Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів проектування бібліотеки ORM на PHP
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-19-3
Бут В.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник доц. Голян В.В.
(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Програмної інженерії
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)
Тип програми Освітньо-наукова програма
(освітньо-професійна або освітньо-наукова)
Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бут Віталію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів проектування бібліотеки ORM на PHP» затверджена наказом університету від 26 березня 2021 № 385 Ст
2. Термін подання студентом роботи до екзаменаційної комісії «08» травня 2021 р.
3. Вихідні дані до роботи *методи проектування бібліотеки ORM на PHP, середовище об'єктно-орієнтованого проєктування*
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, методи проектування бібліотеки ORM на PHP, особливості проектування бібліотеки на PHP*
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, ілюстрацій *мета завдання, обґрунтування доцільності розроблення, постановка задачі, методи і алгоритми, структурно-логічна схема взаємодії даних, опис отриманих результатів, інтерфейс програмної системи, демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	Голян В.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.01.21 – 19.02.21	*Виконано*
2	Огляд існуючих методів та бібліотек	20.02.21 – 10.03.21	*Виконано*
3	Бібліотека ORM	11.03.21 – 25.03.21	*Виконано*
4	Підготовка пояснювальної записки	26.03.21 – 15.04.21	*Виконано*
5	Спецчастина	15.04.21 – 17.04.21	*Виконано*
6	Підготовка презентації та доповіді	18.04.21 – 19.04.21	*Виконано*
7	Попередній захист	20.04.21	*Виконано*
8	Нормоконтроль, рецензування	21.04.21 – 22.04.21	*Виконано*
9	Занесення диплома в електронний архів	23.04.21	*Виконано*
10	Допуск до захисту у зав. кафедри	07.05.21	*Виконано*

Дата видачі завдання 26 березня 2021р

Студент _____ Бут В.С.
(підпис)

Керівник роботи _____ доц. Голян В.В
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи бакалавра: 70 с., 20 рис., 17 джерел.

Об’єкт розробки – бібліотека ORM

Мета розробки – створити бібліотеку ORM, яка зможе стати конкурентом для таких ORM як: Eloquent, PHP ActiveRecord, RedBean, Propel, Doctrine ORM та буде краще їх в таких аспектах як: складання SQL запитів, виконання складених SQL запитів, простота використання, повнота тестового покриття, слідування стандартам PSR.

Метод рішення – мова PHP

В результаті розробки створено програмний додаток, а саме була реалізована бібліотека Sota ORM на мові програмування PHP

Explanatory note to the certification work of the bachelor: 70 pp., 20 figs., 17 sources.

The object of development is the ORM library

The purpose of development is to create an ORM library that can compete with such ORMs as: Eloquent, PHP ActiveRecord, RedBean, Propel, Doctrine ORM and will be better in such aspects as: compiling SQL queries, executing composite SQL queries, ease of use, completeness of the test coverage, compliance with PSR standards.

The solution method is PHP

As a result of development the software application was created, namely the Sota ORM library in PHP programming language was implemented

Я, Бут Віталій Сергійович, студент гр. ІПЗм-19-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя

кваліфікаційна робота на тему «Дослідження методів проектування бібліотеки ORM на PHP», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

ORM або Object-relational mapping (рус. Об'єктно-реляційне відображення) - це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами програмування. ORM використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому ORM сама піклується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних - нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі.

Завдання довготривалого зберігання об'єктів виникає, при створенні програмного забезпечення за допомогою об'єктно-орієнтованого підходу.

Проблема семантичного розриву між способами обробки об'єктів та їх зберіганням [1] зявляється, внаслідок використання реляційної БД для збереження об'єктно-орієнтованих об'єктів. Програмісти вимушені використовувати будь-які концепції при розробці бізнес логіки додатка (об'єктно-орієнтований підхід) і при взаємодії зі сховищем об'єктів (реляційна модель). Між концепціями здійснюється перехід кожен раз при зверненні до БД, а це в свою чергу призводить або до багаторазового дублювання коду, або до створення бібліотеки «прошарку» між базою даних і додатком. Розробка бібліотеки з точки зору технології програмування представляється кращім варіантом. Клас бібліотек, що надають методи для прямого і зворотнього перетворення об'єктів в реляційну модель, називається ORM.

Зберігання об'єктів в об'єктно-орієнтованих базах даних [2] і є альтернативою використання об'єктно-реляційного відображення. Такий підхід усуває

семантичний розрив між уявленнями об'єктів. Але, якісне об'єктно-орієнтоване відображення залишається на сьогоднішній день актуальним завданням через високу поширеність реляційних БД,.

Використанні методи дослідження, під час виконання роботи:

– метод аналіза;

– метод абстрогування;

– метод порівняння.

Практичне значення роботи полягає в тому, що отримані результати спрощують розробку, простоту використання, слідування стандартам PSR.

Результати даної роботи було опубліковано у збірнику матеріалів 41-ї Всеукраїнської практично пізнавальної конференції «НАУКОВА ДУМКА СУЧАСНОСТІ І МФЙБУТНЬОГО», яка відбулась 20-28 лютого 2021 року у м. Дніпро, Україна.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Поняття об'єктно-реляційного відображення

ORM (англ. Object-Relation Mapping, укр. об'єктно-реляційне відображення) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

При створенні програмного забезпечення за допомогою об'єктно-орієнтованого підходу, виникає завдання довготривалого зберігання об'єктів. Використання реляційної БД для зберігання об'єктно орієнтованих даних призводить до семантичного розриву між способами обробки об'єктів та їх зберігання [1]. Програмісти змушені використовувати різні концепції при розробці бізнес логіки додатка (об'єктно-орієнтований підхід) і при взаємодії зі сховищем об'єктів (реляційна модель). Перехід між концепціями відбувається кожного разу при зверненні до БД, що призводить або до багаторазового дублювання коду, або до створення бібліотеки «прошарку» між базою даних і додатком. Варіант з розробкою бібліотеки з точки зору технології програмування представляється кращим. Клас бібліотек, що надають методи для прямого і зворотнього перетворення об'єктів в реляційну модель, називається ORM.

Альтернативою використання об'єктно-реляційного відображення є зберігання об'єктів в об'єктно-орієнтованих базах даних [2]. Такий підхід усуває семантичний розрив між уявленнями об'єктів. Проте, через високу поширеність реляційних БД, якісне об'єктно-орієнтоване відображення залишається на сьогоднішній день актуальним завданням.

1.2 Цілі об'єктно-реляційного відображення

Цілі використання ORM бібліотек:

- зменшення числа помилок у процесі перетворення даних між реляційної та об'єктної моделями;
- розбиття логіки додатка та логіки зберігання даних;
- створення автономного коду, який не залежить від конкретної реалізації СУБД;
- зниження ризиків порушення цілісності даних;
- підвищення безпеки роботи з даними;
- зменшення розмірів програмного кода.

1.3Завдання об'єктно-реляційного відображення

Розробка універсального підходу до автоматичного прямого і зворотного відображення об'єктно-орієнтованих і реляційних даних пов'язані з труднощами (в англ. літературі - «object-relational impedance mismatch» [1], [3]):

- ступінь деталізації. Об'єктна модель може містити більше класів, ніж кількість відповідних таблиць в базі даних. Об'єктна модель є більш деталізованою ніж реляційна модель;
- спадкування. Спадкування є однією з особливостей об'єктно орієнтованих мов програмування. Сучасні реляційні СУБД не мають нічого подібного спадкоємства;
- ідентичність. Реляційні СУБД визначають ідентичність за допомогою первинного ключа. У ООП об'єкти не мають аналога первинного ключа, і ідентичність для об'єктів визначається самим програмістом;

- асоціації. Асоціації в об'єктно-орієнтованих мовах представлені у вигляді односпрямованих посилань. У реляційних СУБД використовуються поняття зовнішнього ключа. Якщо в об'єктно-орієнтованій мові потрібно визначити двосторонній зв'язок, то необхідно визначати її двічі;
- доступ до даних. В об'єктно-орієнтованих мовах для доступу до даних відбувається перехід від однієї асоціації до іншої по графу об'єктів. Такий спосіб не є ефективним при отриманні даних з реляційної БД. Як правило, кількість SQL запитів корисно мінімізувати. Завантаження декількох об'єктів здійснюється за допомогою JOIN операцій по всіх цільових об'єктів відразу.

Подолання цих труднощів в процесі досягнення цілей з попереднього параграфа призводить до необхідності вирішення наступних завдань:

- генерація схеми реляційної бази даних, ґрунтуючись на об'єктній моделі;
- підтримка різних реляційних баз даних;
- опис об'єктної моделі, типів полів, асоціацій;
- завантаження об'єктів по первиному ключу;
- виконання SQL запитів;
- створення механізмів опису відображень повернутих даних в об'єкти;
- автоматична генерація SQL запитів;
- захист бази даних від SQL-injection атак;
- раннє / відкладене завантаження даних.

1.4 Критерії порівняння

Головні критерії порівняння для об'єктно-реляційного відображення:

Функціональність:

- складання SQL запитів;

– виконання складених SQL запитів;
– механізм опису відображення;
– управління схемою БД.

Простота використання.

Модульність бібліотеки.

Повнота тестового покриття.

Наявність залежностей від інших бібліотек.

Слідування стандартам PSR.

Шаблон проектування.

1.5 Приклад бібліотек об'єктно-реляційного відображення

1.5.1 Eloquent

Функціональність:

– складання SQL запитів: присутній;
– виконання складених SQL запитів: відсутній;
– механізм опису відображення: відсутній;
– управління схемою БД: присутній ручне управління схемою бази даних.

Простота використання: надає досить просте API.

Модульність бібліотеки: бібліотека не є модульною, жорстко вбудована в фреймворк.

Повнота тестового покриття: хороше покриття тестами.

Наявність залежностей від інших бібліотек: залежить від інших бібліотек фреймворка Laravel.

Слідування стандартам PSR: слідує стандартам.

Шаблон проектування: ActiveRecord [4].

1.5.2 PHP ActiveRecord

Функціональність:

- складання SQL запитів: присутній;
- виконання складених SQL запитів: відсутній;
- механізм опису відображення: відсутній;
- управління схемою БД: не вміє генерувати схему бази даних по об'єктної моделі.

Простота використання: досить просто використовувати.

Модульність бібліотеки: один модуль.

Повнота тестового покриття: хороше покриття тестами.

Наявність залежностей від інших бібліотек: бібліотека не залежить від інших бібліотек, але сама реалізує безліч якісних сторонніх бібліотек.

Слідування стандартам PSR: не слідує жодному стандарту.

Шаблон проектування: ActiveRecord [4].

1.5.3 RedBean

Проста, але малофункціональна ORM. Дозволяє просто отримувати і зберігати дані з бази даних. Немає необхідності описувати схему бази даних, вона генерується бібліотекою автоматично. Однак, через відсутності об'єктної моделі цю бібліотеку не можна віднести до ORM. Дана бібліотека принципово являє собою Table Gateway [4].

Функціональність:

- складання SQL запитів: обмежена функціональність;
- виконання складених SQL запитів: відсутній;
- механізм опису відображення: відсутній;

– управління схемою БД: бібліотека створює схему бази даних на льоту.

Простота використання: дуже проста у використанні, низький поріг входження.

Модульність бібліотеки: складається з одного модуля.

Повнота тестового покриття: хороше покриття тестами.

Наявність залежностей від інших бібліотек: не залежить від інших бібліотек.

Слідування стандартам PSR: слідує частині стандартів.

Шаблон проектування: Table Gateway [4].

1.5.4 Propel

Функціональна і потужна ORM бібліотека, вміє генерувати код об'єктної моделі по базі даних. Підтримує всі базові операції (створення, отримання, оновлення, видалення) над об'єктами. Вміє керувати асоціаціями, транзакціями. Підтримує успадкування. Для оновлення схеми бази даних використовується механізм «міграцій». Однак має досить великий поріг входження і складна у використанні.

Функціональність:

– складання SQL запитів: присутній;

– виконання складених SQL запитів: відсутній;

– механізм опису відображення: відсутній;

– управління схемою БД: присутній у вигляді конфігураційних файлів XML.

Простота використання: складна у використанні, високий поріг входження.

Модульність бібліотеки: бібліотека складається з одного модуля.

Повнота тестового покриття: відмінне покриття тестами.

Наявність залежностей від інших бібліотек: залежить від ряду сторонніх бібліотек.

Слідування стандартам PSR: повністю слідує стандартам.

Шаблон проектування: Data Mapper [4].

1.5.5 Doctrine ORM

Функціональність:

- складання SQL запитів: обмежена функціональність;
- виконання складених SQL запитів: відсутній;
- механізм опису відображення: відсутній;
- управління схемою БД: бібліотека створює схему бази даних на льоту.

Простота використання: складна у використанні, високий поріг входження.

Модульність бібліотеки: складається з незалежних модулів.

Повнота тестового покриття: відмінне покриття тестами.

Наявність залежностей від інших бібліотек: залежить від великого числа сторонніх бібліотек.

Слідування стандартам PSR: слідує частині стандартів.

Шаблон проектування: Data Mapper [4].

1.6 Огляд Doctrine ORM

Doctrine ORM являє собою модульну бібліотеку засновану на інших бібліотеках Doctrine Project [5]:

- Doctrine Common

Відповідає за конфігурацію, кешування і різні допоміжні дії:

- Doctrine DBAL

Являє собою шар абстракції бази даних (англ. Database Abstraction Layer). Має безліч функцій серед яких управління і самоаналіз схеми бази даних. Doctrine ORM дозволяє описувати конфігурацію відображення у чотирьох різних форматах:

- анотації;
- XML;
- YAML;
- PHP.

По конфігурації вміє генерувати код об'єктної моделі та схему бази даних. Має безліч функціональних можливостей за описом різних асоціацій і роботі з ними. Дозволяє описувати запити на об'єктно-орієнтованому діалекті SQL, а так само за допомогою будівника запитів і за допомогою рідного SQL мови використовуваної СУБД. Має потужну систему кешування запитів і результатів запитів. Однак високий поріг входу часто відштовхує розробників, а складність реалізації викликає непередбачувані складнощі у використанні.

1.7 Методи досліджень та послідовність етапів проведення наукових досліджень

Метод аналізу – це метод наукового дослідження шляхом розкладання об’єктів на компоненти, а синтез - це поєднання частин, отриманих протягом аналізу. Методи аналізу та синтезу в наукових дослідженнях органічно взаємопов'язані і можуть приймати різні форми в залежності від особливостей предмета, мети дослідження, ступеня знання предмета та глибини проникнення в його суть. візьми це.

Метод абстрагування – метод наукового пізнання, що полягає в мисленому виділенні суттєвих, найістотніших рис, відношень, сторін предмета. За його допомогою формується ідеальний образ реальності. Процес абстрагування є складним, двоступеневим: спочатку відокремлюються суттєве від несуттєвого,

загальне від одиничного, важливе від неважливого, а потім установлюється незалежність або слабка залежність об’єкта пізнання від певних факторів для того, щоб відвернутися від них.

Метод порівняння – один з найпоширеніших методів пізнання, який встановлює подібність або відмінність різних об’єктів дослідження за певними ознаками. Порівняння – це процес зіставлення предметів або явищ дійсності з метою установлення подібності чи відмінності між ними, а також знаходження загального, притаманного, що може бути властивим двом або кіль-ком об’єктам дослідження.

Наукове дослідження – процес вивчення, експерименту, концептуалізації та перевірки теорії, пов'язаний з отриманням наукових знань.

Дослідження починаються з розробки програми. Програма обстеження - це документ, який регулює всі етапи, підготовку, організацію та етапи здійснення конкретного обстеження. Програма дослідження включає методологічні підходи та теоретичні демонстрації методологічних методів вивчення конкретних явищ чи процесів.

Початком наукових досліджень є детальний аналіз сучасного стану розглянутої проблеми. Це базується на пошуку інформації, який активно використовує джерела у глобальній комп’ютерній мережі Інтернет. На основі аналізу проблеми редагуються огляди та звіти, даються ключові класифікації напрямів та визначаються конкретні завдання дослідження.

Фактичною практикою наукових досліджень є вирішення спочатку поставлених завдань. Використовується математичне моделювання. Математичне моделювання передбачає кілька послідовних кроків. Це редагування математичної моделі дослідницького процесу на основі зібраних даних, використання готової моделі дослідницького процесу на основі зібраних даних або використання готової моделі з основами та коригуваннями. Допоміжний фактор. Для зручності визначення поставленої задачі математичний опис явища здійснюється у безрозмірних одиницях на основі теорії подібності. Потім розгляньте деякі умови та оберіть метод (аналітичний або наближений) для вирішення проблеми. Термін

виконання; оптимальна вартість матеріалу. Результати експерименту обробляються за допомогою комп’ютера.

Завершення наукової розробки - це аналіз отриманих результатів та їх проектування. Теоретичні та експериментальні результати порівнюються та дається аналіз їх можливих відмінностей. Звіт про проведені наукові дослідження буде підготовлений та підготовлений відповідно до державних стандартів.

Виокремимо п’ять основних етапів прикладних досліджень.

– формулювання теми;

– формулювання цілей та завдань дослідження (огляд літератури, порівняння та критика проблемної інформації, узагальнення та висвітлення проблем за темами);

– теоретичні дослідження (дослідження фізичної природи явищ, формулювання гіпотез, виведення математичних залежностей та їх теоретичний аналіз);

– експериментальні дослідження (розробка експериментальних цілей та завдань, планування, вимірювальні засоби, постановка експериментів, проведення експериментів, обробка результатів);

– аналіз та проектування результатів досліджень (загальний аналіз теоретичних та експериментальних досліджень, порівняння цих результатів, аналіз відмінностей, уточнення теорії за потребою, проведення додаткових експериментальних досліджень).

1.8 Постановка задачі

Головними задачами на роботу є:

- Вивчення методів проектування бібліотеки ORM;
- Дослідження методів підходу до проектування цієї бібліотеки;
- Розроблення об'єктно-реляційного відображення, яке було б достатньо функціонально і одночасно просте в реалізації.

Отримане рішення повинно відповідати таким вимогам:

- мати просту конфігурацію;
- отримувати інформацію про об'єктну модель з коду програми;
- по об'єктної моделі генерувати схему бази даних;
- мати простий спосіб отримання об'єктів;
- можливість запиту на SQL мовою СУБД.

2 АРХІТЕКТУРА БІБЛІОТЕКИ

2.1 Порівняльний аналіз можливих підходів

Існує кілька підходів (патернів) до створення бібліотеки об'єктно-реляційного відображення (англ. Data Source Architectural Patterns): Table Gateway, Row Gateway, Active Record, Data Mapper [4].

У цьому параграфі ми розглянемо і порівняємо перераховані патерни.

2.1.1 Table Gateway

Об'єкт виконує роль шлюзу до таблиці БД на рисунку 2.1. Один екземпляр об'єкта обробляє всі рядки в таблиці.

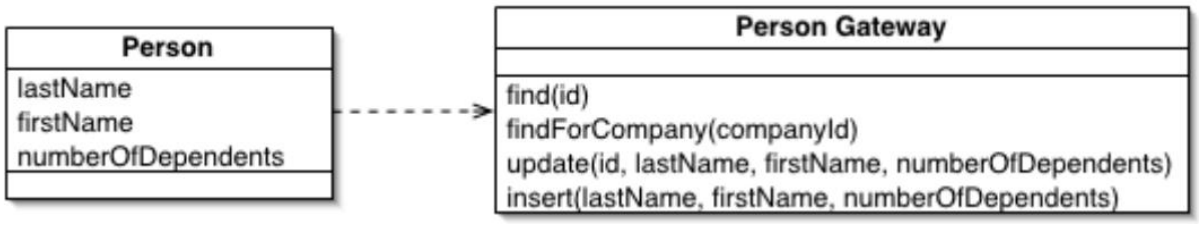

Рисунок 2.1 - Об'єкт виконує роль шлюзу до таблиці БД

Об'єкт який реалізує Table Gateway містить всі SQL запити для доступу до однієї таблиці з БД: вибірки, вставки, оновлення та видалення рядків. Код працює з об'єктом Table Gateway викликає відповідні методи для взаємодії з базою даних.

Недоліки:

- не відбувається поділу логіки програми та логіки зберігання даних;
- від розробників потрібна побудова коректних оптимізованих SQL-запитів.

2.1.2 Row Gateway

Об'єкт виконує роль шлюзу до одного рядка в таблиці БД на рисунку 2.2. Створюється один екземпляр об'єкта для кожного рядка БД.

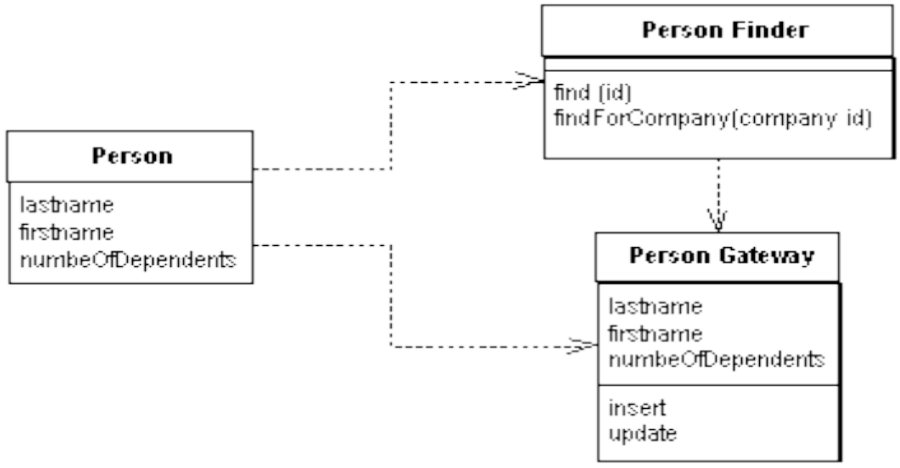

Рисунок 2.2 - Об'єкт виконує роль шлюзу до одного рядка в таблиці БД

Row Gateway надає об'єкти, що мають ту ж структуру, що і рядки в таблиці БД. Всі деталі взаємодії з таблицею БД приховані за цим інтерфейсом.

Недоліки:

- якщо об'єкти мають свою бізнес логіку, додавання коду для взаємодії з базою даних, збільшує складність розробки;
- уповільнення роботи програми (у тому числі, в режимі тестування) через постійне звернення до БД при вибірці об'єктів.

2.1.3 Active Record

Об'єкт представляє обгортку для рядка таблиці БД і додає бізнес-логіку до цих даних (рис.2.3).

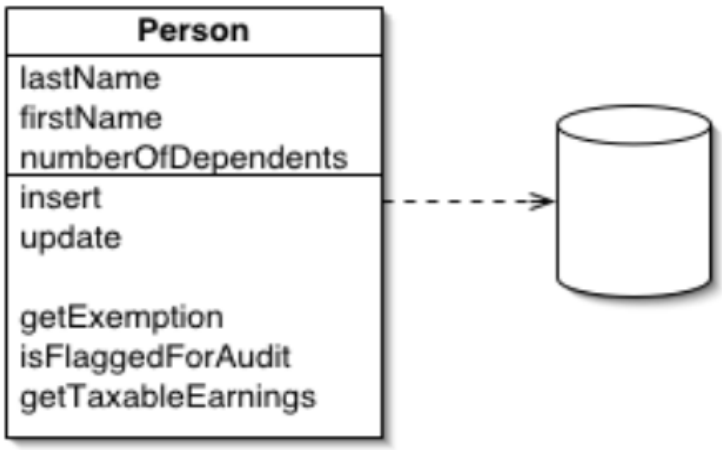

Рисунок 2.3 - Об'єкт представляє обгортку для рядка таблиці БД і додає бізнес-логіку до цих даних.

Об'єкт який реалізує патерн Active Record містить в собі як дані, так і поведінку. Значна частина даних містяться в об'єкті повинна бути збережена в БД. Active Record використовує найбільш очевидний підхід, включивши логіку доступу до даних в бізнес-логіку об'єкта.

2.1.4 Data Mapper

Даний патерн на рисунку 2.4 представляє об'єкт, який переміщує дані між об'єктною моделлю і базою даних, зберігаючи їх незалежними один від одного.

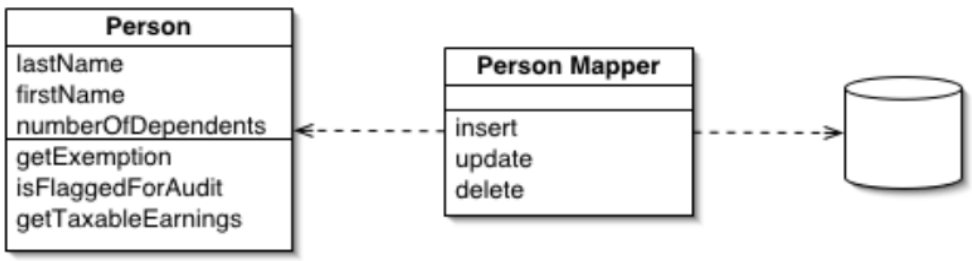

Рисунок 2.4 - Патерн представляє об'єкт, який переміщує дані між об'єктною моделлю і базою даних, зберігаючи їх незалежними один від одного.

Об'єктна і реляційна моделі даних мають різні механізми для роботи з ними. Багато частин об'єктної моделі, такі як, колекції та успадкування, не присутні в реляційних даних. При організації об'єктної моделі з великою кількістю бізнес-логіки корисно розділяти дані і поведінку. Це призводить до протиріччя: об'єктна модель і реляційна схема не збігаються. Data Mapper являє собою шар програмного забезпечення, який відокремлює об'єктну модель від бази даних. Data Mapper повинен передавати дані між об'єктною моделлю і БД, а так само ізолювати їх один від одного. При використанні паттерна Data Mapper, об'єктна модель нічого не знає про структуру БД, не містить ніяких методів отримання даних, не містить в собі SQL запити.

2.2 Переваги обраного підходу

Для розробки бібліотеки об'єктно-реляційного додатка був обраний патерн проектування Active Record. Вибір був продиктований такими міркуваннями:

- патерн Active Record дозволяє уникнути перерахованих для Table Gateway і Row Gateway недоліків;
- широко поширений і добре відомий в середовищі веб-програмістів;
- у порівнянні з Data Mapper істотно виграє за кількістю коду, який необхідно написати програмісту для аналогічних операцій з даними.

Демонстрацію приклада роботи з Data Mapper за допомогою Doctrine ORM можна побачити на рисунку 2.5.

Такий самий код може бути переписаний при використанні Active Record з використанням меншої кількості рядків коду, таким чином залишаючи менше місця для потенційних помилок.

```php
// Пример использования Data Mapper с Doctrine ORM.

// Для работы с Data Mapper нужно всегда получать/создавать EntityManager класс.
$entityManager = new EntityManager();

// Создаём объект.
$message = new Message();
$message->user = $user;
$message->datetime = new \DateTime();
$message->text = $text;

// Размещаем объект в EntityManager.
$entityManager->persist($message);

// Производим выполнение всех необходимых запросов для синхронизации с БД.
$entityManager->flush();
```

Рисунок 2.5 - Приклад використання Data Mapper з Doctrine ORM

Приклад того ж коду при використанні Active Record із застосуванням розробляється в дипломі бібліотеки Sota ORM зображено на рисунку 2.6.

```php
// Пример использования Active Record с Granula ORM.

// Создаём объект.
$message = new Message();
$message->user = $user;
$message->datetime = new \DateTime();
$message->text = $text;

// Сохраняем объект в БД.
$message->save();
```

Рисунок 2.6 - Приклад використання Active Record з Sota ORM

2.3 Опис концепції

В основу архітектури розробляється бібліотеки Sota ORM покладено патерн Active Record. Код бібліотеки відповідає стандартам оформлення коду PSR-1 і стандарту автозавантаження класів PSR-4.

Мінімально необхідна версія PHP для роботи бібліотеки - 5.5. Бібліотека використовує безліч нововведень, що з'явилися в останніх версіях PHP, такі як пізніше статичну зв'язування, трейти / домішки, генератори та багато іншого.

Існуючі реалізації Active Record вимагають, щоб всі об'єкти бізнес-логіки успадковувалися від деякого класу, що реалізує даний патерн. Sota ORM використовує домішки для додавання необхідної функціональності. Приклад використання домішки представлений в рисунку 2.7.

```php
namespace Model;

use Granula\ActiveRecord;

class User
{
    use ActiveRecord;
}
```

Рисунок 2.7 - Приклад використання домішки представлений

2.4 Структура розроблених класів

Схема структури розроблених класів відображено на рисунку 2.8.

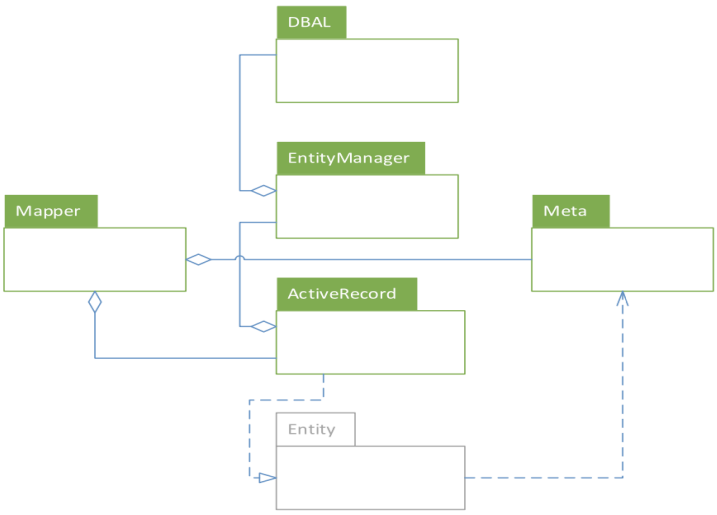

Рис. 2.8 - Схема структури розроблених класів

Entity - набір класів, конкретна реалізація бізнес-логіки додатка, реалізується програмістами при розробці ПЗ.

ActiveRecord - компонент, що містить в собі методи для роботи з об'єктами, найбільш часті і загальні методи побудови запитів.

EntityManager - управляє всіма об'єктами бізнес-логіки, конфигурирует підключення до бази даних, кешує об'єкти в пам'яті.

DBAL - шар абстракції бази даних (англ. Database Abstraction Layer), управляє схемою бази даних і конвертацією типів між базою даних і додатком.

Meta - набір класів представляє інформацію про бізнес-логікою додатка.

Mapper - компонент аналізує SQL запити і створює функцію відображення.

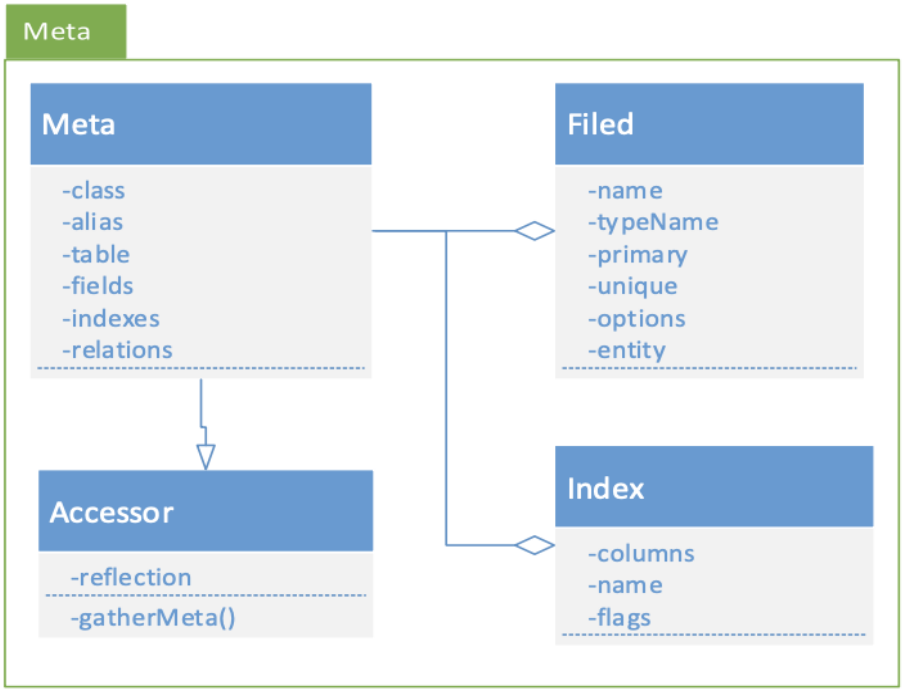

Рисунок 2.9 - Схема структура класів компонента Meta

На Рисунку 2.9 показана структура класів компонента Meta. Клас Meta містить в собі посилання на класи Filed і Index відповідають за поля і індекси об'єктів відповідно. Клас Accessor успадковується від класу Meta і реалізує в собі функціональність, що дозволяє отримати мета- інформацію про об'єктах використовуючи DocBlock.

2.5 Управління доступом до внутрішніх даних об'єктів

У PHP відсутні механізми для обмежувати доступ до певних членів класу (англ. Setter | getter). У Sota ORM реалізований трейт Sota \ Setter, який додає дану функціональність, а також додає підтримку обходу графа об'єктної моделі, управління ленивим завантаженням, перевірку типів підставляється значень. При цьому залишається підтримка роботи користувальницьких сеттерів і геттеров.

```php
namespace Model;

use Granula\ActiveRecord;
use Granula\Setter;

// Описываем класс с приватными полями и включаем трейт Setter.
class User
{
    use ActiveRecord;
    use Setter;

    private $id;
    private $name;
    private $email;
}
```

Рисунок 2.10 - Декларація класа з приватними членами

Рисунок 2.10 демонструє декларацію класу з приватними членами, доступ до яких може бути здійснений за допомогою трейта Sota \ Setter.

```php
// Создаём экземпляр класса.
$user = new User();

// Устанавливаем заначения полей.
$user->name = 'User Name';
$user->email = 'mail@domain.com';
```

Рисунок 2.11 - Трейт Sota \ Setter

Тепер, якщо потрібно поміняти логіку роботи сетера name, слід визначити метод setName рисунок 2.12.

```php
// Описываем класс с приватными полями и включаем трейт Setter.
// Переопределяем сеттер для поля name.
class User
{
    use ActiveRecord;
    use Setter;

    private $id;
    private $name;
    private $email;

    public function setName($value)
    {
        $this->name = strtoupper($value);
    }
}
```

Рисунок 2.12 - Метод setName

Код, що привласнює значення для об'єкта Рисунок 2.11, залишається без змін, але логіка роботи сетера змінена.

Так само, даний трейт управляє завантаженням об'єктів при обході дерева об'єктів.

```php
// Получение названия города проживания пользователя.
$cityName = $user->address->city->name;
```

Рисунок 2.13 - Звичайний спосіб обходу дерева об'єктів

У Рисунку 2.13 показаний звичайний спосіб обходу дерева об'єктів. У цьому випадку перед виконанням оператора з БД був завантажений лише об'єкт «користувач». Для отримання необхідного значення бібліотека робить ще два запити до бази даних для отримання пов'язаних об'єктів «адреса» і «місто», для чого автоматично генеруються і виконуються наступні SQL-запити (рис.2.14):

```
SELECT u.id AS u_id, u.name AS u_name, address.id AS address_id, address.city AS address_city FROM users u LEFT JOIN address address ON u.address = address.id WHERE u.id = ? LIMIT 1

SELECT a.id AS a_id, city.id AS city_id, city.name AS city_name FROM address a LEFT JOIN city city ON a.city = city.id WHERE a.id = ? LIMIT 1
```

Рисунок 2.14 - SQL-запити

2.6 Управління схемою бази даних

Для управління схемою бази даних була застосована бібліотека Doctrine DBAL. Опис схеми можна зробити двома різними способами.

Перший спосіб - потребує, щоб об'єкти бізнес-логіки реалізовували статичний метод describe (Sota \ Meta). В даний метод передається екземпляр класу Sota \ Meta, який і є описом мета даних об'єктної моделі і схеми бази даних. Приклад опису схеми БД першим способом представлений в рисунку 2.14.

```php
namespace Model;

use Granula\Meta;
use Granula\ActiveRecord;
use Granula\Setter;

class User
{
    use ActiveRecord;
    use Setter;

    private $id;
    private $name;
    private $password;
    private $email;
    private $avatar;
    private $profile;
    private $friend;
    private $date;

    public static function describe(Meta $meta)
    {
        $meta->table('users');
        $meta->field('id', 'integer')->primary()->options(['autoincrement' => true]);
        $meta->field('name', 'string');
        $meta->field('password', 'string');
        $meta->field('email', 'string')->unique()->options(['notnull' => true]);
        $meta->field('avatar', 'string')->options(['notnull' => false, 'default' => '']);
        $meta->field('profile', 'entity')->entity(Profile::class);
        $meta->field('friend', 'entity')->entity(User::class);
        $meta->field('date', 'datetime');
        $meta->index(['name', 'email'], 'name_email_index');
    }
}
```

Рисунок 2.15 - Опис схеми БД

Екземпляр класу Sota \ Meta дозволяє описати схему бази даних: ім'я таблиці, поля, типи полів, опції, відносини і індекси.

Другий спосіб - застосовує рефлексію класів PHP [6]. Бібліотека Sota ORM аналізує назви полів, коментарі до них, і здійснює побудова метаданих схеми

Sota\ Meta, як якщо б вони були описані в методі describe. Приклад опису схеми БД другим способом продемонстрований в Рисунку 2.15.

Sota ORM працює в двох режимах - develop і production. В режимі develop бібліотека отримує інформацію про структуру БД, структурі схеми даних об'єктної моделі, виконує порівняння отриманих схем, після чого генерує SQL-код, який виконує зміна схеми БД для відповідності об'єктної моделі.

2.7 Завантаження об'єкта по первинному ключу

Для завантаження об'єктів з БД по первинному ключу реалізований метод find (int).

Перед генерацією та виконанням SQL коду перевіряється, чи не був цей об'єкт завантажений раніше. Таким чином, вдається уникнути повторних запитів і появи копій одного і того ж об'єкта. Управління та зберігання об'єктів здійснюється класом Sota \ EntityManager.

Якщо об'єкт не міститься в Sota \ EntityManager, то виконується генерація SQL коду для отримання всіх полів цього об'єкта з бази даних. Якщо цей об'єкт має зв'язку з іншими об'єктами, SQL код буде доповнений JOIN конструкціями для завантаження цих об'єктів. Таким чином, вдається уникнути зайвих запитів до БД при обході дерева об'єктів.

2.8 Перетворення типів даних

При роботі з БД виникає проблема перетворення типів даних. Для її вирішення використовуються перетворювачі типів даних, що викликаються при кожному прямому і зворотному відображенні. Підтримувані типи даних:

- integer;
- float;
- string;
- text;
- binary;
- blob;
- boolean;
- datetime;
- array;
- object.

Користувач може описати свій тип даних. Приклад опису користувацького типу даних «гроші» представлено в рисунку 2.16.

```php
namespace Model\Types;

use Doctrine\DBAL\Types\Type;
use Doctrine\DBAL\Platforms\AbstractPlatform;

// Пользовательский тип данных представляющий "деньги".
class MoneyType extends Type
{
    const name = 'money'; // Название типа данных.

    public function getSqlDeclaration(array $fieldDeclaration, AbstractPlatform $platform)
    {
        return 'MyMoney';
    }

    public function convertToPHPValue($value, AbstractPlatform $platform)
    {
        return new Money($value);
    }

    public function convertToDatabaseValue($value, AbstractPlatform $platform)
    {
        return $value->toDecimal();
    }

    public function getName()
    {
        return self::name;
    }
}

Type::addType(MoneyType::name, MoneyType::class);
```

Рисунок 2.16 - Опис користувацького типу даних «гроші»

3 ОПИС СЦЕНАРІЇВ ВИКОРИСТАННЯ

3.1 Конфігурація і підключення в базі даних

На рисунку 3.1 наведено приклад конфігурації, в якому вказується:

– драйвер БД;

– параметри підключення до БД;

– режим роботи бібліотеки.

```
$params = [
    'dev' => true,
    'driver' => 'pdo_mysql',
    'host' => 'localhost',
    'user' => 'root',
    'password' => '',
    'dbname' => 'granula',
    'charset' => 'utf8'
];

$em = new EntityManager($params, [
    User::class,
    Profile::class,
]);
```

Рисунок 3.1 - Приклад конфігурації

Якщо зазначений режим роботи «develop», то буде проведена синхронізація схем БД і об'єктної моделі. Якщо схема БД відсутня, буде згенеровано SQL код для створення цієї схеми.

В клас EntityManager необхідно передати список об'єктів об'єктної моделі.

3.2 Створення і збереження об'єктів

Рисунок 3.2 демонструє створення двох об'єктів і збереження їх в базі даних. Спочатку створюється, заповнюється і зберігається об'єкт «Profile». Потім створюється об'єкт «User», з посиланням на об'єкт «Profile», і за допомогою методу lazy створюється і присвоюється посилання на неіснуючий об'єкт іншого користувача.

```php
$profile = new Profile();
$profile->age = 21;
$profile->tags = ['one', 'two'];
$profile->birth = new DateTime();
$profile->city = 'Saint Petersburg';
$profile->create();

$user = new User();
$user->name = 'Anton';
$user->avatar = null;
$user->profile = $profile;
$user->friend = User::lazy($friendId = 12);
$user->create();
```

Рисунок 3.2 - Створення двох об'єктів і збереження їх в базі даних

Після виконання цього коду будуть згенеровані і виконані наступні SQL запити.

```sql
INSERT INTO profile (id, city, birth, age, tags) VALUES (null, "Saint Petersburg", "2014-04-14 20:03:56", 21, "one,two")
INSERT INTO users (id, name, avatar, profile, friend) VALUES (null, "Anton", null, 10, 12)
```

Рисунок 3.3 - Завантаження об'єкта «користувач» по первинному ключу

3.3 Отримання користувача по первинному ключу

Рисунок 3.4 демонструє завантаження об'єкта «користувач» по первинному ключу. Об'єкт «профіль» буде завантажений разом з об'єктом «користувач», без додаткового запиту до БД.

```
$user = User::find(1);
$user->profile->tags = ['one', 'two'];
$user->profile->save();
```

Рисунок 3.4 - Отримання користувача по первинному ключу.

Рисунок 3.5 демонструє згенерований SQL запит, для завантаження о’єкта «користувач».

```
SELECT u.id AS u_id, u.name AS u_name, u.password AS u_password, u.email AS u_email, u.avatar
AS u_avatar, u.profile AS u_profile, u.friend AS u_friend, profile.id AS profile_id,
profile.city AS profile_city, profile.birth AS profile_birth, profile.age AS profile_age,
profile.tags AS profile_tags, friend.id AS friend_id, friend.name AS friend_name,
friend.password AS friend_password, friend.email AS friend_email, friend.avatar AS
friend_avatar, friend.profile AS friend_profile, friend.friend AS friend_friend FROM users u
LEFT JOIN profile profile ON u.profile = profile.id LEFT JOIN users friend ON u.friend =
friend.id WHERE u.id = 1 LIMIT 1;
UPDATE profile SET tags = "one,two" WHERE id = 10;
```

Рисунок 3.5 - Виконання призначеного для користувача SQL запиту до БД

3.4 Виконання призначеного для користувача SQL запиту

Рисунок 3.6 демонструє виконання призначеного для користувача SQL запиту до БД.

```php
$result = User::query('SELECT * FROM users u WHERE u.id IN (?)', [[1, 2]],
[Connection::PARAM_INT_ARRAY]);

foreach ($result as $user) {
    // Работа с экземпляром $user.
}
```

Рисунок 3.6 - Виконання SQL запиту

Метод query (string, array, array, mixed) приймає SQL запит, список параметрів, типи параметрів і функцію, що реалізує відображення рядка в об'єкт. В даному прикладі функція відображення не передається. Тому вона буде створена автоматично по мета-інформації про об'єкт.

Метод query повертає ітератор (створений за допомогою генератора), який може бути обійдений в циклі.

```sql
SELECT * FROM users u WHERE u.id IN (1,2)
```

Рисунок 3.7 - Отриманий SQL запит

3.5 Використання власної функції відображення рядка в об'єкт

В даному прикладі(рис. 3.8) в функцію query передається для користувача функція відображення. У неї передаються результати запиту до БД. Функція може повернути будь-який результат, він буде переданий в ітератор, що повертається методом query.

```php
$users = User::query('SELECT * FROM users u WHERE u.id > ?',
 [1],
 [\PDO::PARAM_INT],
 function ($result) {
    $user = new User();
    $user->id = $result['id'];
    $user->name = $result['name'];
    $user->email = $result['email'];
    $user->profile = Profile::lazy($result['profile']);
    return $user;
});

foreach($users as $user) {
    // Работа с экземпляром $user.
}
```

Рисунок 3.8 - До функції query передається функція відображення

ВИСНОВКИ

Під час аналізу предметної області були застосовані такі методи, як:

- метод аналіза
- метод абстрогіування
- метод порівняння

В результаті роботи була реалізована бібліотека Sota ORM на PHP, в якої є декілька переваг, у порівнянні з найближчими аналогами:

- легкість використання;
- швидкість вивчення;
- широка функціональність;
- відповідність до стандартів PSR.

У бібліотеці реалізовані наступні можливості:

- автоматичне створення схеми БД;
- два типи конфігурації об'єктної моделі: DocBlock і PHP;
- API для роботи з об'єктною моделлю;
- API для роботи з будівником запитів;
- трансформування типів даних;
- асоціації «один до одного» і «один до багатьох»;

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. R. Johnson: J2EE Design and Development, - Wrox, 2002. -768 с.
2. E. Bernard, «The Object-Relational Impedance Mismatch» URL: http://hibernate.org/orm/what-is-an-orm/#the-object-relational- impedance-mismatch (дата звернення: 24.02.2021).
3. M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002. – 560 с.
4. K. Dunglas, Persistence in PHP with Doctrine ORM, Packt Publishing, 2013. – 114 с.
5. «PHP: Reflection» URL: http://www.php.net/manual/ru/book.reflection.php (дата звернення: 13.03.2021).
6. Rasmus Lerdorf, Kevin Tatroe : Programming PHP, 2006. – 540 с.
7. Larry Ullman, PHP Advanced and Object Oriented Programming, 2012. – 504 с.
8. Джош Локхарт. Современный PHP. Новые возможности и передовой опыт, 2016 . – 304 с.
9. М. Зандстра. PHP: объекты, шаблоны и методики программирования, 5-е издание, 2019 – 736 с.
10. Уразливості і погрози вебдодатків в 2020 році / URL: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/web-vulnerabilities-2020- 63 rus.pdf (дата звернення: 14.03.2021).
11. Kachko O., N. Bilous, Semerkov V. Research on methods for secure web applications development Information Technologies in Innovation Business (ITIB), 7-9 October, 2015, Kharkiv, Ukraine Proceedings of ITIB, ISBN 978-966-659-214-2, P. 26- 27, (дата звернення: 11.04.2021).
12. O Kalynychenko, S Chalyi, S Shabanov-Kushnarenko, V Golyan. Discriminative Approach to Discovery Implicit Knowledge (дата звернення: 17.04.2021).

13. Olga Kalynychenko, Sergii Chalyi, Yevgeniy Bodyanskiy, Vira Golian, Nataliia Golian. Implementation of search mechanism for implicit dependences in process mining (дата звернення: 15.04.2021).
14. Gorbenko, I.D., Kachko, O.G., Yesina, M.V. Analysis of asymmetric NTRU prime IIT Ukraine encryption algorithm with regards to known attacks. Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika), 77(9), 2018, C. 799-816 (дата звернення: 08.04.2021).
15. Kachko O., N. Bilous, Semerkov V. Research on methods for secure web applications development Information Technologies in Innovation Business (ITIB), 7-9 October, 2015, Kharkiv, Ukraine Proceedings of ITIB, ISBN 978-966-659-214-2, P. 26- 27, (дата звернення: 11.04.2021).
16. Olga Kalynychenko, Sergii Chalyi, Yevgeniy Bodyanskiy, Vira Golian, Nataliia Golian. Implementation of search mechanism for implicit dependences in process mining (дата звернення: 15.04.2021).
17. O Kalynychenko, S Chalyi, S Shabanov-Kushnarenko, V Golyan. Discriminative Approach to Discovery Implicit Knowledge (дата звернення: 17.04.2021).