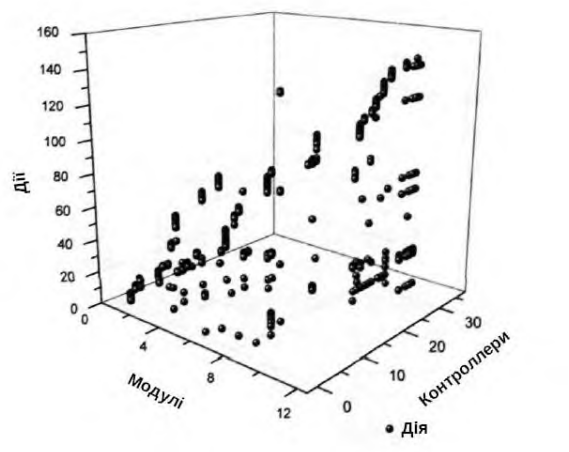


Рисунок 2.8 – MVC-графік вебзастосунку

На осях графіку відкладаються різні коди: по осі абсцис розташовуються коди модулів, по осі ординат відкладаються коди контролерів, а по осі аплікат розташовуються коди дій. Кожній дії контролера в запропонованій моделі відповідає точка на графіку. Даний графік безпосередньо візуалізує метрики вебзастосунку, що дає можливість провести графічний аналіз. Далі надано список визначень графічного аналізу.

2.3 Методи розробки рівня моделі

Моделі є частиною архітектури MVC . Вони є об’єктами, що представляють ділові дані, правила та логіку.

Ви можете створити класи моделей, розширивши yii \ base \ Model або його дочірні класи. Базовий клас yii \ base \ Model підтримує безліч корисних функцій:

- Атрибути : представляють бізнес-дані і до них можна отримати доступ як звичайні властивості об’єкта або елементи масиву;
- Мітки атрибутів : укажіть мітки відображення атрибутів;
- Масове призначення : підтримує заповнення кількох атрибутів за один крок;
- Правила перевірки : забезпечує вхідні дані на основі оголошених правил перевірки;

- Експорт даних : дозволяє експортувати дані моделі у вигляді масивів із настроюваними форматами.

ModelКлас також базовий клас для більш просунутих моделей, таких як Active Record . Будь ласка, зверніться до відповідної документації, щоб отримати докладнішу інформацію про ці вдосконалені моделі.

Інформація: Вам не потрібно базувати класи своїх моделей на yii \ base \ Model . Однак, оскільки існує багато компонентів Yii, побудованих для підтримки yii \ base \ Model , зазвичай це найкращий базовий клас для моделі.

Атрибути. Моделі представляють бізнес-дані з точки зору атрибутів . Кожен атрибут схожий на загальнодоступну властивість моделі. Метод yii \ base \ Model :: attributes () визначає, які атрибути має клас моделі.

Ви можете отримати доступ до такого атрибута, як доступ до звичайної властивості об’єкта:

```
$model = new \app\models\ContactForm;

// "name" is an attribute of ContactForm
$model->name = 'example';
echo $model->name;
```

Визначення атрибутів. За замовчуванням, якщо ваш клас моделі поширюється безпосередньо з yii \ base \ Model , усі його нестатичні змінні загальнодоступного члена є атрибутами. Наприклад, ContactFormклас нижче модель має чотири атрибути: name, email, subjecti body. ContactFormМодель використовується для представлення вхідних даних , отриманих з HTML форми.

```
namespace app\models;
use yii\base\Model;
class ContactForm extends Model
{    public $name;
   public $email;
   public $subject;
```

```
    public $body;
}
```

Ви можете замінити yii \ base \ Model :: attributes (), щоб визначити атрибути по-іншому. Метод повинен повертати імена атрибутів у моделі. Наприклад, yii \ db \ ActiveRecord робить це, повертаючи імена стовпців пов'язаної таблиці бази даних як імена її атрибутів. Зверніть увагу, що вам може знадобитися замінити магічні методи, такі як __get(), __set()щоб атрибути могли бути доступні як звичайні властивості об'єкта.

Мітки атрибутів

Під час відображення значень або отримання вхідних даних для атрибутів часто потрібно відображати деякі мітки, пов’язані з атрибутами. Наприклад, з урахуванням іменованого атрибута firstName, можливо, ви захочете показати ярлик, First Name який є більш зручним для користувача, коли відображається для кінцевих користувачів у таких місцях, як введення форми та повідомлення про помилки.

Строго кажучи, мітки атрибутів є частиною подань . Але оголошення ярликів у моделях часто буває дуже зручним і може призвести до дуже чистого та багаторазового коду.

Сценарії.Модель може використовуватися в різних сценаріях . Наприклад, Usermодель може використовуватися для збору вхідних даних для входу користувачів, але вона також може використовуватися для цілей реєстрації користувачів. У різних сценаріях модель може використовувати різні бізнес-правила та логіку. Наприклад, emailатрибут може знадобитися під час реєстрації користувача, але не під час входу користувача.

Модель використовує властивість yii \ base \ Model :: $ scenario для відстеження сценарію, в якому вона використовується. За замовчуванням модель підтримує лише один сценарій з іменем default. Наступний код показує два способи встановлення сценарію моделі:

```
// scenario is set as a property
```

```
$model = new User;
$model->scenario = User::SCENARIO_LOGIN;

// scenario is set through configuration
$model = new User(['scenario' => User::SCENARIO_LOGIN]);
```

Правила перевірки. Коли дані моделі отримуються від кінцевих користувачів, їх слід перевірити, щоб переконатися, що вони відповідають певним правилам (так звані правила перевірки , також відомі як бізнес-правила). Наприклад, для даної ContactFormмоделі ви можете переконатися, що всі атрибути не порожні, а emailатрибут містить дійсну адресу електронної пошти. Якщо значення деяких атрибутів не задовольняють відповідним діловим правилам, повинні відображатися відповідні повідомлення про помилки, щоб допомогти користувачеві виправити помилки.

Правило може бути використано для перевірки одного або декількох атрибутів, а атрибут може перевірятися одним або декількома правилами. Будь ласка, зверніться до розділу перевірки введення, щоб отримати докладнішу інформацію про те, як оголосити правила перевірки.

Експорт даних. Моделі часто потрібно експортувати в різних форматах. Наприклад, вам може знадобитися перетворити колекцію моделей у формат JSON або Excel. Процес експорту можна розбити на два незалежних етапи:

- моделі перетворюються на масиви;
- масиви перетворюються в цільові формат

Поле - це просто іменований елемент у масиві, який отримується викликом методу yii \ base \ Model :: toArray () моделі.

За замовчуванням назви полів еквівалентні іменам атрибутів. Тим НЕ менше, ви можете змінити цю поведінку, перекриваючи поля () і / або extraFields () методи. Обидва методи повинні повернути список визначень полів. Поля, визначені параметром, fields()є полями за замовчуванням, тобто вони toArray()будуть повертатися за замовчуванням. extraFields()Метод визначає

додатково доступні поля , які також можуть бути повернуті до toArray()тих пір , як ви визначаєте їх з допомогою $expandпараметра. Наприклад, наступний код повертає всі поля , певні в fields()і тому prettyNamei fullAddressполе , якщо вони визначені в extraFields().

$array = $model->toArray([], ['prettyName', 'fullAddress']);

Ви можете замінити fields()додавання, видалення, перейменування або перевизначення полів. Повернене значення fields() має бути масивом. Ключі масиву - це імена полів, а значення масиву - це відповідні визначення полів, які можуть бути як іменами властивостей / атрибутів, так і анонімними функціями, що повертають відповідні значення полів. У спеціальному випадку, коли назва поля збігається з назвою визначального атрибута, ви можете опустити ключ масиву. Кращі практики. Моделі є центральними місцями для представлення ділових даних, правил та логіки. Їх часто потрібно використовувати повторно в різних місцях. У добре розробленому додатку моделі, як правило, набагато товщі, ніж контролери .

Таким чином, моделі

- може містити атрибути для представлення ділових даних;
- може містити правила перевірки для забезпечення достовірності та цілісності даних;
- може містити методи реалізації ділової логіки;
- Не повинен безпосередньо отримувати доступ до запитів, сеансів або будь-яких інших екологічних даних. Ці дані повинні вводитися контролерами в моделі;
- слід уникати вбудовування HTML або іншого презентаційного коду - це краще робити у поданнях ;
- уникайте занадто багато сценаріїв в одній моделі.

Зазвичай ви можете розглянути останню рекомендацію вище, коли розробляєте великі складні системи. У цих системах моделі можуть бути дуже товстими, оскільки вони використовуються в багатьох місцях і, отже, можуть

містити багато наборів правил та ділової логіки. Це часто закінчується кошмаром при підтримці коду моделі, оскільки одне натискання коду може вплинути на кілька різних місць. Щоб зробити код моделі більш ремонтопридатним, ви можете взяти таку стратегію:

- Визначте набір базових класів моделей, якими користуються різні програми або модулі . Ці класи моделей повинні містити мінімальні набори правил та логіки, які є загальними серед усіх їхніх звичок.
- У кожному додатку або модулі, що використовує модель, визначте конкретний клас моделі, виходячи із відповідного базового класу моделі. Класи конкретної моделі повинні містити правила та логіку, які є специфічними для цієї програми або модуля.

Наприклад, у розширеному шаблоні проекту ви можете визначити базовий клас моделі common\models\Post. Потім для інтерфейсної програми ви визначаєте та використовуєте конкретний клас моделі, frontend\models\Postякий поширюється на common\models\Post. І аналогічно для додаткової програми, яку ви визначаєте backend\models\Post. З цією стратегією ви будете впевнені, що код, що frontend\models\Post входить, стосується лише програми інтерфейсу, і якщо ви внесете до нього будь-які зміни, вам не потрібно турбуватися, чи може ця зміна зламати програму серверного інтерфейсу.

Поле - це просто іменований елемент у масиві, який отримується викликом методу yii \ base \ Model :: toArray () моделі.

За замовчуванням назви полів еквівалентні іменам атрибутів. Тим НЕ менше, ви можете змінити цю поведінку, перекриваючи поля () і / або extraFields () методи. Обидва методи повинні повернути список визначень полів. Поля, визначені параметром, fields()є полями за замовчуванням, тобто вони toArray()будуть повертатися за замовчуванням. extraFields()Метод визначає додатково доступні поля , які також можуть бути повернуті до toArray()тих пір , як ви визначаєте їх з допомогою $expandпараметра. Наприклад, наступний код

повертає всі поля , певні в fields()і тому prettyNamei fullAddressполе , якщо вони визначені в extraFields().

$array = $model->toArray([], ['prettyName', 'fullAddress']);

Ви можете замінити fields()додавання, видалення, перейменування або перевизначення полів. Повернене значення fields() має бути масивом. Ключі масиву - це імена полів, а значення масиву - це відповідні визначення полів, які можуть бути як іменами властивостей / атрибутів, так і анонімними функціями, що повертають відповідні значення полів. Кращі практики.Моделі є центральними місцями для представлення ділових даних, правил та логіки. Їх часто потрібно використовувати повторно в різних місцях. У добре розробленому додатку моделі, як правило, набагато товщі, ніж контролери .

Таким чином, моделі

- може містити атрибути для представлення ділових даних;
- може містити правила перевірки для забезпечення достовірності та цілісності даних;
- може містити методи реалізації ділової логіки;
- Не повинен безпосередньо отримувати доступ до запитів, сеансів або будь-яких інших екологічних даних. Ці дані повинні вводитися контролерами в моделі;
- слід уникати вбудовування HTML або іншого презентаційного коду - це краще робити у поданнях ;
- уникайте занадто багато сценаріїв в одній моделі.

Щоб зробити код моделі більш ремонтопридатним, ви можете взяти таку стратегію:

- Визначте набір базових класів моделей, якими користуються різні програми або модулі . Ці класи моделей повинні містити мінімальні набори правил та логіки, які є загальними серед усіх їхніх звичок.
- У кожному додатку або модулі, що використовує модель, визначте конкретний клас моделі, виходячи із відповідного базового класу моделі. Класи

конкретної моделі повинні містити правила та логіку, які є специфічними для цієї програми або модуля.

Наприклад, у розширеному шаблоні проекту ви можете визначити базовий клас моделі common\models\Post. Потім для інтерфейсної програми ви визначаєте та використовуєте конкретний клас моделі, frontend\models\Postякий поширюється на common\models\Post. І аналогічно для додаткової програми, яку ви визначаєте backend\models\Post. Рівень моделей вебзастосунку (рис. 2.9) відповідає за зберігання стану S вебзастосунку. Воно включає в себе стан між запитами користувача протягом сеансу роботи (короткочасний S_S) і стан між сеансами роботи (довгостроковий S_{DB}).

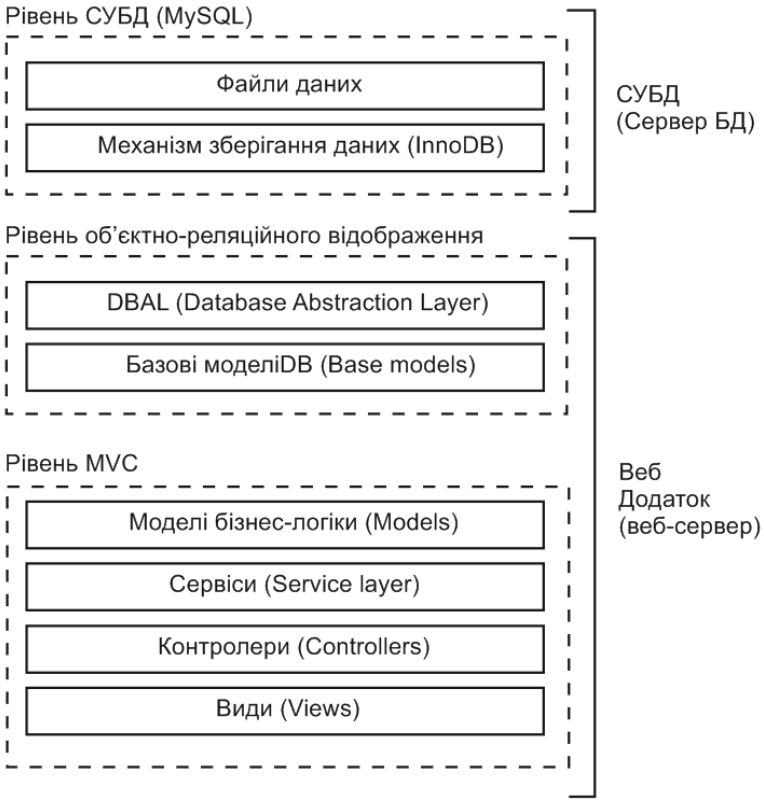


Рисунок 2.9 –Абстракції на рівні моделей

В ході роботи запропоновано метод об'єктно-реляційне перетворення, який здійснює пряме і зворотнє перетворення таблиць бази даних в екземпляри класів базових моделей програмного коду. Існуючими аналогами є системи, такі як Doctrine, Propel [33

З огляду на реляційна БД і класичні класи в ООП припускають різні підходи до зберігання даних, а також механізми управління цими даними, виникає завдання синхронізації змін між цими рівнями.

Завдання синхронізації досить складне і містить в собі великий спектр підзадач, серед яких картування, обчислення змін, безпеку, оптимізаційні заходи і т.д.

ORM (Object Relational Mapper) - це якраз той інструмент, який бере рішення всіх цих завдань на себе [31]. Крім цього він надає ряд додаткових корисних можливостей, таких як Events, QueryBuilder, DQL для швидкої побудови запитів і т. д.

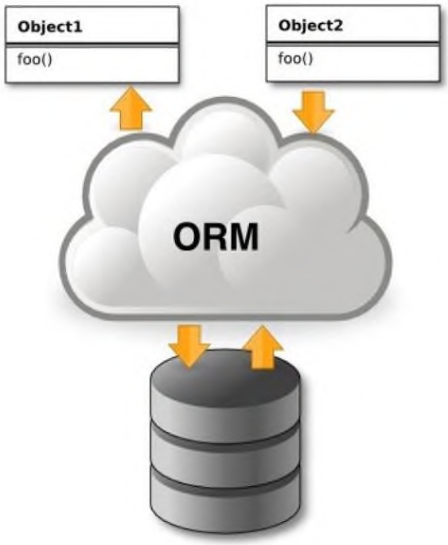


Рисунок 2.10 - Архітектура Doctrine 2

В основі Doctrine лежать патерни і абстракції, розуміння яких допоможе краще розібратися в принципах роботи цієї ORM. игляді і є реалізація цього патерну.

Data Mapper pattern. З тієї причини, що об'єктна і реляційна схеми неідентичні, виникає проблема обміну даними між ними. У розробника без ORM просто немає іншого виходу, як писати однотипні, повторювані SQL-запити для всіх об'єктів програми. Це вкрай незручно і складно в підтримці, а значить, дорого.

Data mapper вирішує цю задачу за рахунок ізоляції об'єктів і БД відносно один одного і в якості основної, концептуальної абстракції використовує Entity.

За фактом, Entity - це звичайний PHP клас, де властивості зіставлені з полями таблиці з бази даних, для якої і створювалася Entity (далі «сутність»). Всю ж роботу по Маппінг полів, обчисленню змін і т.д. бере на себе mapper.

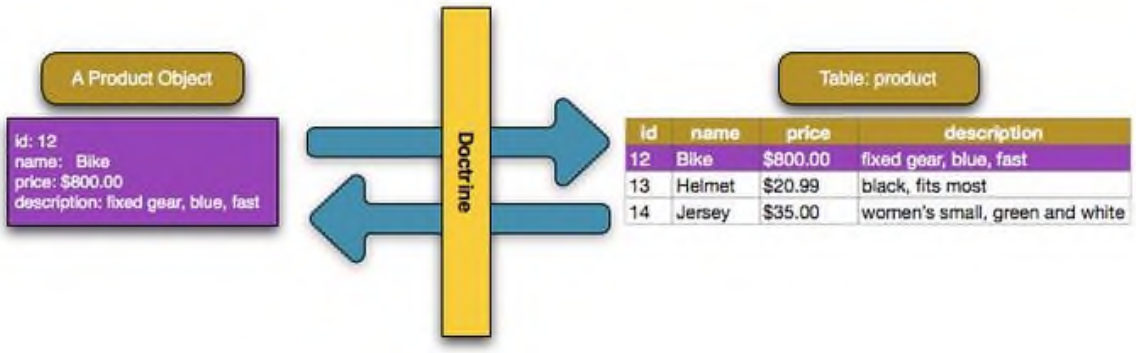


Рисунок 2.11 - EntityManager

EntityManager (EM) - це класична реалізація патерну Facade. Через його легкий API ми працюємо з рядом підсистем: Unit Of Work, Query language і Repository API. Управління сутностями виконується виключно через API EntityManager-a.

Варто згадати, що все ж якась оптимізація і в цьому випадку присутній: після виконання другого запиту до БД новий об'єкт суті створено не буде, задіюється об'єкт, який вже збережений в карті.

Lazy Loading Pattern. Коли ми хочемо отримати інформацію про сутність, в якій є асоціації, Doctrine дозволяє управляти тим, чи потрібно конструювати об'єкти також для них.

Коли це корисно? Уявіть: ви працюєте з системою типу «форум», де користувачі можуть залишати пости в топіках. У вас виникла необхідність запросити сутність топіка з БД, і оскільки коментарі є пов'язаними сутностями, можна припустити, що Doctrine запросить дані по ним, в тому числі. Очевидно, що це зайвий, невиправданий overhead на передачу даних, зайві запити, крім цього, з додатком буде потрібно більше пам'яті для зберігання об'єктів коментарів.

Цю проблему Doctrine вирішує за допомогою механізму ледачого завантаження (Lazy Loading). За замовчуванням, до речі, цей механізм активований для всіх асоціацій, всього ж доступно три варіанти:

- LAZY (за замовчуванням) - в пам'ять буде завантажений тільки керований об'єкт, а асоціації будуть довантажуючи тільки при першому зверненні до них;
- EAGER - в пам'ять буде завантажений як керований об'єкт, так і всі асоціації;
- EXTRA LAZY - в деяких випадках, подгрузка всіх пов'язаних об'єктів недоцільна, навіть якщо це відбувається тільки при першому зверненні. Скажімо, ви хочете отримати тільки кількість пов'язаних об'єктів Collection # count (). Для оптимального вирішення цього завдання можна задіяти EXTRA_LAZY опцію. Doctrine - це складна бібліотека, і вона має великий обсяг програмного коду і надлишок кількості реалізованих функцій (більше 100 класів)., що викликає проблеми з продуктивністю.

З огляду на переваги застосування ОРП-систем сформулюємо завдання для розроблювального підходу:

1) Адекватне відображення об'єктів предметної області (різних за струтурою, розміщених в багатьох таблицях на різних серверах) вебзастосунків і їх зв'язків в реляційну структуру БД.

2) Підтримка багатомовності.

3) Вдосконалені пошукові можливості (підтримка складних запитів).

Для здійснення РО-перетворень на стороні об'єктної моделі запроваджено декларативний опис реляційної моделі на мові опису об'єктів (*ODL - Object Definition Language*) [17], та мови сімейства OQL (*Object Query Language*) [19]. Відмінності OQL від SQL полягають в тому, що вхідними даними запитів OQL є об'єкти, а не таблиці.

Це передбачає значні витрати на обчислювальну потужність сервера, тому в цьому досліджені пропонується ряд змін, спрямованих на підвищення продуктивності операцій перетворення:

1) Зведення ОДЛ до діалекту основної мови розвитку. Для досліджуваної платформи LAMP це PHP.

2) Визначення кінцевого набору трансформованих типів та їх відображень.

3) Використання діалекту основної мови платформи для виконання операцій OQL, а інтерпретатора платформи - для інтерпретації команд OQL.

4) Визначення спеціальних типів таблиць та їх ODL.

Це передбачає значні витрати на обчислювальну потужність сервера, тому в цій тезі пропонується ряд змін, спрямованих на підвищення продуктивності операцій перетворення:

1) Зведення ODL до діалекту основної мови додатку. В даному випадку це PHP.

2) Визначення кінцевого набору трансформованих типів та їх відображень.

3) Використання діалекту основної мови платформи для виконання операцій OQL, а інтерпретатора платформи - для інтерпретації команд OQL.

4) Проєктування спеціальних типів таблиць та їх ODL.

Об'єктом *O* називається представлення сутності предметної області, яке є множиною атрибутів об'єктної моделі *A* $\{A_{O1}, A_{O2}, \ldots, A_{On}\}$. Під класом *C* розуміється сутність, яка може бути визначена як сукупність реалізацій класу.

Запропонована «ОРП» модель буде виглядати так:

$$a = \left\{\left\{\mathrm{M}_{\text{відношення}}\right\}_n, \{A\}_n, B\right\},$$

де a − об'єкт класу «ОРП», n − кількість відносин, що входять в модель (n ≥ 1); $\mathrm{M}_{\text{відношення}}$− набір метаданих відносин, А− набір атрибутів відносин, В − види функцій над об'єктами класу.

Тоді логічну модель даних $\mathrm{M}_{Л}$ визначимо:

$$\mathrm{M}_{Л} = \{\bar{T}, \bar{R}\},$$

де $\bar{T}$ − набір декларативних описів схем відношень, які належать моделі, $\bar{R}$ − набір зв'язків з іншими відношенями.

Вся множина зв'язків буде виглядати наступним чином:

$$T_i = \{name, ct, db, \vec{C}\}; Ci = \{cname, ctype\};$$

$$R_i = \{rt, FK, DK, DM, iT, iFK, iDK\};$$

$$type \in \{T_C, T_{P,} T_F, T_{FT}\};$$

$$rt \in \{hasOne, hasMany, hasManyToMany, hasManyToManyExtended\},$$

де *name* − назва відношення в базі; *ct* – тип відношення; *db* − назва БД на сервері розподіленої системи керування; $\vec{C}$ – множина описів атрибутів реляційної моделі; *cname* – ім'ч атрибуту реляційної моделі; *ctype* – тип даних PHP, для цього атрибуту; *rt* – тип зв'язку; *DK* – ключ отримувача зв'язку; *FK* – зовнішній ключ джерела зв'язку; *iT* – таблиця перетинів для відношення багато-до-багатьох, *iFK* – зовнішній ключ перетинів; *iDK* – ключ отримувача зв'язку через таблицю перетинів.

Особливістю методики «ОРП» є те, що n кількість відношень, які використовуються для відображення) може бути більше 1, тоді, як для існуючих методик Doctrine, Propel, n = 1.

Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність..

Система створює віртуальну об'єктну базу даних (логічні моделі групуються в колекції), в якій користувач має змогу працювати з об'єктами, а не таблицями і відношеннями.

Множина операцій системи ділиться на ОР перетворення, та зворотні РО перетворення, а також службові операції. Реляційно об'єктні перетворення необхідні для отримання даних від сервера БД і перенаправлення їх у вебзастосунок, ОР-перетворення застосовуються при зворотному процесі.

2.4 Висновки

У цьому розділі викладено практичні прийоми розробки вебзастосунків із використанням безкоштовних Інтернет-технологій та запропоновано шляхи реалізації теоретичних основ першого розділу. Підсумовуючи зміст розділу, ми можемо зробити наступні висновки:

2) Запропоновано «ресурсний» метод організації контролерів вебзастосунків. Головна ідея методу полягає в тому, що розробка рівня контролерів проводиться, з врахуванням не моделей, і їх представлень, а на сутностях, над якими будуть проводитись операції. Ці сутності виступають ресурсами вебзастосунку, а зовнішній ПЗ, яке забезпечує інтерфейс до ресурсу буде називатися контролером ресурсу.

3) Запропонована модель «ОРП», особливістю якої є те, що n кількість відношень, які використовуються для відображення може бути більше 1, тоді, як для існуючих методик Doctrine, Propel, $n = 1$. Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність.

3 МЕТОД РОЗРОБКИ ВЕБДОДАТКІВ НА ОСНОВІ ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ПЕРЕТВОРЕННЯ

3.1 Алгоритм роботи зі сторонніми бібліотеками

При розробці певного набору функцій відповідно до відомих вимог доцільно провести пошук готового рішення або прикладу програмного коду, що реалізує таке або подібне завдання. Майже завжди вирішення деяких проблем уже існує, але виникає проблеми інтеграції сторонніх бібліотек у вебзастосунок [37].

Далі описуються методи вирішення практичних проблем, що реалізують ці підходи (рис. 3.1).

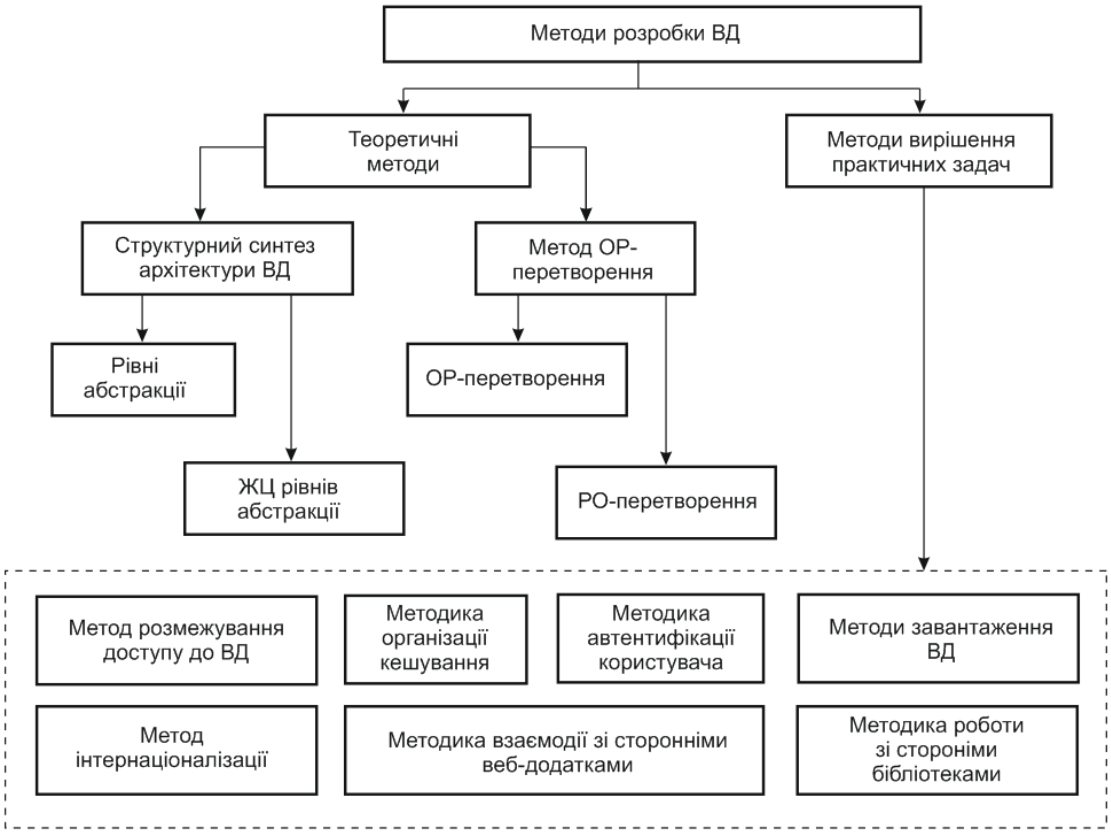


Рисунок 3.1 – Методи розробки вебзастосунку

Зараз існують «каркаси вебпроектування», що складаються з сукупності модулів або ж реалізовані, як монолітний вебдодаток. Як приклад каркасів

першого типу можна навести таку бібліотеку, Ruby on Rails [39]. Каркаси монолітного типу представлені бібліотеками Symfony [40], Yii та іншими.

Каркас вебзастосунків, вебфреймворк— програмний каркас, призначений для створення вебзастосунків, служб або ресурсів. Він спрощує розробку, частково за рахунок автоматизації, і позбавляє від необхідності написання рутинного коду. Більшість каркасів спрощують доступ до баз даних, також зменшують дублювання коду.

Велика частина каркасів вебдодатків реалізує шаблон проектування Модель-вид-контролер (MVC). Однак, також можуть використовуватися й інші шаблони, наприклад, Model-View-Presenter (Backbone.js) або Model-View-View-Model(Knockout.js).

Веб каркаси можна також за рівнем абстракції умовно розділити на 3 типи:

- Засновані на запитах: каркас безпосередньо обробляє вхідні запити. Збереження стану відбувається за рахунок серверних сесій. Приклади: Django, Ruby on Rails, Struts, Grails.
- Засновані на компонентах: каркас абстрагує обробку запитів всередині стандартних компонентів і самостійно стежить за станом. Своєю поведінкою дані каркаси нагадують стандартні програмні графічні інтерфейси. Приклади: JavaServer Faces, Tapestry, Wicket.
- Rich Internet Application-каркаси: служать для розробки повноцінних додатків, що запускаються всередині браузера. Приклад: Apache Flex

Бібліотека розширення - це частина вебзастосунку, яка знаходиться поза її модулями. У той же час логіка предметної області не повинна відображатися в бібліотеці розширення. Запропонований алгоритм використання бібліотеки розширення такий:

1) Визначено простір імен сторонніх бібліотек. Усі файли бібліотеки повинні зберігатися в директорії, яка відповідає простору імен.

2) Визначено простір імен бібліотеки розширення. Створено відповідний підкаталог у директорії з розширеннями.

3) Якщо в оригінальній бібліотеці потрібні зміни, визначається позиція файлу, в який вносяться зміни, та його залежності. Після цього створюються файли з тими самими іменами, що й розширення, але в директорії бібліотеки розширень.

4) Усі посилання в коді програми на бібліотеки розширення замінюються посиланнями на розширення.

Часто виникає необхідність інтеграції розроблюваного програмного продукту з будь-яким стороннім додатком [38]. Для вирішення цього завдання застосовується підхід, який полягає в доповненні множини атрибутів ($A_{ІНТЕГР}$) множиною атрибутів власної моделі ($A_{ВЛАСН}$). Має виконуватися вимога, щоб множини не перетиналися.:

$$(A_{ВЛАСН} \cup A_{ІНТЕГР} = \emptyset) \vee (A_{ВЛАСН} \cup A_{ІНТЕГР} = \{id\}).$$

У даного методу є істотний недолік − стороні вебзастосунки не будуть заповнювати атрибути з множини $A_{ВЛАСН.}$

3.2 Вдосконалений метод модульної збірки вебзастосунків на основі ОРП

В існуючому методі модульної збірки вебзастосунків на вхід передається опис компонентного складу, на основі якого будується формалізований список цільових модулів з залежностями. Далі цей список використовується для пошуку модуля в одному або декількох репозиторіях з подальшим формуванням файлової композиції цільового вебзастосунку [13].

Процес модульної збірки відбувається шляхом виконання трьох основних етапів, пов'язаних з рівнем представлення: процесний (формування графа, що визначає функціональні особливості програми), компонентний (етап формування модулів з їх залежностями, що реалізують логічний рівень процеси), структурний (формує план, згідно з яким необхідні модулі будуть імпортовані і розміщені в структурі проекту).

Розроблено вдосконалений метод модульної збірки вебдодатків (рис. 3.2), що відрізняється тим, що він дозволяє відновлювати набори процесів і модулів згідно їх часткового опису. Він включає три стадії: 1) вибір процесів; 2) вибір модулів; 3) формування файлового складу вебзастосунку.

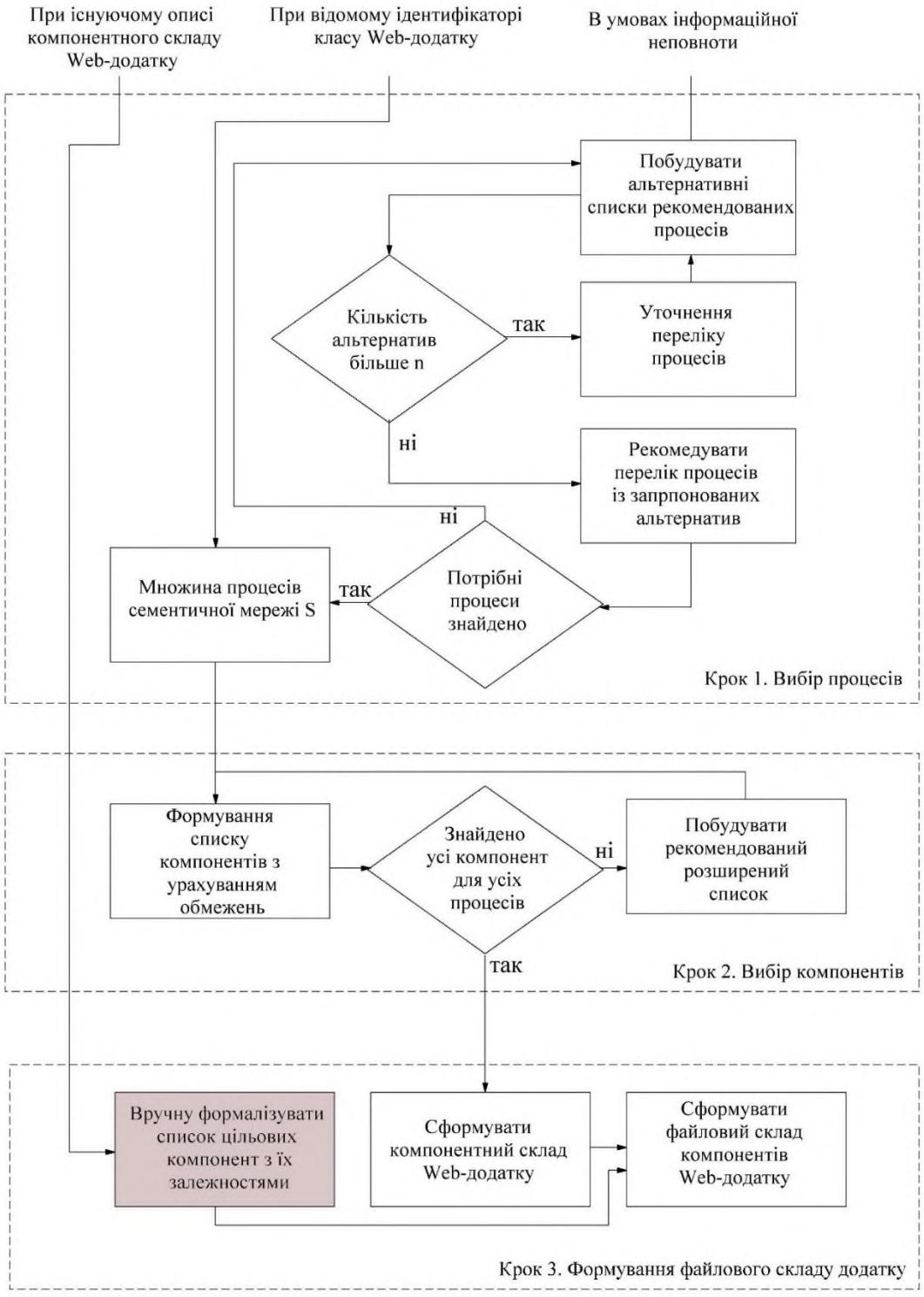


Рисунок 3.2 - Вдосконалений метод проектування вебдодатків

Спочатку (вибір процесів) перевіряється тип запиту.

Якщо метод переноситься на підмножину процесів, його інтерактивне застосування виконується на основі ідеї «побудови множин» та вибору асоціативних правил шляхом модифікації алгоритму FPG (генерація часткових шаблонів) з побудовою FP -дерево [28] пристосоване для роботи із запропонованою структурою. моделі.

На кожному кроці ітерації формується набір рекомендованих наборів процесів, які уточнюються.

На другому кроці (вибір модуля) створюються надзвичайні підмножини, для елементів яких визначаються потоки даних. На кожній ітерації формується перелік модулів з урахуванням обмежень: операційна система, платформа розробки, вартість. Якщо список модулів був побудований, то перехід до третього кроку, який полягає у формуванні файлової композиції веб-програми.

3.3 Висновки

Узагальнюючи вміст розділу, можна зробити наступні висновки:

1. Створено алгоритм двоетапного завантаження веб-додатку за пропонованою архітектурою.

2. Запропоновано метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти. Він застосовується для модульної збірки веб-додатків, що включає три етапи: 1) вибір процесів; 2) вибір модулів; 3) формування файлового складу веб-додатку.

3. Для організації механізмів введення / виведення була використана агентна схема управління запитами до бази знань стандартизованого профілю ІС/ Крім того, запропонований алгоритм перетворення даних про процеси і компоненти вебзастосунку із загальної моделі, яка використовується для опису предметної області на стороні сервера додатків в модель ОРП

4 **ЗАСТОСУВАННЯ МЕТОДІВ РОЗРОБКИ ВЕБЗАСТОСУНКІВ**

4.1 Проєктування вебзастосунку

Основною метою практичної розробки веб-додатку є задоволення запитів користувачів на інформацію про об'єкти предметної області, розміщення інформації про власні об'єкти, класифікація, модифікація (зміна / видалення), а також здійснення ефективної взаємодії між користувачами через віртуальний торговий майданчик для спілкування.

До головної сторінки ставляться такі вимоги:

1) Неавторизований користувач який вперше потрапив на головну сторінку вебзастосунку, повинен чітко розуміти призначення ресурсу і способи користувацького впливу.

2) На головній сторінці повинні бути представлені більшість типів об'єктів предметної області, оскільки користувач складає уявлення про веб-програму на основі операцій, що надаються йому над об'єктами предметної області;.

3) Аутентифікований користувач повинен отримати візуальне підтвердження успішної автентифікації.

Додаток орієнтований на платформу LAMP на базі операційної системи сімейства Linux. Тому основною ОС була обрана Ubuntu Linux Server Edition 16.04 64bit. Програмний Nginx, через підвищену швидкості обробки запитів. В якості БД застосовувалася MySQL Додаток розроблявся за допомогою інтерпретатора мови PHP ≥ 7.0.24 в формі бінарного пакета php7.0-fpm для роботи в FastCGI режимі.

Предметна область вебзастосунку, розроблена в рамках магістерської роботи, представлена реальними об'єктами, які беруть участь у процесі торгових відносин між фізичними особами (користувачами) та юридичними особами (компаніями). Предметом відносин є товари та замовлення на купівлю, які є логічною протилежністю товару.

Контролери − це файли виду *Controller.php в папці /Controller модуля, які відповідають за обробку звернень користувачів, виклик функції моделей. При цьому контролери розділені по набору функцій. Контролер за замовчуванням називається так само, як і модуль, наприклад для модуля «Checkout» - CheckoutController.php.

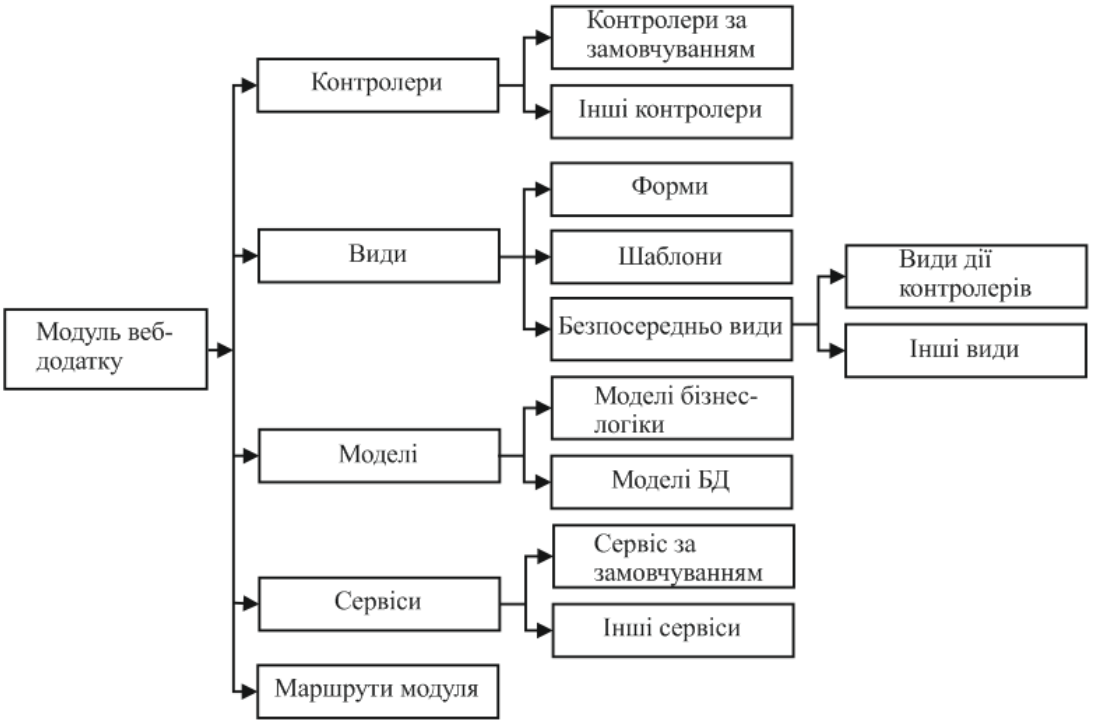


Рисунок 4.1 – Загальна структура модуля вебзастосунку

Представлення відповідають за відображення інформації для користувача або іншого програмного забезпечення.

Шаблони - це представлення першого порядку, тобто всередині них можна обробляти інші типи.

Форми - це класи, які відповідають за перетворення форми на сторінку та перевірку введених даних на стороні сервера.

Моделі поділяються на два типи - основні та логічні моделі предметної області.

Сервіси призначені для вивантаження моделей.

Всі адреси сайтів та посилання на них формуються за допомогою системи маршрутів. Найчастіше формування посилань відбувається у файлах-поданнях (поданнях) наступним чином.

```
<?php echo $this->url(['module,=>'catalog','controller'=> 'product', 'action' =>'view'])?>
```

Для цього викликається функція-хелпер url PHP з двома параметрами., що дозволяє при зміні в одному файлі з маршрутами автоматично змінювати всі пов'язані з ними посилання на ресурсі.

Товари, заявки на покупку та інші динамічні об'єкти предметної області розділяються за тематичними категоріями (рис. 4.2).

$$ACATEGORY = \{id, parent\_id, level, products\_count, leads\_count, name\},$$

де name – масив перекладів назв категорій.

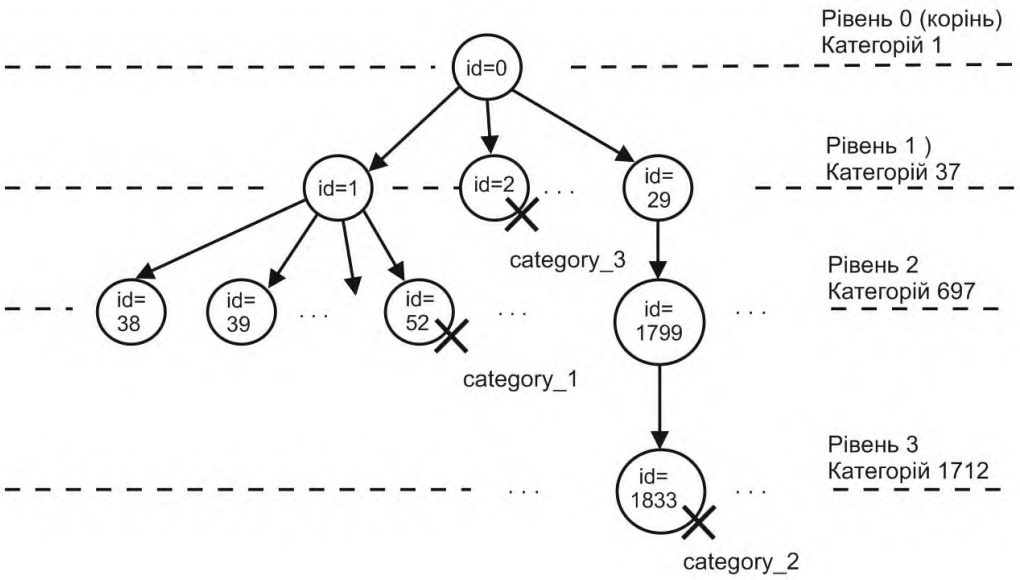


Рисунок 4.2 – Відображення різних категорій в дереві системи

Категорії організовані в трирівневому дереві за полями id, parent_id. На першому рівні існує одна категорія - корінь. На другому рівні 697 категорій, на третьому - 1712. Всього на дереві 2447 категорій. Багато моделей веб-додатків

мають посилання на дерево категорій. Модель товару має три взаємозв'язки з моделлю категорії.

Методика двоступеневого контролю доступу має наступні атрибути:

$$A_{\text{ПРАВИЛА\_ДОСТУПУ}} = \{id, module, controller, action, comp, comp\_owner\_key, comp\_group\_key, guest\ (registered/director/manager/administrator)\_allow, owner\_allow, group\_allow, deleted\_at, deleted\}.$$

Кожен кортеж трансформується в одне правило контролю доступу.

Об'єктами, які є товари (product), атрибути яких визначені в такий спосіб:

$$A_{\text{ТОВАРИ}} = \{id, company\_jd, country\_\text{id}, measure\_\text{id}, currency\_\text{id}, workflow\_id, original_{language}, category_1, views_{count}, articul, is_{taxable}, is_{retail}, price, processing_{cost}, is_{califorprice}, discount_{percentage}, weight, minimum_{order}, created\ (updated/deleted)\_at, deleted, Name, Description\},$$

Логічною протилежністю моделі товару є модель заявки.

$$A_{\text{ЗАЯВКИ}} = \{id, company_{id}, workflow_{id}, original_{language}, cateogory(l/2/3)\_id, views\_count, created\ (updated/deleted)\_at, deleted, Name, Description\},$$

В результаті магістерського дослідження вдалося створити шаблон проектування «ОРП», систему об'єктно-реляційного відображення на його основі, спроектувати систему, багатомодульний вебзастосунок, який інтегрований з різними сервісами.

Система має наступні характеристики:

1) Основна БД має 28 таблиць і містить більше 40 000 записів.

2) Основний додаток має 9 модулів, що містять більше 800 файлів об'ємом 3,1 Mb вихідних кодів

3) Реалізована інтеграція з наступними зовнішніми сервісами, через які може здійснюватися авторизація: Google, Facebook, Twitter.

4.2 Дослідження запропонованого підходу

Для того щоб впровадити запропонований метод, потрібно дослідити його ефективність, а саме їх вплив на швидкодію вебзастосунку, уніфікацію, стандартизацію і підвищення швидкості розробки. Під підвищенням ефективності розуміється:

1) Покращення кількісних характеристик програмного коду додатку.
2) Економії трудовитрат.
3) Покращення якісних характеристик (ISO 9126) коду додатку.
4) Збільшення швидкості обробки HTTP-запитів.

Для тестування було відібрано 15 вебдодатків з відкритим вихідним кодом. Серед них були вебзастосунки з різною архітектурою, з яких відібрано вебдодатки архітектури MVC на базі Zend Framework: bizsense, tomatocms, pimcore. Потім для MVC – графіків (як на рис 2.8) відібраних застосунків, обраховувася коефіцієнт подібності за допомогою ПЗ Image Comparer 3.8 і Scan Graphics 1.0.

Дослідження показали, що застосування запропонованих методів дозволяє збільшити кількість повторно використаного коду на 18%. Як результат, може відбутися зменшення обсягу роботи, бюджету та збільшення швидкості розробки вебдодатку.

Застосування методу структурного синтезу, розробленого в даній дипломній роботі, на 30% зменшується середній обсяг методу класу. Це покращує легкість читання коду, підтримку, виправлення помилок, зміну.

Для проведення формування характеристик був застосований метод експертної оцінки. В якості експертів для оцінки виступили фрілансери загальною кількістю 10 осіб. Їм було запропоновано оцінити показники якості (по ISO / IEC 9126) кожного з 15 вебдодатку за 10-бальною градацією по ДСТУ 2850-94. Параметрами оцінки були:

- функціональність F;
- надійність;

- зручність U;

- ефективність;

- супроводжуваність;

- універсальність.

Експертні оцінки наведені в Додатку Б. Середнє значення експертних оцінок обраховувалося за формулою:

$$X_{\text{СЕРЕДНЄ}} = \frac{\sum_{i=1}^{n} Xi}{n}.$$

В результаті, експертний аналіз показав, що застосування методів, запропонованих в магістерській роботі, покращує якісні характеристик вебдодатку (по ISO/IEC 9126) на 19%.

Проводилося 2 стадії тестування. Спочатку проводилося локальне тестування швидкодії вебдодатку (не враховувалися мережні затримки проходження сигналів).

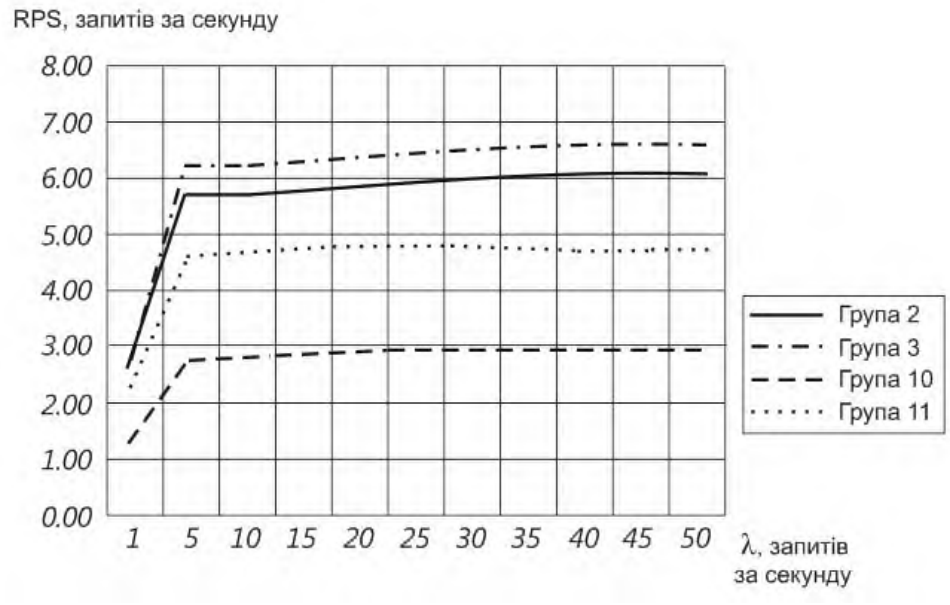


Рисунок 4.3 – Залежність швидкості обробки HTTP-запитів на локальному сервері від щільності потоку

На наступному кроці проводилося дистанційне тестування (рис. 4.4), при якому запити до вебдодатку, розташованому на віддаленому сервері.

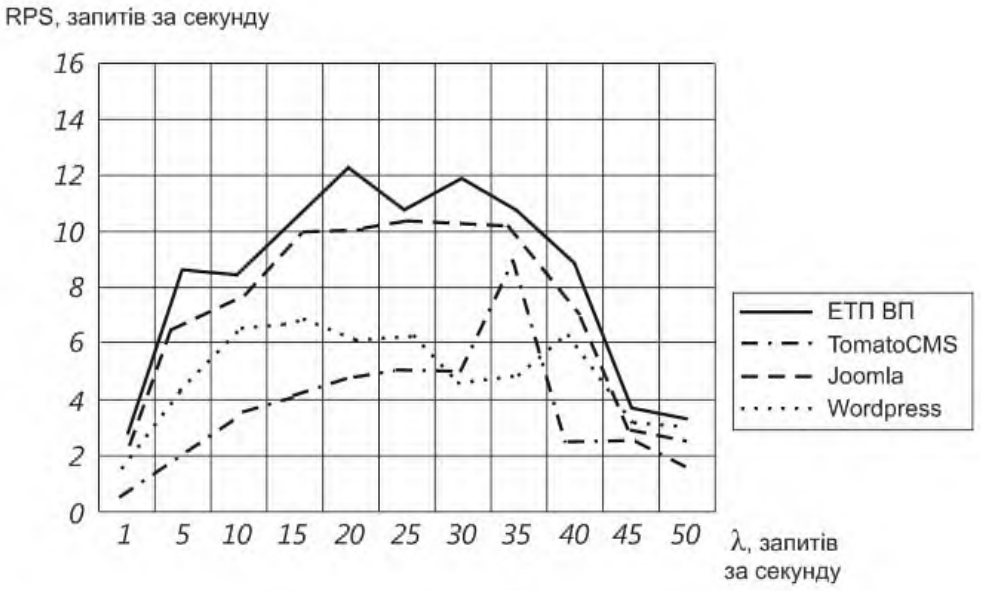


Рисунок 4.4 – Швидкість обробки HTTP-запитів на віддаленому сервері

У всіх тестах не враховувався час підключення і семантичного розбору файлів, а також інших підготовчих операцій. Тестування кожної ОРП-системи відбувалося окремо (рис. 4.5), щоб виключити їх вплив один на одного, а також порядок запуску тестуючих функцій.

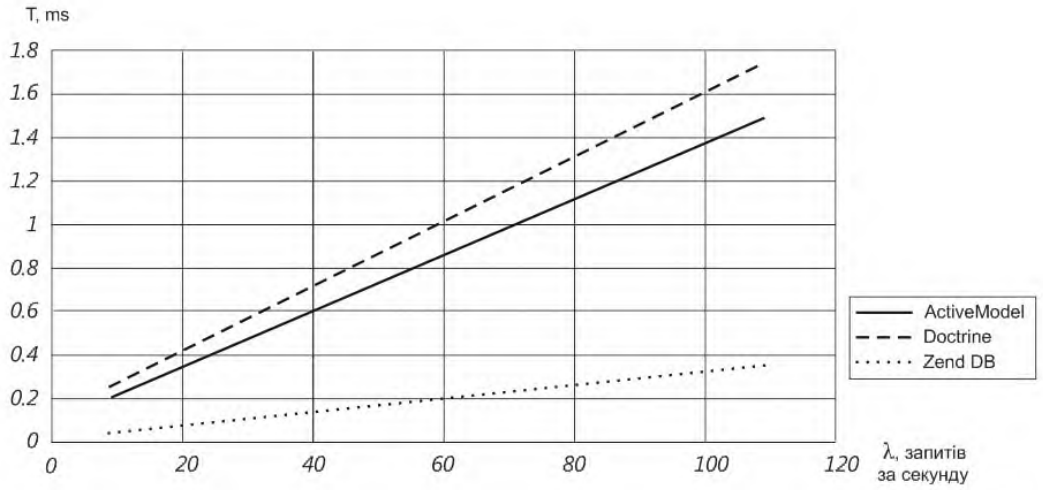


Рисунок 4.5 – Залежність часу обробки запиту для різних ОРП-систем

По кожному тесту проводилося п'ять спроб, а потім обчислювався середній час виконання.

За результатами випробувань можна зробити висновок, що швидкість роботи на досліджуваному зразку зросла на 19-25%. Обсяг зайнятої пам'яті зменшився на 16-24% порівняно з підходом Doctrine. Збільшення швидкості було досягнуто за рахунок зменшення кількості запитів SQL до бази даних, а також операцій синтаксичного аналізу запитів, взаємодії мережі та інших накладних витрат.

4.3 Висновки

Розроблені методи знайшли своє практичне застосування у вебзастосунку для моделі електронної торгової платформи

Тестування розробленого вебзастосунку показало, що значення цільової функції метрик на 13% перевищує максимальне значення цієї функції для інших веб-додатків. Запропоновані методи збільшують кількість багаторазового коду на 18%, що зменшує обсяг роботи, бюджету та збільшує швидкість розробки веб-додатків. Якісні характеристики веб-додатків (відповідно до ISO / IEC 9126) за даними експертного аналізу збільшуються на 19%.

ВИСНОВКИ

Метою магістерської роботи полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу.

У роботі отримано наступні основні теоретичні та практичні результати:

1. Проведено аналітичний огляд уже існуючих готових рішень для веб-проектів. Після визначення недоліків і постановки задачі їх вирішення, вебзастосунок почав розглядатися, з точки зору інформаційної системи. Було показано, що існуючі підходи до проектування інформаційних систем і моделі відкритих інформаційних систем потребують розвитку у аспекті модульної збірки. На основі огляду було виконано обгрунтування актуальності поставленої задачі і сформована ціль - побудова алгоритму і моделі для модульної збірки вебдодатків.

2. Запропоновано «ресурсний» метод організації контролерів вебзастосунків. Головна ідея методу полягає в тому, що розробка рівня контролерів проводиться, з врахуванням не моделей, і їх представлень, а на сутностях, над якими будуть проводитись операції. Ці сутності виступають ресурсами вебзастосунку, а зовнішній ПЗ, яке забезпечує інтерфейс до ресурсу буде називатися контролером ресурсу.

3. Запропонована модель «ОРП», особливістю якої є те, що n кількість відношень, які використовуються для відображення може бути більше 1, тоді, як для існуючих методик Doctrine, Propel, $n = 1$. Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність.

4. Запропоновано метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти. Він застосовується для модульної

збірки веб-додатків, що включає три етапи: 1) вибір процесів; 2) вибір модулів; 3) формування файлового складу веб-додатку.

5. Для організації механізмів введення / виведення була використана агентна схема управління запитами до бази знань стандартизованого профілю ІС/ Крім того, запропонований алгоритм перетворення даних про процеси і компоненти вебзастосунку із загальної моделі, яка використовується для опису предметної області на стороні сервера додатків в модель ОРП

6. Тестування розробленого вебзастосунку показало, що значення цільової функції метрик на 13% перевищує максимальне значення цієї функції для інших веб-додатків. Запропоновані методи збільшують кількість багаторазового коду на 18%, що зменшує обсяг роботи, бюджету та збільшує швидкість розробки веб-додатків. Якісні характеристики веб-додатків (відповідно до ISO / IEC 9126) за даними експертного аналізу збільшуються на 19%.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Адам Фримен. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов, 4-е издание / Адам Фримен – М.: «Вильямс», 2017. – 688 с.

2. Аристов А.В., Web-онтологии окружения открытой системы поддержки диалоговых сервисов [Текст]/ Д.В. Жевнерчук, В.Л. Чепкасов // Системы управления и информационные технологии: №3(53) – Воронеж: ВГТУ, 2013. – С. 50-55

3. Бахрушин В. Є. Математичні основи моделювання систем: Навчальний посібник для студентів. / Бахрушин В.Є. О.І. Михальов – Запоріжжя: Класичний приватний університет, 2015. – 224 с.

4. Буч Г. UML. / Г. Буч, А. Якобсон, Дж. Рамбо. - СПб.:Питер, 2006. - 736с.

5. Гаврилова Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский . - СПб.: Питер, 2016. - 384 с.

6. Гриценко В.И. Поисковый сервис. Проблемы, технологии, перспективы / В.И. Гриценко, К.К. Духновская, А.А. Урсатьев // УсиМ - 2016. - №2. - С. 81-92.

7. Джулій В.М. Методи та алгоритми розробки web-додатків / В.М. Джулій, Ю.О. Гунченко, Д.В. Чешун // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – К.: ВІКНУ, 2017. – Вип. № 56. – с.107-115.

8. Джулій В.М. Методи аналізу та синтезу розробки web-додатків / В.М. Джулій, Д.В. Чешун // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – К.: ВІКНУ, 2017. – Вип. № 57. – с.127-137

9. Жежнич П.І. Основні правила побудови семантично відкритих інформаційних систем /П.І. Жежнич, Р.Б. Кравець, В.В. Пасічник, А.М. Пелещишин // Інформаційні системи та мережі: Вісник Нац. ун-ту “Львівська політехніка - 24 1999. - №383. - С. 84-95.

10. Жежнич П.І. Семантично відкриті інформаційні системи /П.І. Жежнич, Р.Б. Кравець, В.В. Пасічник, А.М. Пелещишин //Інформаційні системи та мережі: Вісник Нац. ун-ту "Львівська політехніка". 1999. - №383. - С. 73-84.

11. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектноориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб: «Питер», 2007. – 366 с.

12. Ландэ Д.В. Поиск знаний в Интернет / Д.В. Ландэ. - М.: Изд. дом "Вильямс", 2005. - 272 с.

13. Муляр І.В. Аналіз підходів до структурної збірки Web- додатків // Муляр І.В. / Тзи доповідей Всеукраїнської науково-практичної конференції молодих вчених, ад'юнктів, слухачів, курсантів і студентів; Молодіжна військова наука у Київському національному університеті імені Тараса Шевченка" 28квітня 2017 року. Тези доповідей XII Міжнародної науково-практичної конференції; Військова освіта і наука: сьогодення та майбутнє . 28 квітня 2017.– К. : ВІКНУ, 2017, с 96

14. Мілер В.П. Метод оцінки ефективності функціонування вузла зв'язку корпоративної мережі з врахуванням інформаційної безпеки / В.В. Рикун, В.С. Орленко // «Інтелектуальний потенціал – 2020» - збірник наукових праць молодих науковців і студентів / Колектив авторів – Хмельницький: ПВНЗ УЕП, 2020. – Частина 2. С. 56-60

15. Мілер В.П. Методи проектування захищенних вебдодатків/ К.В. Молодецька, В.І. Чорненький, А.А. Берназ В.П. Мілер // Тези доповідей XVI Міжнародної науково-практичної конференції "Військова освіта і наука: сьогодення та майбутнє" Том 2 [Текст] / за заг. редакцією Ігоря Толока. – К. : ВІКНУ, 2020. – С. 49-50

16. Огнєвий О.В. Метод структурного синтезу web - додатків //О.В. Огнєвий, С..О. Барабаш / Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – К.: ВІКНУ, 2018. – Вип. № 61. – С.143-152

17. Олецький О.В. Застосування формальних моделей онтологій для формалізації інформаційних потоків у системах управління контентом / В. Олецький // Теоретичні та прикладні аспекти побудови програмних систем: Матер. міжнар. конф. TAAPSD'2005, Київ, 7-9 грудня 2005 р. - С 26-29.

18. Пелещишин А.М. Використання апарату абстрактних автоматів для моделювання Web-систем / А.М. Пелещишин // Інформаційні системи та мережі: Вісник Нац. ун-ту “Львівська політехніка”. - 1998. - №330. - С. 188-201.

19. Пелещишин А.М. Методи та алгоритми моделювання Web-систем / А.М. Пелещишин // Інформаційні системи та мережі: Вісник Нац. ун-ту “Львівська політехніка”. - №406. - 2000. - С. 199-211.

20. Пелещишин А.М. Побудова формальної моделі Web-системи / А.М. Пелещишин // Інформаційні системи та мережі. Задачі та методи прикладної математики. Вісник Львівського Нац. ун-ту ім. Франка. - 1998. - С. 182-185.

21. Проскудіна Г.Ю. Онтології в інформаційних системах / Г.Ю. Проскудіна., О.М. Овдій //Теоретичні та прикладні аспекти побудови програмних систем: спец. випуск Вісника Київськ. нац. ун-ту ім.Т.Г. Шевченка, 2004. - С.164-169.

22. Таненбаум Э. Современные операционные системы / Э. Таненбаум // М.Питер, 2010 – 1115с.

23. About MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/about/. 6. An Introduction to JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.info/intro.

24. About Ruby [Електронний ресурс] – Режим доступу до ресурсу: https://www.ruby-lang.org/en/about/.

25. Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless? [Електронний ресурс] – Режим доступу до ресурсу: https://rubygarage.org/blog/monolith-soa-microservices-serverless.

26. Client–server model [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Client-server_model.

27. Doctrine OPvM [Електроний ресурс]. – Режим доступу: URL: http://doctrine-project.org.

28. Griffith A. Codelgniter 1.7 professional development / A. Griffith / Packt – 2015, 300c.

29. ImageCms [Електроний ресурс] – Режим доступу: URL: http://www.imagecms.net/).

30. InstantCMS [Електроний ресурс] – Режим доступу: URL: http://www.instantcms.ru/.

31. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Model-View-Controller.

32. Noy N.,McGuinness D. Ontology Development 101: A Guide to Creating Your First Ontology// Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical informatics Technical Report SMI-2001-0880, March 2001.

33. Ruby on Rails 2.1 - Unit Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/ruby-on-rails-2.1/rails-unit-testing.htm.

34. Integration Testing: What is, Types, Top Down & Bottom Up Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/integration-testing.html.

35. Software Testing Overview [Електронний ресурс] – Режим доступу до ресурсу:https://www.tutorialspoint.com/software_engineering/software_testing_overview.html.

36. Suren A. pOWL - A Web Based Platform for Collaborative Semantic Web Development //powi.sourceforge.net/swc/powl_usage.pdf Semantic Web Development Platform - pOWL, www.powLsourceforge.net.

37. WHAT IS A GEM? [Електронний ресурс] – Режим доступу до ресурсу: https://guides.rubygems.org/what-is-a-gem/.

38. Yii Framework 2.0 API Documentation [Електронний ресурс]. – Режим доступу: URL: http://www.yiiframework.com/doc-2.0/.

39. Zend Framework: Documentation: ZendDb [Електронний ресурс]. – Режим доступу: URL: https://framework.zend.com/learn.

ДОДАТОК А

(обов'язковий)

Експертні оцінки якості

Таблиця А.1 – Значення експертних оцінок якості

ВД	F	R	U	E	S	P	Ц
drupal	4	5	7	4	7	5	32
ЕТП ВД	8	7	9	9	10	6	49
lifestreet	6	6	6	6	4	4	32
joomla	5	2	6	1	6	3	23
bizsense	7	8	3	7	5	7	37
modx	5	3	4	5	5	5	27
tomatocms	7	8	7	4	7	8	41
xoops	5	5	2	6	3	7	28
instantCMS	4	6	5	5	4	6	30
imageCMS	3	3	4	4	6	4	24
joostine	5	4	4	5	3	1	22
typo3	3	4	6	4	3	2	22
luta	4	3	3	2	2	2	16
wordpress	3	8	7	5	7	7	37
nextgen	4	4	2	4	3	3	20
pimcore	4	7	5	7	8	7	38

ДОДАТОК Б

(обов'язковий)

Копії наукових праць

Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення

Мілер В.М., Орленко В.С.
Хмельницький національний університет

З метою покращення розробки вебдодатків проектувальники намагаються знайти рішення, яке дозволить швидко і ефективно використовувати компонентну базу існуючих бібліотек.

Найпоширенішою платформою для розробки вебдодатків є LAMP [1]. LAMP - абревіатура набору вільного ПЗ з відкритим кодом, в який входять ОС Linux, веб-сервер Apache, СКБД MySQL, та інтерпретатор Perl/PHP/Python - основні компоненти для побудови життєздатного багатоцільового вебсервера

В процесі роботи пропонується метод, який здійснює реляційно-об'єктне та об'єктно-реляційне перетворення (ОРП) таблиць бази даних у копіях класів базових моделей програмного коду. В даний час є розробки подібних систем, таких як Doctrine, Propel [2], але вони мають суттєвий недолік - великий обсяг програмного коду і надлишок реалізованих функцій. Наприклад, Доктрина включає більше 100 класів. Це спричиняє проблему з продуктивністю, оскільки інтерпретатор мови повинен завантажити значну

кількість файлів. Крім того, на об'єкти, отримані за допомогою ОРП, існує ряд обмежень:

1) Вони можуть знаходитися тільки в одній таблиці. Таким чином, проєктування об'єкта додатка зводиться до композиції об'єктів ОРП, що збільшує трудомісткість розробки.

2) Вони знаходиться в єдиній базі даних. Це випливає з попереднього пункту. Неможливо створити об'єкт, розподілений по декільком БД.

3) Мають фіксований набір властивостей. Зокрема, кількість колонок в таблиці має дорівнювати кількості властивостей об'єкта;

4) Ускладнена багатомовна підтримка, оскільки формується єдина таблиця, в якій частина стовпців "відповідає" за різні мови, а після вибірки необхідно фільтрувати дані,.

Враховуючи переваги використання систем PDO та виходячи з недоліків існуючих систем, необхідно сформулювати цілі розробленої системи PDO:

1) Адекватне відображення предметів та їх зв'язків. Основною функцією PDO є відображення об'єктів предметної області веб-програми в реляційній структурі бази даних, а також відображення взаємозв'язків об'єктів. Це може призвести до низки проблем. Наприклад, об'єкт може зберігатися в багатьох таблицях безлічі баз даних на різних серверах. Об'єкти можуть бути різними за своєю будовою. Наприклад, нові елементи в процесі розробки можуть бути додані до об'єкта профілю користувача, а нові властивості можуть з'являтися в об'єкті.

2) Підтримка багатомовності. При розробці веб-додатків часто доводиться підтримувати кілька мов. Багатомовна підтримка повинна буги прозорою, тобто розробник повинен бути впевнений, що об'єкт зараз відображається поточною мовою для користувача. Але в той же час система повинна мати достатню гнучкість, щоб змінити мову відображення на таку, яку вимагає розробник.

3) Розроблені можливості пошуку. Система повинна підтримувати складні запити, наприклад, "знайти електронний лист першого менеджера компанії, який розмістив останній продукт".

В рамках даного дослідження було запропоновано систему об'єктно-реляційного відображення «Активна модель», яка вирішує вищевказані завдання. Для здійснення реляційно-об'єктних -перетворень необхідно на стороні об'єктної моделі реалізувати декларативний опис реляційної моделі. Для цього зараз використовуються мови опису об'єктів (*ODL - Object Definition Language*). Вони призначені для таких потреб:

1) Визначення схеми бази даних.

2) Забезпечення універсальності опису схеми бази даних. Можна змінити базу даних на іншу, тоді як опис на ODL не зміниться. ODL використовується як "загальний знаменник" при описі схеми бази даних.

3) Трансформація типів даних, тобто генерування типів зі схеми бази даних певною мовою програмування, з якої планується доступ до неї.

У свою чергу, мови сімейства OQL (Object Query Language) використовуються для реалізації RO-перетворень. Це мови запитів об'єктів, декларативні мови доступу до бази даних, подібні до мови SQL для баз даних. Стверджується, що вирази в OQL на 90% сумісні з синтаксисом оператора select з SQL'92 [2]. Відмінності між OQL та SQL полягають у тому, що вхідними даними запитів OQL є об'єкти, а не таблиці. Конструкція select-from-where використовується для написання запиту, як у SQL. Результатом запиту, як правило, є набір об'єктів - набір. Тоді цей набір можна перетворити на список, масив. Набір може оброблятися в циклі та отримувати окремі його елементи - об'єкти бази даних повністю або набір значень з бази даних у вигляді будь-якого типу мови програмування. Таким чином, взаємна модельна трансформація складається з інтерпритації OQL. В існуючих системах (наприклад, в Doctrine) модельне перетворення включає в себе такі операції як парсинг OQL-запиту, валідацію, кешування, перетворення OQL в SQL, виконання SQL, отримання «сирих» даних, гідрацію (перетворення сирих даних в об'єктний вид). З цим пов'язані значні витрати обчислювальних потужностей сервера, тому в даній роботі пропонується ряд змін, спрямованих на збільшення продуктивності операцій перетворення:

1) Зведення ODL до діалекту основної мови розробки. Для досліджуваної платформи LAMP – це PHP.

2) Визначення кінцевої множини перетворюваних типів і їх відображень.

3) Використання діалекту основної мови платформи для здійснення OQL-операція, а інтерпретатора платформи – для інтерпретації OQL команд.

4) Визначення спеціальних типів таблиць і їх ODL.

Діалект MII PHP, який використовується в якості ODL, буде називатися AMDL (ActiveModel Definition Language), а діалект, який використовується в якості OQL – AMQL (ActiveModel Query Language). Результат модельної трансформації на стороні об'єктної моделі буде називатися AM (ActiveModel).

Перед визначенням AMDL і AMQL-діалектів необхідно визначити структури, які підлаються РО перетворенню. Як відомо, інформація в РСУБД організована у вигляді множини таблиць (сукупності схем відносин і даних). У даній роботі пропонується класифікація схем на три типи: T_C – звичайна (classic), T_P-схема додаткових полів (x-properties), T_F – схема прапорів (x-flags) і T_{FT} – схема прапорів (x-flags-temporary) з темпоральною валідацією.

Реляційна модель задається наступним чином. Нехай A_1, A_2, ..., A_n імена атрибутів. Кожному імені атрибута A_i відповідає допустима множина значень, які може приймати атрибут A_i. Це множина значень D_i називається доменом атрибуту A_i, $i = 1, n$. За визначенням, домени є непорожніми кінцевими або злічинними множинами. Поняттю домену D_i відповідає множина значень, що знаходяться в стовпці A_i розглянутої таблиці [4].

Схемою відношення $R(A_{R1}, A_{R2}, ..., A_{Rn})$ називається кінцева множина імен атрибутів $\{A_{R1}, A_{R2}, ..., A_{Rn}\}$, причому атрибут A_i приймає значення з множини D_{Ri} (i = 1, 2, ..., n), де n - розмірність відношення.

Об'єктна модель задається наступним чином. Об'єктом O називається представлення сутності предметної області, яке використовується при моделюванні. $A\ \{A_{O1}, A_{O2}, ..., A_{Om}\}$ є множиною атрибутів об'єктної моделі. Класом C називається загальна сутність, яка може бути визначена як сукупність елементів (реалізацій класу). Клас є родовою ознакою об'єктів.

Після визначення операцій перетворення типу даних можна визначити набір операцій перетворення. Об'єктно-реляційна система відображення перетворює набір кортежів (записів, рядків) даних, що складають набір значень атрибутів, у форму, з якої можуть взаємодіяти функції мови програмування PHP. У запропонованому методі (і системі ORP "Активна модель") набір кортежів відображається на екземплярі класу. Об'єктом класу "Активна модель" є сукупність відносин, отриманих в результаті дії природного зв'язку, застосованих до безлічі кортежів (обов'язково по одному для кожного зв'язку) та набору функцій (поведінки) над сукупність відносин, успадкованих від класу програмного забезпечення. Натуральним зв'язком є операція SQL NATURAL JOIN, яка повертає відношення, яке містить усі можливі кортежі (K), які є комбінаціями двох (або більше) кортежів, що належать до двох (або більше) заданих відносин, за умови, що в комбінованих кортежах є однакові значення в одному (або декількох) загальних для вихідних атрибутів (і ці загальні значення з'являються в результуючому кортежі рівно один раз).

Основна відмінність методики «Активна модель» від інших методик в тому, що n (кількість відношень, які використовуються для відображення) може бути більше 1, тоді, як для існуючих методик об'єктно-реляційного відображення на платформі LAMP (маються на увазі методики Doctrine, Propel, Yii Active Record) $n = 1$. З цього випливає, що реалізація моделі даних, подібної до «Активної моделі», вимагає створення в них n моделей. Відповідно, вартість пам'яті, необхідної для зберігання моделей даних, збільшиться за рахунок збільшення кількості зразків класів.

Перелік посилань

1. Майк Кон. Scrum: Гибкая разработка ПО. / Майк Кон. — Изд-во: Диалектика-Вильямс, 2016. — С. 576.

к.т.н., доц. Миколенко К.В. (ХНТУ)
к.т.н., доц. Чорненький В.І. (ХНТУ)
Кірнат А.А.(УВК, Ванкувер)
Мізер В.М. (ХНТУ)

Методи проектування захищених вебдодатків

Найпоширенішою платформою розробки веб-додатків є LAMP, до складу якої входить операційна система Linux, веб-сервер Apache, реляційна база даних MySQL та інтерпретатор PHP. PHP як мова програмування, що входить до складу платформи, володіє низьким порогом входження, що також збільшує поширеність платформи, проте саме це є причиною створення програмного коду, що не задовольняє вимогам якості. Разом з тим, на даній платформі розробляються багатокомпонентні програмні рішення з участю багатьох десятків розробників. Веб-додатки починають оперувати величезними обсягами даних, в тому числі і надважливими персональними даними, що зумовлює високі вимоги до програмного коду. У той же час, велика тривалість розробки, погодження та затвердження міжнародних і національних стандартів призводить до їх консерватизму, а також до хронічного відставання вимог і рекомендацій цих документів від сучасної практики та технології створення складних систем. Сучасні методи розробки програмних засобів, їх уніфікація та стандартизація, не повною мірою враховують специфіку розробки на платформі LAMP з використанням вільних інтернет-технологій. В цих умовах створення методів розробки веб-додатків є актуальною задачею.

Найперспективнішим на даний момент розробки веб-додатків є метод розробки, заснований на парадигмі «Модель-Представлення-Контролер». Однак цей метод не лишався певний недоліків, найістотнішим з яких є відсутність методів структурного синтезу програмного коду рівня моделей, рівня представлення і рівня контролерів. У зв'язку з цим поширеним явищем стало невірне трактування архітектури, що призвело до появи великої кількості не уніфікованого програмного коду, що не задовольняють вимогам якості.

Для усунення розглянутих недоліків пропонується модель архітектури веб – додатка і метод структурного синтезу архітектури, а також вихідного коду рівня моделей, рівня представлення і рівня контролерів. Синтез здійснюється методом угрупування сутностей предметної області. Сутності

-н

веб-додатка необхідно представити спеціальними класами - моделями. Сукупність моделей становить шар доступу до даних. У запропонованому методі модельний шар охоплює меншу множину функцій, що реалізовуються, ніж модельний шар в існуючій архітектурі «Модель-Представлення-Контролер» за рахунок введення сервісного шару.

ДОДАТОК В

(обов’язковий)

Презентація

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Мілер Вадим Петрович

ВДОСКОНАЛЕННЯ МЕТОДУ ПРОЄКТУВАННЯ ВЕБДОДАТКІВ НА ОСНОВІ ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ПЕРЕТВОРЕННЯ

Науковий керівник
к.т.н., доцент Орленко В.С.

кафедра кібербезпеки та комп’ютерних систем і мереж

Мета магістерської роботи полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу

Об'єкт дослідження: процес створення вебзастосунків

Предмет дослідження: методи структурного синтезу та розробки вебзастосунків.

Задачі досліджень у роботі формулюються наступним чином:

1. Провести аналіз існуючих підходів до синтезу та розробки веборієнтованих систем.
2. Визначити можливості підвищення ефективності структурного синтезу, взаємодії вебзастосунків і їх зберігання.
2. Проектування семантичної моделі процесів і розробки вебзастосунків.
3. Розробити ефективну архітектуру та алгоритм функціонування вебзастосунків.
4. Розробити метод об'єктно-реляційного перетворення для синтезу та розробки вебзастосунків, а також шаблону проектування на його основі.
5. Реалізація розроблених алгоритмів структурного синтезу вебзастосунків на базі запропонованого методу.

Наукова новизна роботи полягає в наступному:

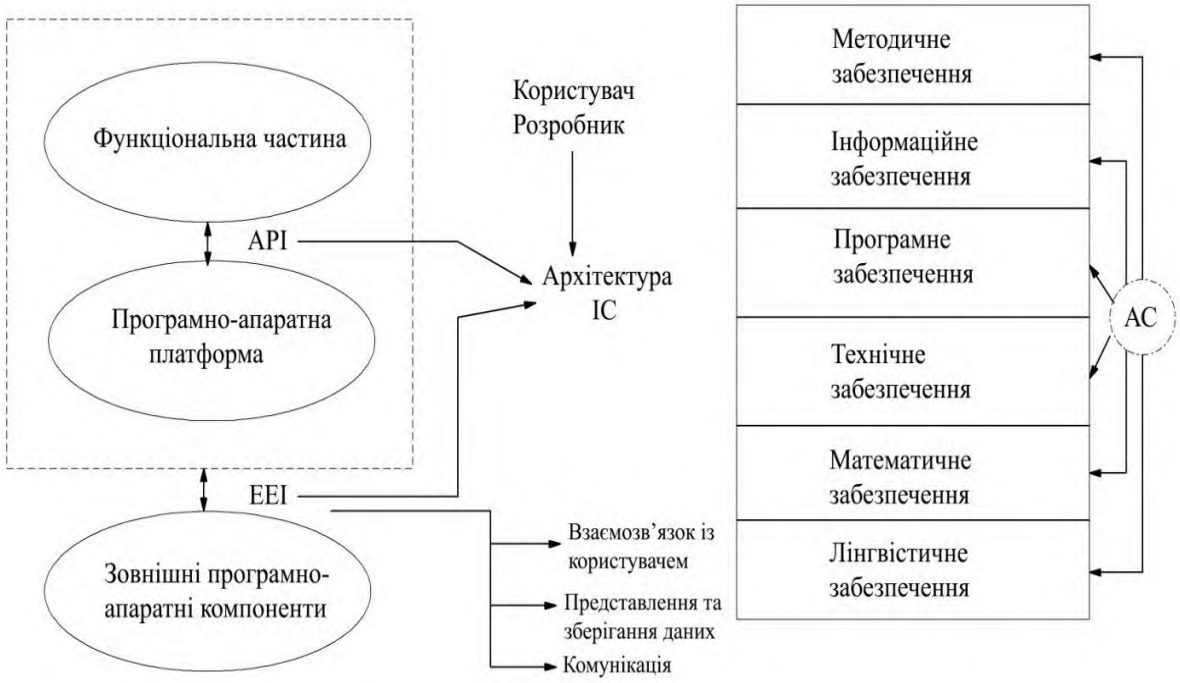
1. Вдосконалена модель архітектури вебзастосунку, в якій враховуються індивідуальні і системні властивості процесів і модулів.
2. Запропоновано метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти

Методика проведення досліджень: методи теорії системного аналізу, теорії множин, теорії реляційних баз даних, теорії алгоритмів теорії графів, і методів імітаційного моделювання

Практична цінність. Практична реалізація розроблених методик дозволила створити новий клас адаптивних автоматизованих систем, що суттєво підвищують ефективність процесу збірки вебзастосунків і покращені структурності, повторюваності, модифікованості.

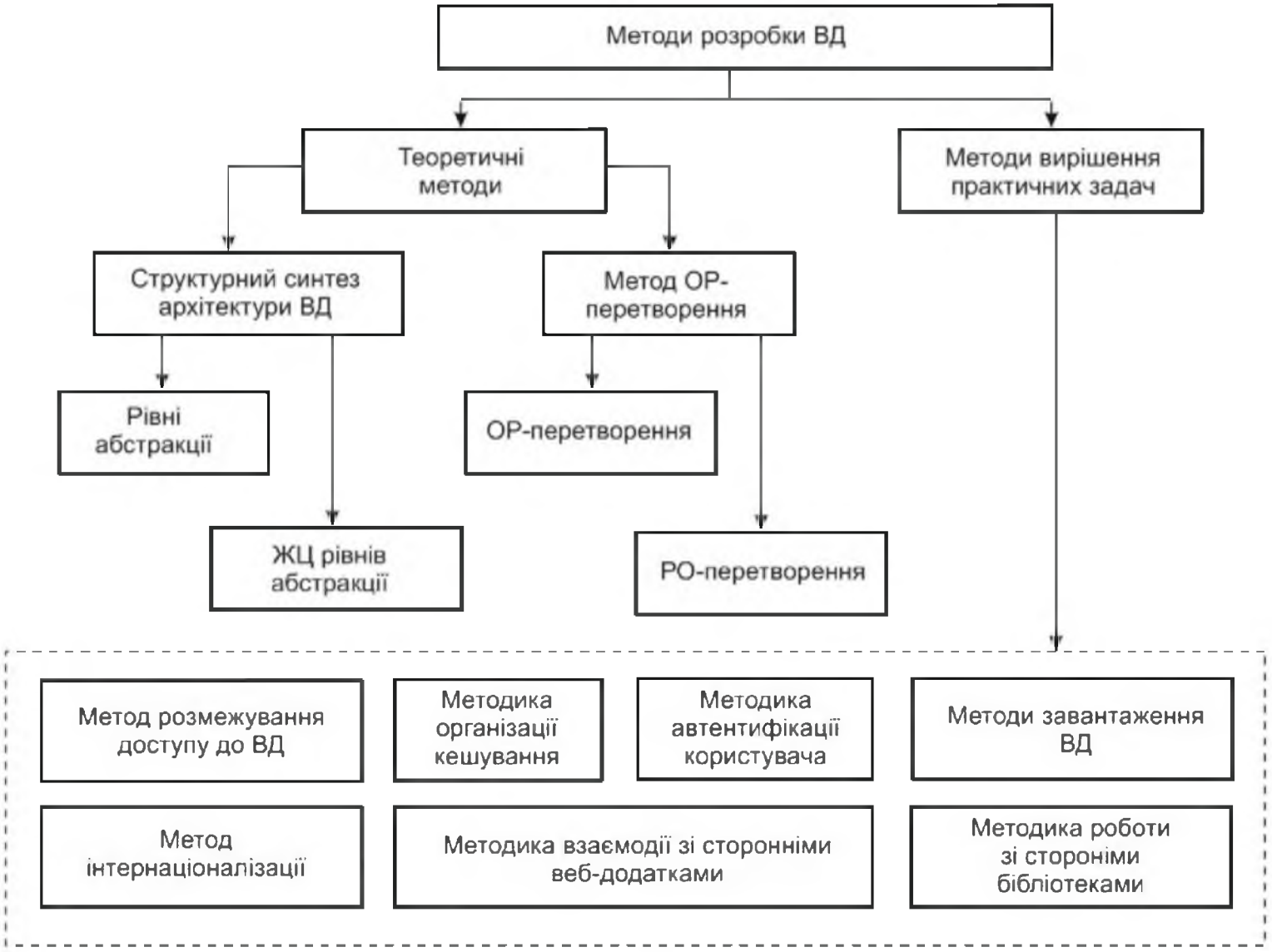
Публікації. По темі магістерської роботи опубліковано 1 стаття у нефаховому журналі (збірник НПК МНІС ІП-2020), 1 - теза доповідей на всеукраїнській конференції (Тези доповідей XVI Міжнародної науково-практичної конференції "Військоваосвіта і наука: сьогодення та майбутнє" [Текст] / за заг. редакцією Ігоря Толока.– К. : ВІКНУ, 2020)

Узагальнена структура відкритої автоматизованої інформаційної системи



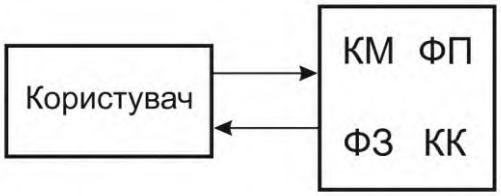
Класифікація рівнів інтеграції даних

Методи розробки вебдодатків

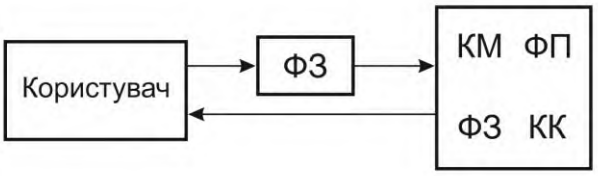


Існуючі моделі архітектур вебзастосунків

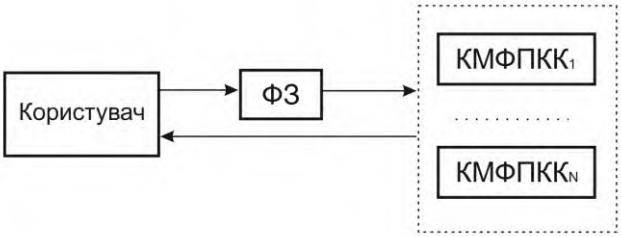
Архітектура монолітного ВД



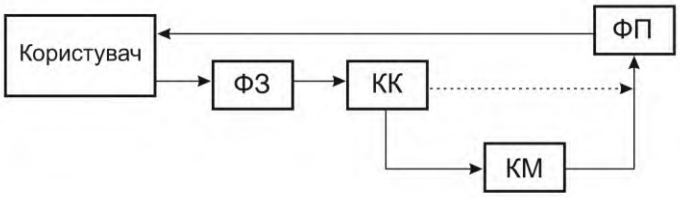
МВК з виділеним завантажувачем



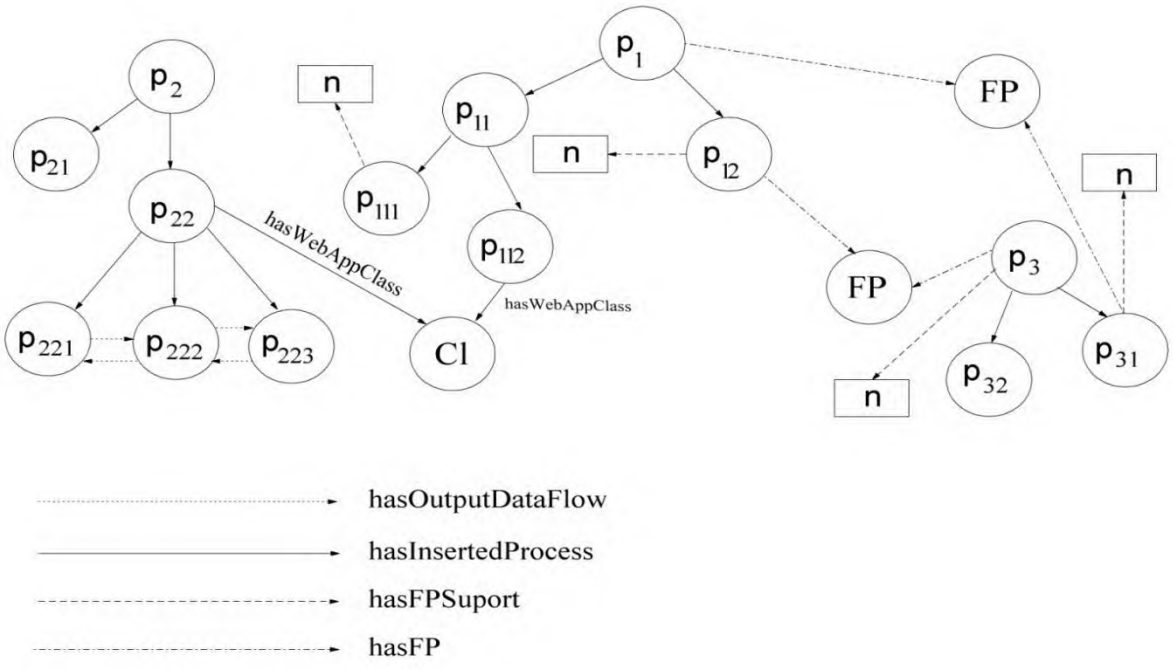
Архітектура моно-модульного ВД



Архітектура «Модель-Вид-Контролер»

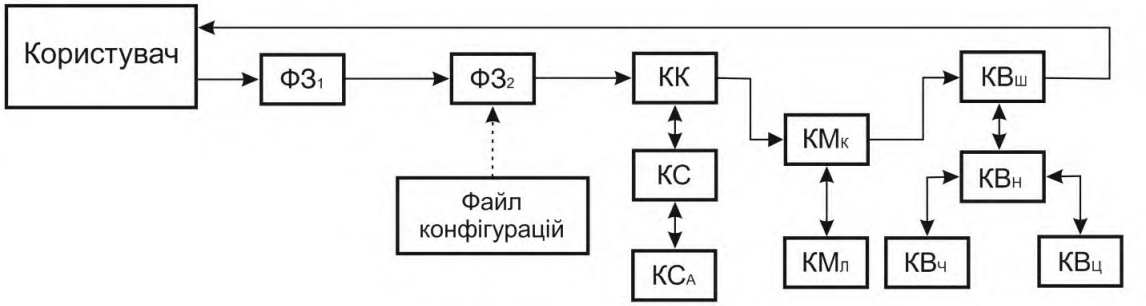


Семантична модель процесу збірки компонентів мобільних веб-додатків

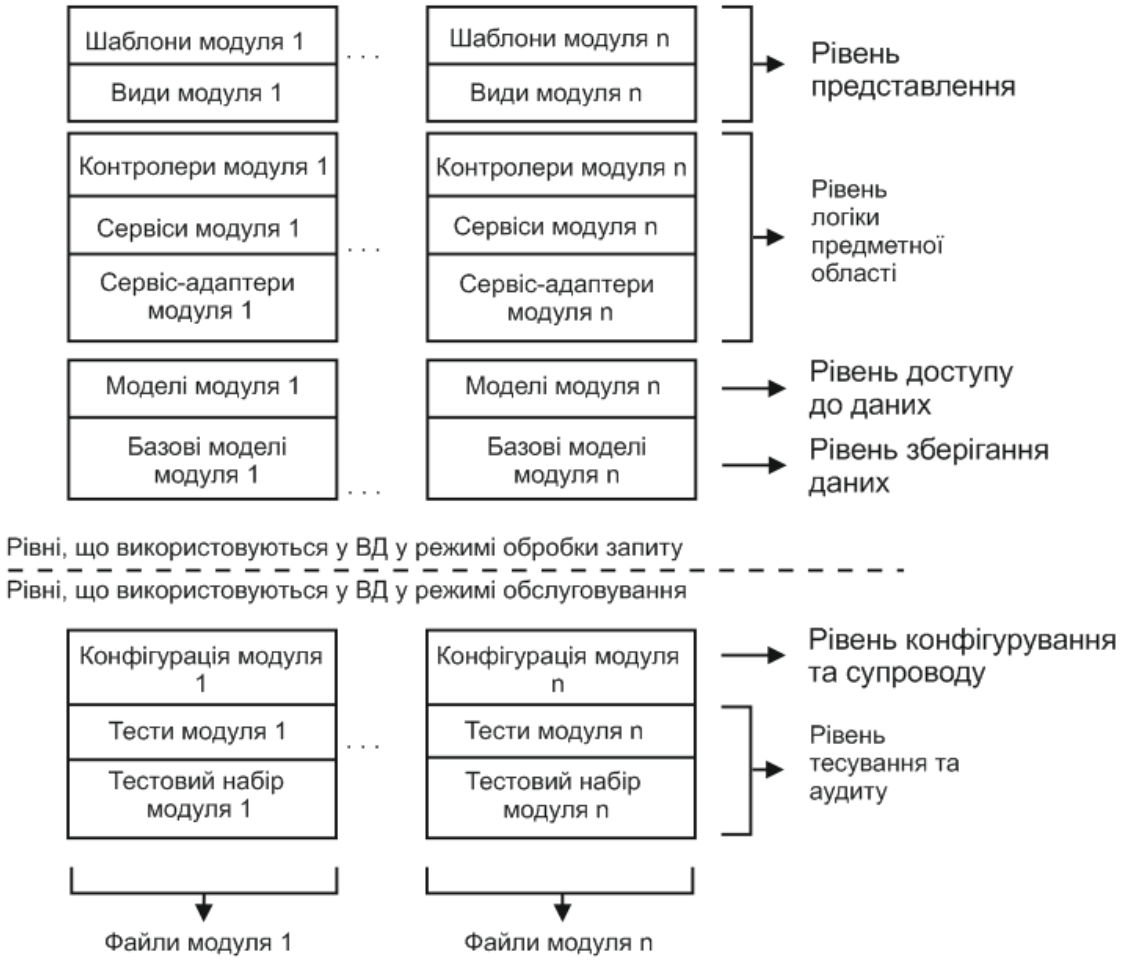


Запропонована модель архітектури

Модель архітектури вебзастосунку



Рівні абстракції в вебзастосунку



Основною перевагою запропонованої моделі архітектури є мінімізація кількості "угод щодо конфігурації", що досягається за рахунок уніфікації інтерфейсів рівнів абстракції і широкого використання шаблонів проектування (design patterns)

Метод об'єктно-реляційного перевторення

Запропонована «ОРП» модель буде виглядати так:

$$a = \left\{\left\{\mathrm{M}_{\text{відношення}}\right\}_n, \{A\}_n, B\right\},$$

де a − об'єкт класу «ОРП», n − кількість відносин, що входять в модель (n ≥ 1); $\mathrm{M}_{\text{відношення}}$− набір метаданих відносин, А− набір атрибутів відносин, В − види функцій над об'єктами класу.

Тоді логічну модель даних $\mathrm{M}_{Л}$ визначимо:

$$\mathrm{M}_{Л} = \{\bar{T}, \bar{R}\},$$

де $\bar{T}$ − набір декларативних описів схем відносшень, які належать моделі, $\bar{R}$ − набір зв'язків з іншими відношенями.

$$T_i = \{name, ct, db, \vec{C}\}; Ci = \{cname, ctype\};$$

$$R_i = \{rt, FK, DK, DM, iT, iFK, iDK\};$$

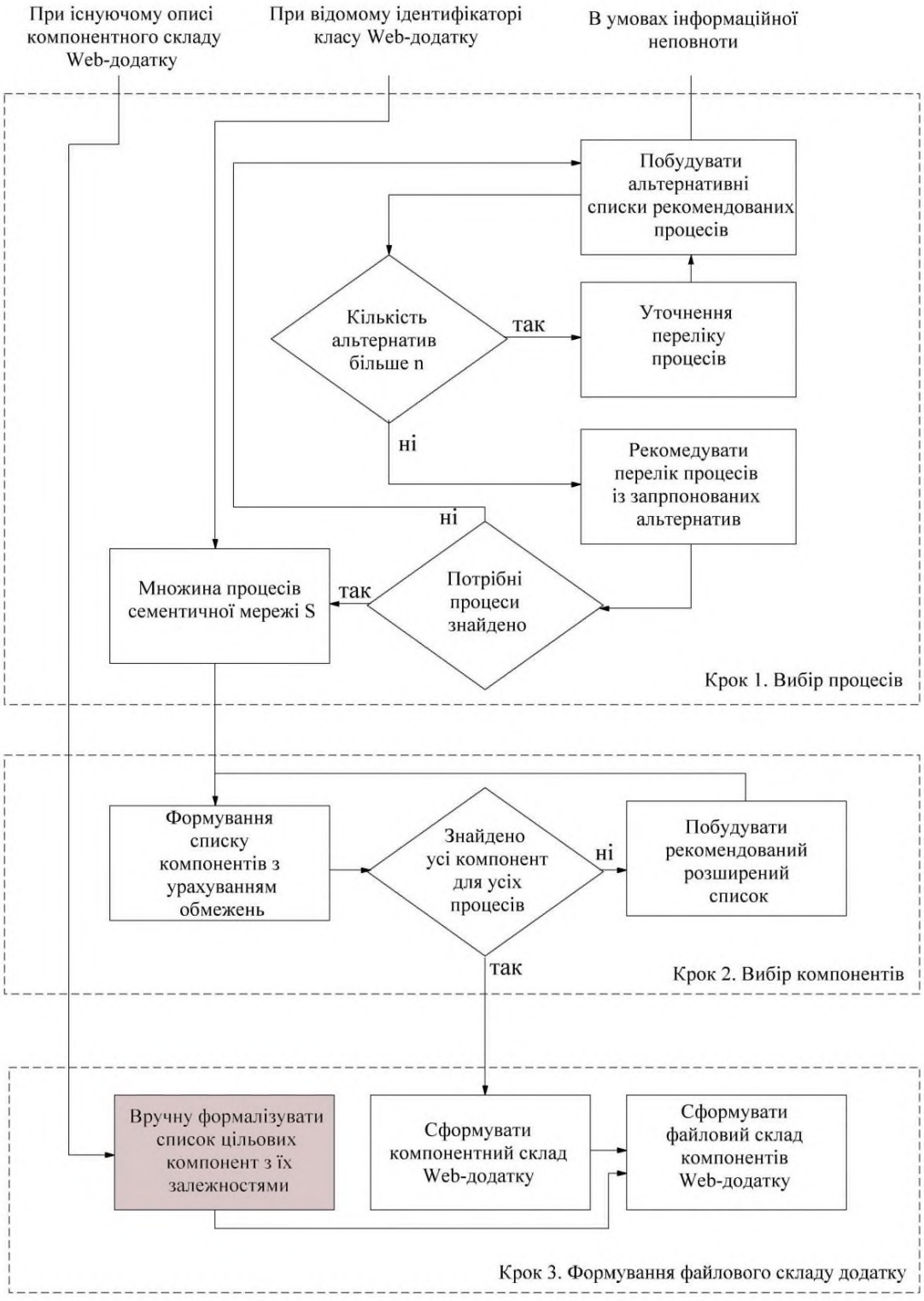
$$type \in \{T_C, T_P, T_F, T_{FT}\};$$

$$rt \in \{hasOne, hasMany, hasManyToMany, hasManyToManyExtended\},$$

де *name* − назва відношення в базі; *ct* – тип відношення; *db* − назва БД на сервері розподіленої системи керування; $\vec{C}$ – множина описів атрибутів реляційної моделі; *cname* – ім’ч атрибуту реляційної моделі; *ctype* – тип даних PHP, для цього атрибуту; *rt* – тип зв'язку; *DK* – ключ отримувача зв'язку; *FK* – зовнішній ключ джерела зв'язку; *iT* – таблиця перетинів для відношення багато-до-багатьох, *iFK* – зовнішній ключ перетинів; *iDK* – ключ отримувача зв'язку через таблицю перетинів.

Застосування запропонованої методики знижує витрати пам'яті та збільшує продуктивність рівня моделі вебдодатку.

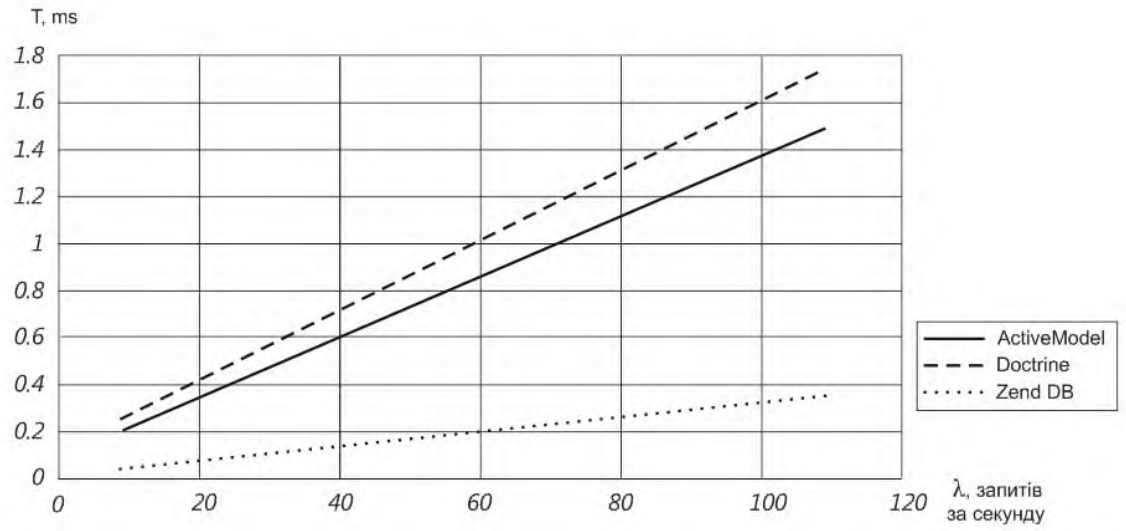
Вдосконалений метод структурного синтезу



Існує 3 варіанти роботи з алгоритмом: за описом компонентного складу, за ідентифікатором класу та в умовах інформаційної неповноціннсоті

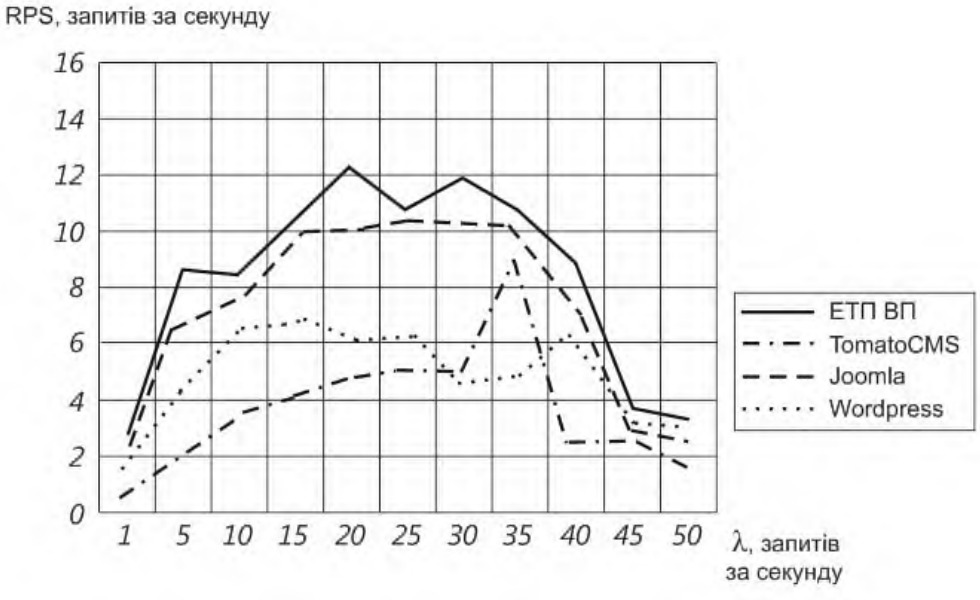
Порівняльний аналіз розробленої системи

Залежність часу обробки запиту від інтенсивності в об'єктоповертаючому режимі для різних ОРП-систем



n=	10	20	50	100
ActiveModel	0,2106	0,321	0,7194	1,383
Zend DB	0,061	0,09	0,164	0,317
Doctrine	0,262	0,407	0,904	1,642
Delta(max,AM)	0,24	0,27	0,26	0,19

Залежність швидкості обробки запитів різних вебзастосунків від щільності потоку



ВИСНОВКИ

Метою магістерської роботи полягає в підвищенні ефективності розробки і взаємодії компонентів вебзастосунків, за рахунок зменшення часових затрат і кількість помилок на етапі структурного синтезу.

У роботі отримано наступні основні теоретичні та практичні результати:

1. Проведено аналітичний огляд уже існуючих готових рішень для веб-проектів. Після визначення недоліків і постановки задачі їх вирішення, вебзастосунок почав розглядатися, з точки зору інформаційної системи. Було показано, що існуючі підходи до проектування інформаційних систем і моделі відкритих інформаційних систем потребують розвитку у аспекті модульної збірки. На основі огляду було виконано обгрунтування актуальності поставленої задачі і сформована ціль - побудова алгоритму і моделі для модульної збірки вебдодатків.

2. Запропоновано «ресурсний» метод організації контролерів вебзастосунків. Головна ідея методу полягає в тому, що розробка рівня контролерів проводиться, з врахуванням не моделей, і їх представлень, а на сутностях, над якими будуть проводитись операції. Ці сутності виступають ресурсами вебзастосунку, а зовнішній ПЗ, яке забезпечує інтерфейс до ресурсу буде називатися контролером ресурсу.

3. Запропонована модель «ОРП», особливістю якої є те, що *n* кількість відношень, які використовуються для відображення може бути більше 1, тоді, як для існуючих методик Doctrine, Propel, $n = 1$. Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність.

4. Запропоновано метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти. Він застосовується для модульної збірки веб-додатків, що включає три етапи: 1) вибір процесів; 2) вибір модулів; 3) формування файлового складу веб-додатку.

5. Для організації механізмів введення / виведення була використана агентна схема управління запитами до бази знань стандартизованого профілю ІС/ Крім того, запропонований алгоритм перетворення даних про процеси і компоненти вебзастосунку із загальної моделі, яка використовується для опису предметної області на стороні сервера додатків в модель ОРП

6. Тестування розробленого вебзастосунку показало, що значення цільової функції метрик на 13% перевищує максимальне значення цієї функції для інших веб-додатків. Запропоновані методи збільшують кількість багаторазового коду на 18%, що зменшує обсяг роботи, бюджету та збільшує швидкість розробки веб-додатків. Якісні характеристики веб-додатків (відповідно до ISO / IEC 9126) за даними експертного аналізу збільшуються на 19%.

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 1.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 9%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 81083 Название: Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення Добавлено в БД: 2020-11-24 Авторы: Мілер В.П. Руководители: Орленко В.С. Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	108462	944	2564 (2%)	30 (3%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

UNICHECK

User name:
Kafedra kiberbezpeky

Check ID:
1005326468

Check date:
02.12.2020 11:03:19 EET

Check type:
Doc vs Internet

Report date:
02.12.2020 11:33:56 EET

User ID:
100005590

File name: **Робота_Мілер_ю**

Page count: **89** Word count: **16796** Character count: **129703** File size: **5.31 MB** File ID: **1005449416**

Text modifications detected (similarity score might be affected)

7.98% Matches

Highest match: **2.18%** with Internet source (**http://science.lp.edu.ua/sites/default/files/Papers/02_136.pdf**)

7.98% Internet sources	304	..... Page 91

No Library search was conducted

0% Quotes

Exclusion of quotes is off

Exclusion of references is off

0% Exclusions

No exclusions

Modifind

Text modifications detected. Find more details in the online report.

Replaced characters	284

Suspicious formatting	15 Pages

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів ідентичності схожості:
Назва: Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення

Автор: Мілер Вадим Петрович
Спеціальність: 123 Комп'ютерна інженерія
Освітня програма: Програмування та захист комп'ютерних систем і мереж
Науковий керівник: Орленко Вікторія Сергіївна, к.т.н., доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформлені посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;
4) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності схожості, складає 7.98% і адресується до 304 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.С. Орленко

Завідувач кафедри КбКСМ, гарант ОП _____ Ю.П. Кльоц

Дата: 03.12.2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «магістр»

Магістр Мілер В.П.

Тема Вдосконалення методу проєктування вебдодатків на основі об'єктно-реляційного перетворення

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «магістр»:

кількість листів креслень 10 ; кількість сторінок записки 94

1. Короткий зміст кваліфікаційної роботи та прийнятих рішень в рамках роботи розроблено модель «ОРП», особливістю якої є те, що *n* кількість відношень, які використовуються для відображення може бути більше 1. Застосування запропонованої методики знижує витрати пам'яті (не потрібно створювати n моделей, все знаходиться в одній) та збільшує продуктивність. На основі моделі вдосконалено метод об'єктно-реляційного перетворення, для синтезу та розробки вебзастосунків, який забезпечує перетворювання таблиць сервера баз даних в програмні компоненти

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота ОС «магістр» у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині дипломної роботи.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі висвітлюється актуальність теми роботи, дається аналіз досліджуваної проблеми і обґрунтовується застосовуваний підхід до її вирішення, формулюються цілі і завдання дослідження, описується наукова новизна і практична значимість отриманих результатів. У першому розділі розглядаються питання забезпечення ефективного проєктування вебдодатків. Наступні розділи присвячені розробці моделі процесів і модулів вебдодатку, та його структурному синтезу. Розглянуто питання застосування розробленого методу.

4. Позитивні сторони роботи Кваліфікаційна робота містить ряд інноваційних рішень, зокрема запропонований метод дозволяє значно знизити часові та ресурсові затрати під час проєктування вебдодатків

5. Негативні сторони роботи Використання розробленого методу потребує поглиблених знань розробників

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно. Пояснювальна записка відповідає нормам для її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційної роботи заслуговує позитивної оцінки. Весь матеріал дипломної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої задачі.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Розглянувши позитивні та негативні сторони предетавленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку добре

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
к.т.н, доц. кафедри ПТЗ Форкун Ю.В.

« 4 » грудня 2020. (підпис)

ХПУ Зам. 54, тир. 6000, 2007