Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Системотехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка веб-застосунку для генерації ORM-моделей на основі схеми реляційної БД
(тема роботи)

Виконав: здобувач групи ІТПм-21-1
Долгополов К.В.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Інформаційні технології проектування
(повна назва освітньої програми)
Керівник доц. каф. СТ Імангулова З.А.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри системотехніки _____ Гребеннік І.В.
(підпис) (прізвище, ініціали)

2022 р.

Я як студент(ка) ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

17.12.2022 Долгополов

(дата, підпис, прізвище студента/-ки)

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Керівник кваліфікаційної роботи *доц. Імангулова З.А.*

Signer ID: PN2IPLNC29...

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Керівник кваліфікаційної роботи *доц. Імангулова З.А.*

Signer ID: PN2IPLNC29...

Попередній захист проведено 17 грудня 2022 р.

Керівник кваліфікаційної роботи *доц. Імангулова З.А.*

Signer ID: PN2IPLNC29...

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних Наук
Кафедра Системотехніки
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Інформаційні технології проектування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри

(підпис)
« 19 » грудня 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Долгополову Кирилу Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Розробка веб-застосунку для генерації ORM-моделей на основі схеми реляційної БД
затверджена наказом по університету від « 21 » листопада 2022р. № 1504 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20.12. 2022 р.
3. Вихідні дані до роботи Аналоги інформаційних систем для проектування реляційних баз даних. Перелік використовуваних програмних засобів: AllFusion Data Modeler, StarUML, Hackolade, Visual Studio Code, Lucidchart, Figma.
4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ. 4.2 Аналіз предметної області, постановка задачі та обгрунтування варіантів рішення. 4.2.1 Опис предметної області. 4.2.2 Детальний опис методів та технологій застосованих для розробки системи. 4.2.2.1 Веб-платформа. 4.2.2.2 Схема «сутність-зв'язок» реляційної бази даних. 4.2.2.3 Технологія ORM. 4.2.3 Аналіз існуючих рішень для роботи зі схемами РБД та ORM. 4.2.4 Визначення області застосування системи 4.3 Огляд методів та технологій, які застосовуються в предметній області. 4.3.1 Огляд методів візуального проектування. 4.3.1.1 Метод розміщення «Drag and Drop». 4.3.1.2 Метод розміщення «Point and Click». 4.3.1.3 Координатне розміщення. 4.3.1.4 Автоматичне розміщення. 4.3.2 Огляд форматів представлення схеми даних у вигляді конфігураційного файлу. 4.3.2.1 Формат XML. 4.3.2.2 Формат JSON. 4.3.2.3 Формат YAML. 4.3.3 Огляд способів збереження конфігураційних даних для генерації коду ORM-моделей. 4.3.3.1 Реляційна база даних. 4.3.3.2 Документоорієнтована база даних. 4.3.1.3 Конфігураційний локальний файл. 4.4 Постановка задачі дослідження. 4.4.1 Мета та

призначення системи. 4.4.2 Функціональні вимоги до системи. 4.4.3 Нефункціональні вимоги до системи. 4.4.4 Вихідна інформація. 4.5 Розробка інформаційної технології вирішення задачі. 4.5.1 Опис методу генерації конфігураційного файлу. 4.5.2 Опис методу генерації файлів моделей.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 5.1 Нотація Чена (1 аркуш формату А4). 5.2 Нотація Мартіна («вороняча лапка») (1 аркуш формату А4). 5.3 Нотація діаграми класів UML (1 аркуш формату А4). 5.4 Нотація IDEF1X (1 аркуш формату А4). 5.5 Схема конфігураційної бази даних сервісу (1 аркуш формату А4). 5.6 Алгоритм вибору типу даних ORM-бібліотеки (1 аркуш формату А4).

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Розділи спеціальної частини	*доц. Імангулова З.А.*	Signer ID: PN2IPLNC29...	17.12.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Отримання завдання на виконання роботи	10.10.2022	
2	Огляд літератури та аналіз предметної області	11-19.10.2022	
3	Аналіз систем-аналогів	20-24.10.2022	
4	Визначення області застосування системи	25-29.10.2022	
5	Огляд методів та технологій	30.10-07.11.2022	
6	Постановка задачі дослідження	08-12.11.2022	
7	Опис методу генерації конфігураційного файлу	13.11-19.11.2022	
8	Опис методу генерації коду для ORM-моделей	20-29.11.2022	
10	Оформлення пояснювальної записки	30.11-12.11.2022	
11	Оформлення додатків	13-16.12.2022	
12	Представлення на рецензування	17.12.2022	

Дата видачі завдання 10 жовтня 2022 р.

Студент _____ Долгополов К.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Імангулова З.А.
(підпис) (посада, прізвище, ініціали)
Signer ID: PN2IPLNC29...

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 80 с., 4 табл., 6 рис., 2 додатки, 35 джерел.

АВТОМАТИЗОВАНІ ІНФОРМАЦЙНІ СИСТЕМИ, ГЕНЕРАЦІЯ КОДУ, КОНФІГУРАЦІЙНИЙ ФАЙЛ, РЕЛЯЦІЙНІ БАЗИ ДАНИХ, СЕРВІСНА АРХІТЕКТУРА, СХЕМИ ДАНИХ, ORM-БІБЛІОТЕКИ

Об’єктом досліджень є процес генерації ORM-моделей на основі логічної та фізичної схем даних.

Предметом досліджень є особливості веб-платформи як місця для реалізації автоматизованої інформаційної системи, програмні засоби та підходи необхідні для створення інструменту візуального проектування схеми даних, формати представлення даних для конвертації схеми бази даних в текстовий формат, інформаційні технології автоматичної генерації програмного коду.

Метою роботи є дослідження предметної області автоматизованих інформаційних систем для генерації програмного коду, аналіз отриманої інформації, визначення ефективних методів для візуального проектування схеми даних, представлення конфігураційних даних, а також проектування інформаційних технологій для конвертації візуальної схеми даних в текстовий формат та автоматизація процесу генерації ORM-моделей на основі конфігураційного файлу схеми даних.

Методи дослідження – системний підхід, методи структурного аналізу та моделювання, аналіз існуючих методів візуального проектування, форматів текстового представлення конфігураційних даних та способів збереження конфігураційних даних.

Новизна роботи полягає у розробці інформаційної системи, яка дасть можливість користувачу виконувати генерацію програмного коду ORM-моделей для обраної ORM-бібліотеки на основі створеної схеми бази даних.

Галузь застосування – використання IT-фахівцями для автоматизації процесу розробки віртуальної бази даних.

ABSTRACT

Master's Thesis: 80 p., 4 tables, 6 pic., 2 appendices, 35 sources.

AUTOMATED INFORMATION SYSTEMS, CODE GENERATION, CONFIGURATION FILE, RELATIONAL DATABASES, SERVICE ARCHITECTURE, DATA SCHEMA, ORM LIBRARIES

The object of research is the process of generating ORM models based on logical and physical data schemas.

The subject of research is the peculiarities of the web platform as a place for the implementation of an automated information system, software tools and approaches necessary to create a tool for visual design of a data scheme, data presentation formats for converting a database scheme into a text format, information technologies for automatic generation of program code.

The purpose of the work is to research the subject area of automated information systems for generating program code, analyze the received information, determine the effective methods for visual data schema design, configuration data presentation, as well as information technology design for converting a visual data schema into a text format and automating the ORM generation process models based on the data schema configuration file.

Research methods – systematic approach, methods of structural analysis and modeling, analysis of existing methods of visual design, formats of text representation of configuration data and methods of saving configuration data.

The novelty of the work consists in the development of an information system that will enable the user to generate the software code of ORM models for the selected ORM library based on the created database scheme.

Scope of usage – usage by IT specialists to automate the process of developing a virtual database.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АІС – автоматизована інформаційна система;
БД – база даних;
ЕОМ – електронна обчислювальна машина;
ІС – інформаційна система;
ОС – операційна система;
ПЗ – програмне забезпечення;
РБД – реляційні бази даних;
СА – система автоматизації;
СКБД – система керування базами даних;
API – Application Programming Interface, прикладний програмний інтерфейс;
ER-модель – Entity-Relationship model, модель «сутність-зв'язок»;
JSON – JavaScript Object Notation, текстовий формат обміну даними між комп'ютерами;
ORM – Object-Relational Mapping, об'єктно-реляційне відображення;
REST – Representational State Transfer, передача репрезентативного стану;
RFC – Request for Comments, документ із серії пронумерованих інформаційних документів Інтернету, що містить технічні специфікації та Стандарти;
SOAP – Simple Object Access Protocol, протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі
SQL – Structured query language, мова структурованих запитів;
UI – User interface, інтерфейс користувача;
UX – User Experience, досвід користування;
W3C – World Wide Web Consortium, Консорціум Всесвітнього павутиння;
XML – Extensible Markup Language, розширювана мова розмітки;
YAML – YAML Ain't Markup Language, «YAML – не мова розмітки».

ВСТУП

В сучасному світі важко уявити, що вся інформація раніше велась в паперовому вигляді, неймовірна кількість довідок, звітів, квитанцій та інших паперових носіїв, які об’єднувались в архіви та могли займати великі площі території для розміщення та зберігання. Окрім цього, постає ще питання у підтримці тривалого та стабільного зберігання носіїв даних такого формату, адже цілий набір факторів такі як волога, час, комахи та багато інших нещадно знищують її.

Але з плином часу розвиток технологій набрав значних оборотів, настала нова епоха технічної революції – цифрової революції, що привнесла значні зміни в стані інформаційно-комп’ютерних технологій. Звичайно це не могло не зачепити область опрацювання та зберігання інформації, які замість ручного опрацювання замінились просунутими електронними системами.

Майже кожна система працює з безпосереднім використанням баз даних, яка слугує тим самим «сховищем зі звітами». Спектр застосування СКБД величезний - бази даних використовуються в інтернеті, у виробництві, у промисловості, у маркетингу, у мобільних пристроях, у фінансовій та банківській сферах, на телебаченні, у телекомунікаціях та рекламі [1].

Але окрім очевидних плюсів використання БД звичайно є і мінуси, зокрема процес їх проектування та інтеграції її програмний код. Цю проблему намагаються обходити через використання ORM-бібліотек, які надають зручний програмний інтерфейс для виконання базових запитів. Та сам процес налаштування ORM-бібліотеки в проекті займає багато часу, зокрема написання ORM-моделей.

У зв’язку з цим пропонується розробити веб-застосунок, що дасть змогу IT-фахівцю проводити графічне моделювання бази даних, вносити зміни до її структури та зберігати для подальшої роботи з нею. Окрім безпосереднього моделювання, веб-портал також матиме вбудований інструмент автоматизації для генерації програмного коду на основі розробленої моделі, який зробить легшим роботу з таблицями бази даних в самому проекті.

Для досягнення мети кваліфікаційної роботи було винесено такий перелік задач для вирішення:

- опис предметної області;

- детальний опис застосованих методів та технологій;
- аналіз існуючих подібних рішень;
- визначення області застосування системи;
- огляд методів візуального проектування схеми даних;
- огляд форматів представлення схеми даних;
- огляд способів збереження конфігураційних даних для генерації коду ORM-моделей;
- постановка задачі дослідження;
- опис методу генерації конфігураційного файлу схеми даних;
- опис методу генерації файлів моделей.

Виконання усіх вищеперерахованих задач дасть змогу розробити систему з дотриманням усіх особливостей предметної області та самого проекту.

Результати даної роботи доповідалися і обговорювалися на II Міжнародній студентській науковій конференції «Концепт науки XXI: стратегії, методи та наукові інструменти» (Харків, 2022) [2].

По завершенню етапів дослідження, аналізу та проектування на виході отримаємо автоматизовану інформаційну систему, яка слугуватиме для IT-фахівців єдиним інструментом для проектування рівня даних програмного забезпечення. Зручний графічний інтерфейс проектування дасть можливість для створення схеми даних, а автоматизовані процеси конвертації схеми даних в текстовий формат та генерація програмного коду забезпечать швидкий та безпомилковий спосіб отримання ORM-моделей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОСТАНОВКА ЗАДАЧІ ТА ОБГРУНТУВАННЯ ВАРІАНТІВ РІШЕННЯ

1.1 Опис предметної області

Ринок IT-технологій зараз посідає провідні місця як в усьому світі, так і зокрема в Україні [3]. Інформаційні технології розвиваються без зупину, навпаки тільки набирають оберти. Відбуваються процеси постійного вдосконалення та пристосування до сучасних потреб, світ свідомо переходить в онлайн-режим ведення справ, навчання, розваг та інших потреб життя. Якісна та швидка розробка програмного забезпечення надає можливість максимально покращити конкурентоспроможність бізнес, дозволить зайняти комфортні умови на ринку тощо.

Але створення, оновлення та підтримка програмного забезпечення це досить складний та довгий процес, який вимагає багато структурованого аналізу та проектування. Зокрема це стосується рівня роботи з даними майбутнього додатку, а саме підготовка схеми бази даних та її подальша інтеграція в програмний код.

Метою даної роботи є автоматизація більшості процесів, створення комфортних умов для роботи з рівнем даних, об’єднання ключових процесів та результатів роботи в одному місці. Це пропонується оформити у вигляді онлайн-платформи, яка надасть користувачам інструменти для проведення проектування схеми найбільш популярних баз даних, а головне – забезпечить автоматичну генерацію коду для ORM-моделей у зручному вигляді.

Автоматизація є одним з ключових з напрямів науково-технічного прогресу, який направлений на застосування саморегульованих технічних або програмних засобів, економіко-математичних методів і систем управління, які забезпечують автономність користувача від участі в процесах отримання, перетворення, передачі і використання енергії, матеріалів чи інформації, істотно зменшують міру цієї участі чи трудомісткість виконуваних операцій. Окрім автоматичної роботи, широкої популярності набрало поняття автоматизований, що підкреслює відносно великий ступінь участі людини в процесі, тобто користувач виступає регулятором автоматичних процесів системи.

Програмні чи виробничі засоби, які формують інформаційно об'єднану сукупність програмованих пристроїв автоматизованого та автоматичного контролю, регулювання та керування називаються системами автоматизації (СА) або автоматизованими інформаційними системами (АІС). Системи автоматизації мають власну класифікацію за вирішенням задач. Так одним з типів є автоматизовані інформаційні системи. Основною метою застосування більшості інформаційних автоматизованих систем є використання технологій, які здатні виконувати завдання, які не може обробити людський мозок: обробку великої кількості інформації, виконання складних обчислень і контроль багатьох одночасних процесів [4].

Кожна АІС повинна містити обов’язкові шість компонентів, які повинні бути об’єднані, а саме:

– апаратне забезпечення. До цієї компоненти відноситься ЕОМ, який часто називають центральним процесором (ЦП), адже саме він виконує основні розрахунки, а також вся його допоміжна апаратура для підтримки роботи. Інше апаратне обладнання, таке як пристрої введення і виведення, зберігання даних і зв’язку, периферійні пристрої також відносяться сюди;

– програмне забезпечення: цей термін включає комп'ютерні програми і керівництва користувача для їх підтримки. Вони є машинозчитуваними інструкціями, які направляють схему в апаратних частинах системи, щоб вони функціонували таким чином, щоб отримувати корисну інформацію з даних. Програми зазвичай зберігаються на різних носіях вводу-виводу;

– дані: це факти, які використовуються програмними засобами для отримання корисної інформації. Подібно до програм, дані зазвичай зберігаються в машинозчитуваній формі на відповідних носіях;

– процедури: це процеси та правила, які регулюють роботу комп’ютерної системи;

– люди: кожна система потребує людей, щоб вона могла бути корисною. Ймовірно, це компонент, який найбільше впливає на успіх або невдачу інформаційних систем. Він включає в себе не тільки користувачів, але і обслуговуючий персонал для роботи і обслуговування машини, підтримки даних і мережі комп’ютерів;

– зворотній зв'язок: це ще один компонент інформаційної системи, який визначає її ефективність [5].

В залежності від типу діяльності автоматизовані системи в інформаційній сфері поділяються на:

а) АСК (автоматизовані системи керування);

1) АСК технологічними процесами (АСК ТП);

2) АСК підприємствами (АСКП), виробництвом (АСКВ) тощо;

б) системи автоматизованого проектування;

1) САПР (системи автоматизованого проектування і розрахунку);

2) САПР ТП (системи автоматизованого проектування технологічних процесів);

в) АСНД (автоматизовані системи наукових досліджень);

г) АС оброблення та передавання інформації;

1) АІПС (автоматизована інформаційно-пошукова система);

2) АСІТО (автоматизована система інформаційно-термінологічного обслуговування);

д) АСТПВ (АС технологічної підготовки виробництва);

е) автоматизовані системи контролю та випробувань;

ж) АС, що поєднують функції, перелічених вище систем.

В рамках обраної предметної області рішення розглядається як система автоматизованого проектування і розрахунку, тобто такої, що призначена для автоматизації технологічного процесу проектування виробу, результатом якого є комплект проектно-конструкторської документації, достатньої для виготовлення та подальшої експлуатації об'єкта проектування.

Предметною областю кваліфікаційної роботи є веб-платформа для генерації ORM-моделей на основі схем реляційних баз даних. Таке рішення представлятиме собою сервіс для розробників для проектування та роботи з рівнем даних для будь-якого програмного забезпечення. Сама веб-платформа передбачає розподіл на дві основні компоненти. Першою компонентою є проектування схеми даних за допомогою ER-діаграм. Другою компонентою є автоматизований процес генерації коду для ORM-бібліотек на основі розроблених схем БД. Як і більшість веб-сервісів, ця веб-платформа пропонує користувачу авторизацію, яка в свою чергу розблокує частину функціоналу, зокрема збереження результатів.

1.2 Детальний опис методів та технологій застосованих для розробки системи

1.2.1 Веб-платформа

«Веб-платформа» – це набір стандартизованих API, таких як HTML, CSS, JavaScript, SVG та багато інших, які розробники використовують для побудови сайтів та веб-додатків. Крім «кореневих» технологій, платформа включає ще й локальні браузерні API, які додають у браузер нову функціональність, наприклад: BOM, DOM, Fetch-запити та інші. Особливістю всіх перелічених вище технологій є те, що вони всі взаємодіють з браузером і їх рушіями. Принцип роботи технологій у браузері описується у відповідних документах – специфікаціях. Це докладні інструкції для розробників, як має працювати якась конкретна функціональність [6].

На даний момент, поняття веб-платформи перестало асоціюватися зі стандартами та специфікаціями. Натомість стало тісно пов’язаним з продуктами які можуть бути реалізовані за допомогою вище перерахованих технологій. Так, веб-платформа представляє собою дві основні форми реалізації: веб-сайт та веб-застосунок.

Веб-застосунок представляє собою програму, одна частина якої завантажується в браузер і взаємодіє з користувачем (візуально-інтерфейсна частина), а інша знаходиться на веб-сервері, опрацьовує запити, що надходять від користувача та потім повертає на них відповідь. Частина, яка завантажується в браузер і з якою взаємодіє користувач, називається частиною клієнта або «фронтенд». На веб-сервері знаходиться серверна частина веб-програми або «бекенд». В той час як веб-сайт – це сукупність веб-сторінок, найчастіше інформаційного, статичного характеру. Він може містити контент у формі тексту, графічних зображень, аудіо або відео доріжок тощо. Веб-сайти видають користувачеві готові HTML-сторінки, доступні для перегляду, при чому взаємодія з ними обмежена [7]. Найчастіше можливо лише скористатися пошуком або передплатити новини. Результати порівняння цих двох форм представлені у таблиці 1.1.

Таблиця 1.1 – Порівняння веб-сайту та веб-застосунку

Параметр	Веб-застосунок	Веб-сайт
Основне призначення	Створюється для того, щоб взаємодіяти з користувачем	Припускає лише наявність статей, без будь-якого можливого впливу
Взаємодія з користувачем	Користувач може проводити маніпуляції з даними, однак з обмеженим доступом	Користувач може читати інформаційний контент, проте не може його ніяк міняти
Аутентифікація	Більшість веб- застосунків вимагають аутентифікації, щоб користувач міг скористатися програмою	Для цих сайтів не потрібна обов'язкова автентифікація. Сайт може лише мати пароль які надсилає адміністратор ресурсу
Завдання та складність	Веб-застосунок має багато функцій і закриває безліч проблем	Веб-сайт відображає лише статичну сторінку на якій зображена текстова інформація
Зміна проекту	Щоб додати якісь зміни до проекту, потрібно робити рев'ю всього коду і після вписувати те, що ви хочете змінити	Досить просто вносити зміни, лише зробивши пару змін в HTML коді сторінки

За допомогою технологій веб-платформи пропонуються наступні типи веб-рішень:

– лендінговий або односторінковий сайт. Посадкова сторінка або лендінг (від англійської «landing page») є спеціально спроектованою сторінкою сайту, яка призначена закликати відвідувачів сторінки до здійснення певної дії: придбання продукції, оформлення підписки, замовлення послуги тощо;

– сайт-візитка. Наразі наявність власного веб-представництва в Інтернеті є одним із основних способів просування бізнесу. Різноманітність подібних сайтів-представництв приголомшує уяву: тут і скромні персональні сторінки, і великі корпоративні ресурси, і інформаційно-розважальні портали. Однак, якщо компанія тільки починає свою роботу в мережі Інтернет, то найкращим варіантом буде використання сайту-візитки;

– корпоративні сайти. Корпоративний сайт – повноцінне представництво компанії в Інтернеті. Тут міститься повна інформація про компанію, послуги чи товари, які вона пропонує, тощо. Головна відмінність корпоративних сайтів від сайтів-візиток полягає в тому, що перший має розширений перелік функціональних можливостей і та можливі інтеграції з внутрішніми системами компанії;

– промо-сайти. Починаючи випуск нової лінійки продукції або запускаючи нову товарну пропозицію, фірми роблять все можливе для отримання лояльності та визнання від споживачів, а також розширення кола постійних покупців чи замовників. Зрозуміло, будь-яка новинка, чи то товар чи послуга, потребує реклами. Власне, для просування нових пропозицій створюють промо-сайти;

– інтернет-магазин. Багато компаній вже давно використовують інтернет-магазини для переведення продажів продукції в онлайн-простір та ведення бізнесу через інтернет. Від інших сайтів інтернет-магазини відрізняються, в першу чергу, тим, що тут представлений не тільки каталог доступних товарів, а й надається можливість їх купівлі та оплати безпосередньо на сайті. Популярність цього виду пояснюється тим, що мати свій інтернет-магазин можуть як великі компанії, так і представники малого бізнесу;

– іміджевий сайт. Іміджевий сайт – це сайт, основною метою якого є створення позитивного враження про власника за допомогою візуального оформлення. У ролі замовників іміджевих сайтів можуть виступати як великі компанії, так і представники малого та середнього бізнесу, і навіть приватні особи;

– персональний сайт. На відміну від попередніх видів персональні сторінки є представництвами не компаній, а приватних осіб. Найчастіше, подібні проекти відрізняються малим обсягом, містять у собі відомості біографічного чи особистого характеру, і навіть інформацію про рід діяльності власника чи послуги, що він надає;

– online-сервіси. Незважаючи на те, що Інтернет став загальнодоступним зовсім недавно, вже сьогодні життя без Інтернету практично неможливо уявити, тим більше, що в даний час доступ до Інтернету можливий не тільки зі стаціонарних, але і з мобільних пристроїв. І якщо раніше для оплати квитанції потрібно було відстояти довгу чергу у банку, для пошуку книги доводилося йти до бібліотеки чи книгарні, то сьогодні все вищезгадане та багато іншого можна зробити в Інтернеті;

– портали. Портали – це максимальна кількість корисної інформації, інтерактивних сервісів, зручності для відвідувачів в одному сайті. Портали – це дуже насичені сайти, на яких можна знайти все необхідне: новини, авторські блоги, голосування, пошукові та поштові сервіси та багато іншого [8].

Окрім цих типів, існує ще окремий тип веб-рішень, який представляє собою онлайн-засоби для проектування та розробки. Суть таких засобів полягає у наданні

для користувача зручного інтерфейсу, а головне – широкого функціоналу для можливості виконання поставлених задач. При чому існує безліч різноманітних засобів подібного типу. Так, наприклад сервіс Figma дає можливість виконувати проектування візуальних інтерфейсів для будь-яких програм, сервіси по типу Lucidchart надають функціонал для створення діаграм та схем різних рівнів складності, в тому числі і схем для баз даних, а Codesandbox.io та його аналоги пропонують писати та редагувати код для пришвидшення процесу розробки.

Проте результат дослідження існуючих веб-застосунків показав те, що не існує жодної системи аналогічної до визначеної в рамках предметної області. Є окремі що мають лише частину запланованого функціоналу, при чому це можуть бути не лише веб-орієнтовані платформи, але й десктопні чи навіть консольні. Це робить обрану предметну область унікальним та новим онлайн-рішенням для IT-фахівців.

1.2.2 Схема «сутність-зв'язок» реляційної бази даних

Підготовка рівня даних для будь-якого програмного забезпечення є досить складним та довгим процесом. Зазвичай, проектування бази даних складається з двох послідовних етапів:

– логічне проектування. Перетворення зібраних в результаті аналізу предметної області даних в структуру даних згідно з форматом ER-діаграми;

– фізичне проектування. Визначення особливостей збереження даних згідно до обраної СКБД.

Розробка схем бази даних може виконуватись у два різних підходи, а саме з використанням форм або без них, де під формою мається на увазі візуальне проектування схем. Веб-портал забезпечить користувачів середовищем для обох видів проектування з використанням зручного візуального інтерфейсу. Також пропонується один з двох варіантів проведення проектування фізичної схеми даних:

– комбінований. При використанні цього підходу, пропонується виконати спочатку логічне проектування схеми з використанням ER-нотації, а потім трансформувати та відредагувати схему у відповідності до обраної СКБД;

– фізичний. Цей підхід має на увазі безпосередню розробку схеми з обмеженнями обраної СКБД. Зазвичай такий підхід використовується за умови, що

планується розробка не складної системи, або розробник має достатньо досвіду роботи з обраною СКБД.

При проведені логічного проектування бази даних мається на увазі модель даних конкретної предметної галузі, що виражена незалежно від конкретного продукту керування базами даних або технології зберігання (фізична модель даних), але в термінах структур даних, таких як реляційні таблиці та колонки. Логічні моделі даних подають абстрактну структуру області інформації. Вони часто мають схематичний характер і найтиповіше використовуються у бізнес-процесах, які прагнуть захопити речі, що мають важливе для організації значення, та як вони відносяться одна до одної.

Логічна модель даних слугує джерелом інформації для фізичного проектування і таким чином забезпечує розробника фізичної бази даних засобами пошуку компромісів, необхідних для досягнення поставленої мети, що дуже важливо для ефективного проектування. Логічна модель даних відіграє також важливу роль на етапі експлуатації та супроводу вже готової системи. При правильно організованому супроводі модель даних, що підтримується в актуальному стані, дозволяє точно і наочно представити будь-які зміни, що вносяться в базу даних, а також оцінити їх вплив на прикладні програми і використання даних, що вже є в базі. При фізичному проектування бази даних (процес підготовки опису реалізації бази даних на вторинних пристроях, що запам'ятовують) розглядаються основні відносини, організація файлів та індексів, призначених для забезпечення ефективного доступу до даних, а також усі пов'язані з цим обмеження цілісності та засоби захисту.

Для графічного проектування ER-моделі даних існують безліч різних нотацій, але кожна з них характеризується тим, що обов’язково містить три основні елементи – сутність (таблиця), атрибут (поле), зв’язок. Серед усіх нотацій можна виділити 4 основі, які знайшли своє місце та здобули популярності, а саме:

– нотація Чена;
– нотація Мартіна («вороняча лапка»);
– нотація діаграми класів UML;
– нотація IDEF1X [9].

В нотації Чена сутність відображається прямокутником, її атрибути – овалами, а відносини між сутностями – ромбами. Імена ключових атрибутів підкреслюються.

На дугах можна проставити кратність зв'язків. Наприклад, за допомогою обмежень можна описати, що в навчальній групі не може бути більше 30 спортсменів. Загалом, розібратися в тому, що намальовано можна, якщо її уважно читати, проте читати діаграму дуже втомливо, адже на ній дуже багато елементів. Діаграма швидко перестає входити на лист А4, проблема виникне вже у невеликій за обсягом роботі. Також незрозуміло у якому форматі і де прописуються типи атрибутів. Для позначення обов'язкової наявності екземпляра сутності використовується «округлена стрілка», що більшість інструментів проектування не підтримують. Як висновок ER-діаграма в нотації Чена дуже трудомістка та марна, проте вона непогано відображає загальну ідею цієї діаграми [10]. Приклад нотації Чена наведено на рисунку 1.1.

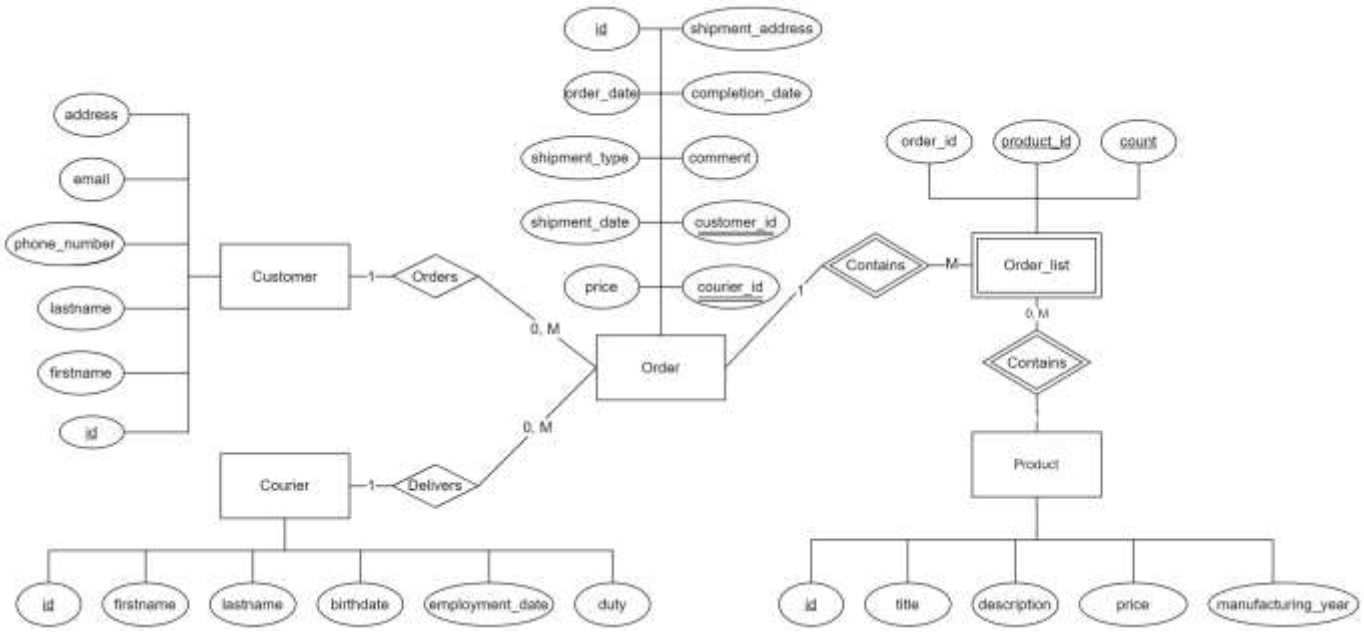

Рисунок 1.1 – Нотація Чена

Відповідно до нотації Мартіна, сутність подається у вигляді прямокутника, де сутність виражається іменником. Ім'я сутності має бути унікальним у рамках однієї моделі. Зв'язок зображується лінією, яка пов'язує дві сутності, що беруть участь у відношенні. Ступінь кінця зв'язку вказується графічно, множинність зв'язку зображується у вигляді «вилки» на кінці зв'язку. Модальність зв'язку також зображується графічно — необов'язковість зв'язку позначається кружком на кінці зв'язку. Атрибути сутності записуються всередині прямокутника, що зображує сутність і виражаються іменником в одному (можливо, з уточнюючими словами).

Серед атрибутів виділяється ключ сутності — ненадлишковий набір атрибутів, значення яких у сукупності є унікальними для кожного екземпляра сутності. За рахунок цього діаграма виходить набагато компактнішою. Відношення більше не зображуються як ромбів, а записуються як тексту над стрілкою [11]. Приклад нотації Мартіна наведено на рисунку 1.2.

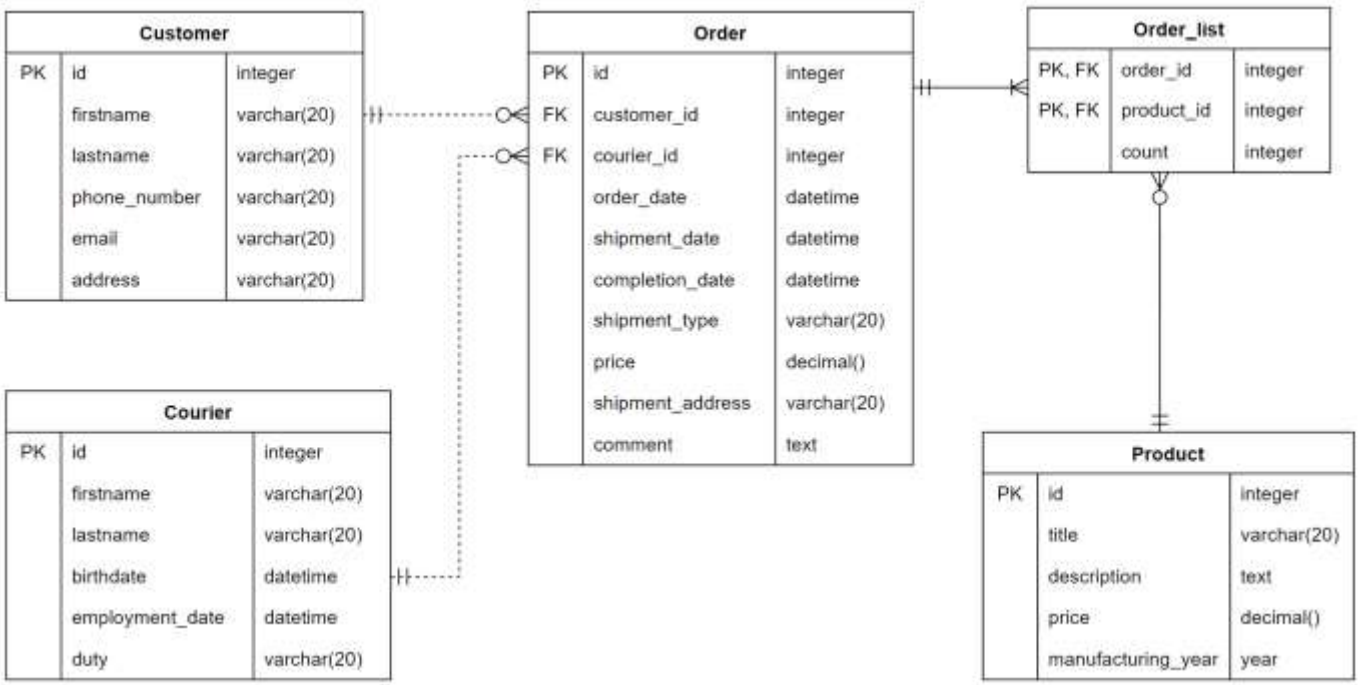

Рисунок 1.2 – Нотація Мартіна («вороняча лапка»)

Нотація діаграми класів багато в чому схожа на нотацію Мартіна. Однак, її застосування для моделювання предметної галузі та структури БД має низку особливостей:

– сутності та його властивості зображуються так само як і нотації Мартіна (прямокутник з секціями). Нижня (третя) секція символу класу частина залишається порожньою, оскільки вона призначена для запису функцій класу, однак іноді в неї містять опис індексів і тригерів БД зі структурами-стереотипами <<index>> та <<trigger>>;

– використовується дуже маленьке підмножина системи позначень, оскільки предметної області немає особливого сенсу віртуальні функції, абстрактні класи, статичні поля, агрегація, закрите успадкування;

– зазвичай виникає потреба у відображенні обмежень, для цього використовуються стереотипи <<PK>> (первинний ключ), <<FK>> (зовнішній ключ) та <<PK, FK>> [12].

Приклад нотації діаграми класів UML наведено на рисунку 1.3.

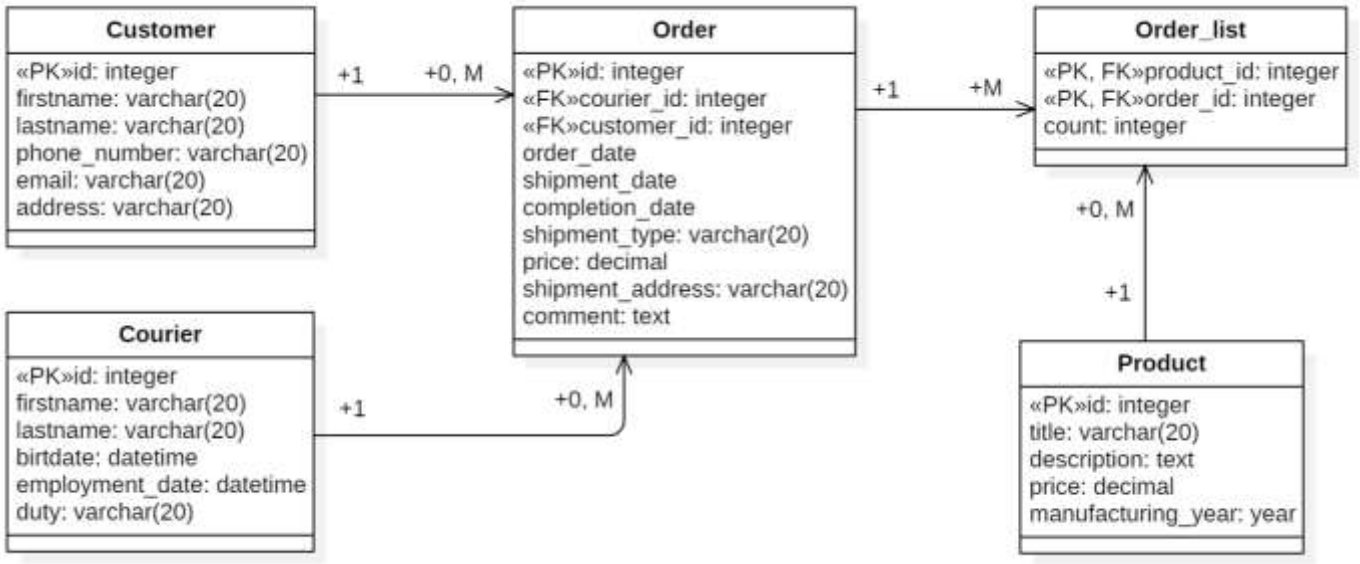

Рисунок 1.3 – Нотація діаграми класів UML

IDEF1X використовується для формування графічних уявлень інформаційних моделей, що відображають структуру та семантику інформації всередині середовища або системи. IDEF1X дозволяє будувати семантичні моделі даних, які можуть служити для підтримки управління даними як ресурсом, інтеграції інформаційних систем та побудови комп'ютерних баз даних.

Сутності зображуються у вигляді заокруглених прямокутників. Домени представляють іменований набір значень даних (фіксованого розміру або, можливо, нескінченних) одного й того самого типу даних, на основі якого будується фактичне значення для екземпляра атрибута. Кожен атрибут має бути визначений лише в одному базовому домені. Декілька атрибутів можуть бути засновані на тому самому базовому домені. Зображуються на діаграмі у вигляді овалу. Можуть бути комбіновані стрілками. Атрибути відображаються у вигляді іменованого списку всередині сутності та можуть представляти використання домену у тих сутності. Зв'язок між екземплярами двох сутностей або між екземплярами однієї і тієї ж сутності. Літера P

(positive) означає потужність "один або багато" і міститься біля точки. Літера Z (zero), вміщена біля точки, означає потужність "нуль чи один". Якщо потужність точно дорівнює деякому числу N, це число (ціле, позитивне) міститься біля точки. Ідентифікуюче відношення між сутністю-батьком і сутністю-нащадком зображується суцільною лінією. Якщо існує ідентифікуюче відношення, то сутність-нащадок завжди є залежною від ідентифікатора сутністю, що зображується блоком із закругленими кутами. Пунктирна лінія зображує неідентифікуюче відношення між сутністю-батьком і сутністю-нащадком. У неідентифікуючому відношенні і сутність-батько, і сутність-нащадок будуть незалежними від ідентифікаторів сутностями, якщо тільки хоча б одна з них не буде сутністю-нащадком в іншому іншому відношенні, що є ідентифікуючим ставленням [13]. Приклад нотації діаграми класів UML наведено на рисунку 1.4.

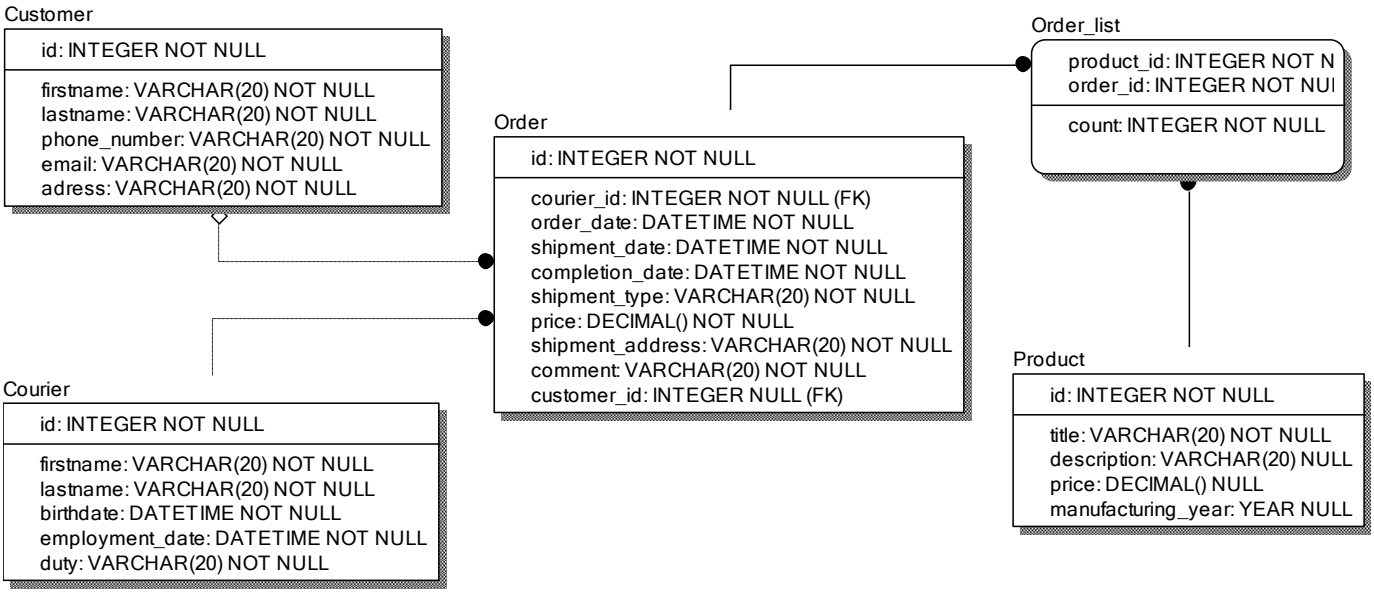

Рисунок 1.4 – Нотація IDEF1X

1.2.3 Технологія ORM

Object-Relational Mapping або ORM представляє собою підхід в програмуванні, що створює тісний зв’язок між реляційною базою даних та концепцією ООП. Ключовою особливістю ORM є відображення, яке використовується для прив'язки об'єкта до даних у БД. ORM створює «віртуальну» схему бази даних у пам'яті і

дозволяє маніпулювати даними вже на рівні об'єктів. Відображення показує як об'єкт та його властивості пов'язані з однією чи кількома таблицями та його полями у базі даних. ORM використовує інформацію цього відображення для керування процесом перетворення даних між базою та формами об'єктів, а також для створення SQL-запитів для вставки, оновлення та видалення даних у відповідь на зміни, які додаток вносить до цих об'єктів [14].

Бібліотеки ORM існують для різних мов програмування. Загалом технологія ORM дозволяє проектувати роботу з даними в термінах класів, а не таблиць даних. Вона дозволяє перетворювати екземпляри класів на дані, придатні для зберігання базі даних, причому схему перетворення визначає сам розробник. Засобами ORM-бібліотеки розробник має підготувати спеціальний файл з кодом для кожної таблиці на схемі. Клас чи об’єкт який представляє собою реалізацію таблиці в коді називається ORM-моделлю. При підготовці ORM-моделі, в її класі описується основна інформація, щодо таблиці, яку вона представляє, найголовніші з них це:

- назва таблиці, під якою звертатись в запитах;
- поля таблиці;
- типи даних полів;
- індекси полів;
- можливість зберігати NULL- значення;
- зв’язок поточної моделі з іншими.

Колекція таких файлів з класами відповідними до таблиць формують віртуальну базу даних в проекті.

Більша частина ORM-бібліотек побудована з використанням одного з двох основних шаблонів проектування – Active Record та Data Mapper.

Active Record – це шаблон проектування або один із шаблонів застосунків, який не несе відповідальності за представлення бізнес-логіки та даних. Active Record дозволяє створювати та використовувати більш просто ті об’єкти, які вимагають постійного зберігання в базі даних. Якщо говорити у відношенні шаблону MVC, то Active Record реалізує модель.

Як правило, всередині моделі, яка реалізує Active Record, прописано властивість, які можуть бути і приватними, тоді окремо створюють сеттери та геттери під кожну власність. Як правило, модель Active Record - це відображення полів моделі

на поля в базі даних. В Active record сама модель відповідає за збереження даних у базі даних. Писати код з використанням цього підходу вдається швидко і легко, в тому випадку, коли властивості об'єкта прямо співвідносяться з колонками в базі даних, а збереження відбувається в одному місці, що дозволяє легко дослідити та дізнатись принцип роботи.

Але водночас моделі цього шаблону порушують принципи SOLID. Зокрема, принцип єдиної відповідальності (Single Responsibility, S). Відповідно до принципу, доменний об'єкт повинен мати лише одну зону відповідальності, тобто лише свою бізнес-логіку. Викликаючи його для збереження даних, ви додаєте йому додаткову зону відповідальності, збільшуючи складність об'єкта, що ускладнює його підтримку та тестування. Окрім цього, реалізація збереження даних тісно пов'язана з бізнес-логікою, а це означає, що якщо необхідно буде використовувати іншу абстракцію для збереження даних (наприклад, для зберігання даних у XML-файлі, а не в базі даних), то доведеться робити рефакторинг коду.

Data Mapper – це шар доступу до даних, який надає двонаправлений мапінг даних між постійним сховищем даних (зазвичай, це SQL база даних) і зберіганням даних у пам'яті.

На відміну від Active Record, у Data Mapper з'являється ще один шар або тип сутності такий як entityManager. Саме цей шар відповідатиме за перенесення стану моделі до бази даних і назад. Таким чином, кожен об'єкт має свою зону відповідальності, тим самим дотримуючись принципів SOLID і зберігаючи кожен об'єкт простим і по суті. Бізнес-логіка та збереження даних пов'язані слабко, і якщо ви хочете зберігати дані в XML-файл або якийсь інший формат, ви можете просто написати новий Mapper, не торкаючись доменного об'єкта. Але така гнучкість, простота тестування та налагодження компенсується тим, що доведеться набагато більше думати перед тим, як написати код. У результаті у вас більше об'єктів в управлінні, що трохи ускладнює код та його налагодження [15].

Серед переваг використання ORM в якості додаткового рівня даних виділяють:

– наявність явного опису схеми БД, подана в термінах будь-якої мови програмування, яка знаходиться та редагується в одному місці;

– можливість оперувати елементами мови програмування, тобто класами, об'єктами, атрибутами, методами, а не елементами реляційної моделі даних;

– можливість автоматичного створення SQL-запитів, яка позбавляє необхідності використання мови для опису структури БД (Data Definition Language) та мови маніпулювання даними (Data Manipulation Language) при проектуванні БД та зміні її схеми відповідно;

– не потрібно створювати нові SQL-запити при перенесенні на іншу систему управління базами даних, оскільки це відповідає низькорівневий драйвер ORM.

– ORM позбавляє необхідності роботи з SQL і опрацювання значної кількості програмного коду, який часто одноманітний і схильний до помилок.

– код, що генерується ORM гіпотетично перевірений та оптимізований, отже не потрібно турбуватися про його тестування;

– розвинені реалізації ORM підтримують відображення успадкування та композиції на таблиці;

– ORM дозволяє ізолювати код програми від деталей зберігання даних.

Але будь-яка технологія має як переваги, так і недоліки і ORM не виключення:

– додаткове навантаження на програміста, якому, у разі використання ORM, необхідно вивчати цей якийсь «додатковий шар» між програмною та базою даних, який до того ж створює додатковий рівень абстракції — об'єкти ORM. У зв'язку з цим можуть виникнути питання відповідності особливостям ООП та відповідним реляційним операціям. Цю проблему називають «impedance mismatch», а сама реалізація ORM веде до збільшення обсягу програмного коду та зниження швидкості роботи програми. Однак, з іншого боку, ORM наочно та в одному місці концентрує різницю між реляційною та об'єктно-орієнтованою парадигмами, що не можна назвати недоліком;

– поява помилок у програмі, що важко піддаються налагодженню, якщо присутні помилки в реалізації ORM, наприклад, помилки в реалізації кешування ORM, такі як узгодження змін у різних сесіях;

– відсутня можливість написати в явному вигляді SQL-запит;

– потрібні окремі таблиці у разі прямого відображення класів у таблиці та необхідності відображення атрибутів множинного характеру.

Провівши аналіз вищенаведених мінусів можна встановити, що деякі з них можуть бути вирішені або не представляють собою критичні проблеми. Так, проблема з комплексними атрибутами в сутностях та їх підтримка у віртуальній базі даних може

бути проігнорована. Це пов’язано з тим, що у більшості випадків використовуються прості типи даних для полів, а складні типи даних на кшталт JSON або JSONB вже підтримуються ORM-модулями. А будь-які проблеми з неможливістю виконання спеціального SQL-запиту вирішується можливістю виконання цього запиту за допомогою ODBC-модулю, який в будь-якому випадку буде використовуватись для під’єднання до БД.

Щодо двох перших мінусів, то метою даної веб-платформи і є вирішення цих недоліків. По-перше вона надає можливість автоматично генерувати код для ORM-моделей на основі заздалегідь сформованої схеми бази даних. Таким чином це сильно зекономить час розробників на створення відповідних програмних файлів для кожної таблиці, особливо для великих за кількістю сутностей баз даних. По-друге автоматична генерація, яка виконується по заздалегідь визначеним і перевіреним алгоритмах вбереже розробників від більшості потенційних помилок в коді, зокрема від тих, що стосуються логічних та синтаксичних помилок.

1.3 Аналіз існуючих рішень для роботи зі схемами РБД та ORM

На стадії аналізу предметної області часто виконується пошук та оцінка існуючих аналогів чи подібних систем. Метою даної задачі є пошук сильних та слабких сторін конкурентів під час їх вивчення. Результат виконання застосовується на стадіях проектування та розробки, щоб на виході отримати систему, яка буде позбавлена об’єктивним недоліків та міститиме найкраще з конкурентних систем [16]. Під час аналізу можуть бути виявлені певні бізнес-функції, критичні баги чи так звані «killer features».

На першому етапі було визначено перелік систем, які мають подібний функціонал до проектованої інформаційної системи. Але не було знайдено жодної системи, що мала б водночас функціонал по проектуванню схеми БД та по генерації ORM-моделей, що в черговий раз підтверджує новизну. Тому було прийнято рішення знайти аналоги для окремих частин системи, а саме цікавлять процеси проектування схем реляційних баз даних та генерації ORM-моделей.

Процес вибору систем з можливістю генерації ORM-моделей був дуже складним та навіть неможливим, адже цьому перешкоджали відразу декілька причин.

По-перше, платформи для генерації моделей представляють собою певне консольне рішення в рамках бібліотеки конкретної мови програмування, а це значить ускладнює його практичне дослідження. По-друге, серед існуючих засобів, які навіть мають хоч якийсь графічний інтерфейс для роботи реалізована генерація лише для однієї, конкретної ORM-бібліотеки, замість певної глобальної реалізації. По-третє, найбільш важливим фактором, що блокує можливість дослідження подібних систем є те , що процес генерації файлів інкапсульований всередині самого засобу генерації.

На відміну від систем з можливістю генерації ORM-моделей, процес вибору систем для проектування схем реляційних баз даних зіткнувся з іншою проблемою: існує безліч систем, які пропонують подібний функціонал, кожен з яких містить свої особливості. Альтернативи було обрано з списку актуальних і сучасних засобів проектування схем реляційних баз даних незалежно від його характеристик, відгуків та популярності. Одною характерною рисою для всіх вищеперерахованих рішень є те, що вони вміють працювати готувати схеми для різних РБД, а не специфікуються на одній. Так було сформовано наступний список:

- Erwin Data Modeler;
- Lucidchart;
- SAP PowerDesigner [17].

Erwin Data Modeler представляє собою CASE-засіб для проектування та документування баз даних. Моделі даних допомагають візуалізувати структуру даних, забезпечуючи ефективний процес організації, управління та адміністрування таких аспектів діяльності підприємства, як рівень складності даних, технологій баз даних та середовища розгортання.

Користувачі можуть використовувати Erwin Data Modeler як спосіб створення концептуальної моделі даних та створення логічної моделі даних, яка залежить від конкретної технології бази даних. Ця схематична модель може бути використана для створення фізичної моделі даних. Потім користувачі можуть направити інженерну мову визначення даних, необхідний створення схеми для низки систем управління базами даних. Програмне забезпечення включає функції для графічної модифікації моделі, у тому числі діалогові вікна для вказівки кількості взаємозв'язків сутностей, обмежень бази даних, індексів і унікальності даних. Erwin підтримує три мови

моделювання даних: IDEF1X, варіант розробки інформаційних технологій, та форму позначення розмірного моделювання.

Недоліки засобу Erwin Data Modeler:

– десктопне рішення;
– незручна та неінтуїтивна система створення звітів;
– проблеми з підтримкою шрифтів та локалізацій;
– незручний та застарілий візуальний інтерфейс;
– слабка підтримка великих схем РБД;
– погано реалізована підтримка сумісної роботи;
– висока вартість базового функціоналу продукту;
– підтримка лише ОС Windows.

Переваги засобу Erwin Data Modeler:

– доступне логічне і фізичне моделювання;
– широкий інструмент візуального моделювання;
– можливість реверсного проектування БД;
– наявність базових прикладів схем БД;
– наявність веб-рішення для моделювання та представлення схем [18].

Lucidchart – це візуальний робочий простір, який поєднує у собі створення діаграм, візуалізацію даних та спільну роботу для прискорення розуміння та стимулювання інновацій. LucidChart орієнтована на веб програмне забезпечення, яке дозволяє користувачам об'єднуватися та працювати разом над створенням блоксхем, організаційних діаграм, шаблонів вебсайтів, UML-діаграм, ментальних карт, прототипів програмного забезпечення та багатьох інших типів діаграм [19].

Недоліки засобу Lucidchart:

– відсутність фізичного моделювання;
– слабкий набір підтримуваних РБД та їх версій;
– слабкоадаптований функціонал для проектування схем БД.

Переваги засобу Lucidchart:

– веб-рішення;
– можливість реверсного проектування;
– можливість поділитися проектом;

– можливість безкоштовного використання без обмежень по часу;
– наявність окремих тарифних планів для використання;
– можливість використання на будь-якій ОС [20].

SAP PowerDesigner (PowerDesigner) — це засіб для спільного корпоративного моделювання, в тому числі і РБД-проектування. PowerDesigner підтримує проектування програмного забезпечення за допомогою архітектури, керованої моделлю. PowerDesigner зберігає моделі у файлах із різними розширеннями, такими як «.bpm», «.cdm» та «.pdm» в залежності від обраного рівня проектування. Завдяки таким можливостям, як моделювання даних, зв'язування та синхронізація, а також керування метаданими, можна одразу охопити рівні архітектури та вимоги, отримати доступ до сховища метаданих і поділитися відкриттями зі своєю командою [21].

Недоліки засобу SAP PowerDesigner:
– десктопне рішення;
– відсутність підтримки нових баз даних або їх нових версій;
– відсутність прикладів схем;
– надзвичайно висока вартість базового функціоналу продукту;
– низька швидкість роботи;
– підтримка лише ОС Windows.

Переваги засобу SAP PowerDesigner:
– широкий функціонал стилізації форм та схеми;
– можливість експорту у вигляді конфігураційного файлу;
– доступне концептуальне, логічне і фізичне моделювання;
– налагоджена система репортингу помилок БД;
– наявність демо-використання;
– можливість реверсного проектування [22].

В результаті огляду та аналізу усіх рішень для проектування та роботи зі схемами БД можна підсумувати, що лише в одному засобі якісно реалізована система сумісної роботи, що може стати корисним функціоналом. Також десктопні засоби пропонують проектування щонайменше двох типів схем, що було б бажано реалізувати для веб-додатку. Серед очевидних недоліків, які було виявлено полягає в підтримці різними ОС. Настільні додатки підтримуються лише ОС Windows, в той час

як веб-платформа автоматично пропонує кросплатформеність завдяки використанню браузерів. Найбільш оптимальний підхід з монетизацією сервісу представлений у Lucidchart, а саме пропонування різних об’ємів функціональності у вигляді багаторівневої підписки. Добре розвинуті системи звітування та можливість реверсного проектування БД можуть слугувати як додатковий розвиток веб-платформи в майбутньому.

1.4 Визначення області застосування системи

Веб-платформа для проведення генерації ORM-моделей на основі схем РБД вже має місце, адже це унікальне рішення в рамках даної предметної області, бо не містить подібних аналогів. І водночас система вирішує актуальні проблеми при використанні ORM-бібліотек, а саме збереження часу формування програмних файлів моделей та забезпечення правильності їх оформлення.

Програмний засіб надає можливість в проведені проектування схеми бази даних, при чому як на логічному рівні, так і на фізичному. Кількість підтримуваних СКБД для фізичного проектування може постійно оновлюватись та розширюватись, таким чином охоплюючи ще більше потенційних користувачів. Автоматизація процесу конвертації логічної схеми в фізичну пришвидшить процес підготовки схеми даних. Впродовж усього процесу моделювання, в скритному режимі, система генерує конфігураційний файл, який містить всю інформацію, яку користувач відобразив на схемі. Функціонал по генерації моделей під заздалегідь обрану ORM-бібліотеку не є обов’язковим для виконання в системі, користувач може обмежитись на етапі підготовки схем та експортувати результати моделювання. Але це є саме тим процесом, який робить платформу унікальною в плані наявності конкурентних систем. Генерація використовує результати попереднього етапу, а саме конфігураційний файл-представлення фізичної моделі даних, проводить його парсинг, аналізує ключові елементи та конвертує їх у вигляді коду. По завершенню, користувач зможе використовувати завантажити результати генерації та використати їх за призначенням. Наостанок потрібно згадати те, що завдяки авторизації на веб-платформі, користувач зможе розширити функціонал системи для себе. Це стосується зокрема створенню проектів, в який входять та зберігаються результати роботи,

можливість його командного використання, збереження та завантаження, що підвищує комфорт використання системи як одному, так і в групі.

При розробці дизайну для веб-платформи необхідно орієнтуватись в першу чергу на зручність та інтуїтивність графічного інтерфейсу платформи, щоб користувачу не було складно знайти потрібні елементи на панелі проектування, зрозуміти суть пунктів навігаційного меню, натиснути правильну кнопку для запуску того чи іншого процесу. Поняття «User Interface» та «User Experience» повинні бути реалізовані на найвищому рівні, щоб забезпечити комфортні умови роботи для користувача з різним досвідом роботи в веб-додатках та програмних засобах в цілому. В той же час, підготовкою адаптивного дизайну для підтримки інших апаратних платформ, таких як смартфони чи планшети, можна знехтувати, адже використання веб-платформи не призначене для них.

В якості основної цільової аудиторії звичайно розглядаються розробники інформаційних систем та систем, які використовують в архітектурі додатків рівень даних. Це можуть бути RDB-проектувальники для яких важливо якісно провести проектування схеми майбутньої бази даних, чи Back-end розробники які хочуть максимально швидко інтегрувати віртуальну базу даних в проект з використанням обраної ORM-бібліотеки. Але, окрім професійно-орієнтованих користувачів та працевлаштованих працівників IT-компаній, система націлена на будь-яких користувачів, яким потрібна подібна платформа. Це можуть бути студенти університетів, школярі чи новачки в сфері розробки ПЗ.

В результаті, дана веб-платформа може стати як єдине універсальне місце для підготовки рівня даних при розробці програмного забезпечення, особливо при командному використанні в IT-компанії.

2 ОГЛЯД МЕТОДІВ ТА ТЕХНОЛОГІЙ, ЯКІ ЗАСТОСОВУЮТЬСЯ В ПРЕДМЕТНІЙ ОБЛАСТІ

2.1 Огляд методів візуального проектування схем та діаграм

Для представлення певної цифрової інформації чи абстрактних об’єктів, зазвчиай використовуються два основні способи моделювання:

– текстовий опис;

– графічне зображення.

Візуальне моделювання більш переважне у ситуаціях, у яких переваги зображень переважують їхні недоліки.

По-перше, графічне відображення дозволяє створювати та підтримувати єдиний образ при роздумах, тобто у процесі аналізу та проектування. Створюючи діаграму, наявний простір використовується у необхідному форматі, створюючи довільні зв'язки між будь-якими об'єктами. З використанням звичайного тексту це робити набагато складніше. При цьому зображення дозволяють краще концентруватися. За допомогою діаграми можна тримати у фокусі уваги всі цікаві аспекти моделі, а перемикання між різними об'єктами займає частки секунди. Тому можна зосередитись на потрібних на даний момент деталях, не боячись втратити інші.

По-друге, діаграми – це ефективний засіб обміну інформацією під час обговорення. Не дарма на відомому графіку ефективності комунікацій перемагає спосіб «дві людини біля дошки». Діаграми компактні, місця для них треба порівняно небагато (на відміну від лінійного тексту та таблиць). Малювати виходить набагато швидше, ніж писати текст. Малювати можна спільно. Внаслідок обговорення з візуалізацією у всіх учасників формується єдина, зрозуміла всім модель з мінімумом різночитань. Загалом, якщо ви досі не малюєте під час обговорень, починайте прямо з наступної зустрічі.

По-третє, діаграми – це добрий спосіб документування. Діаграми дозволяють швидко розібратися в складних системах, а також служити довідковими картами, що дозволяють орієнтуватися в них. Такі карти часто генеруються безпосередньо з існуючих систем, вирішуючи вічну проблему актуальності документації: діаграма бази даних генерується із самої БД, діаграма класів – з вихідного тексту тощо.

Нарешті, по-четверте, діаграми можна використовувати безпосередньо для програмування. Будь-який проект починається з ретельного планування та візуалізації алгоритму дій. У ряді ситуацій є сенс вдатися до використання діаграм. Серед розробників часто використовуються блок-схеми чи UML-діаграми, які потребують зручного та якісного опрацювання.

До найбільш популярних методів для розташування елементів на площині робочого вікна належать наступні:

- «Drag and Drop»;
- «Point and Click»;
- координатне розміщення;
- автоматичне розміщення.

2.1.1 Метод розміщення «Drag and Drop»

«Drag and Drop» – це механіка виконання певних дій у рамках графічного інтерфейсу користувача (GUI), що передбачає використання комп'ютерної миші або сенсорного екрану. У дослівному перекладі з англійської мови означає «тягни та кинь». Дія виконується шляхом оперування видимими на екрані комп'ютера об'єктами за допомогою миші. Суть дії полягає у виборі певного віртуального об'єкту та перетягування його з одного місця в інше і таким чином виконання певних дій або в програмі, або у взаємодії кількох програм, або у візуальному переміщенні об’єкту. Типовими прикладами використання підходу «Drag and Drop» дій є переміщення об'єкта, наприклад перетягування файлу із файлового менеджера у вікно програми чи те чи інше місце ієрархії, перетягування виділеного тексту в редактори з одного місця в інше, якщо це відбувається за допомогою миші. Так само за допомогою цього підходу можна формувати схему бази даних, обираючи необхідну таблицю на робочій формі та переміщати її. Перенесення мишкою може замінити цілу послідовність кліків. Він спрощує зовнішній вигляд інтерфейсу: функції, що реалізуються через Drag-n-Drop, інакше зажадали б додаткових полів, віджетів тощо. А сам процес супроводжується плавною анімацією, яка дає змогу побачити положення об’єкту по завершенню дії.

Для переміщення обраного об’єкту по робочій поверхні необхідно використовувати один з маніпуляторів ведення курсору – маніпулятор-миша, сенсорний екран чи сенсорна панель.

Сам принцип дії підходу складається з наступного порядку дій:

– користувачу необхідно обрати потрібний об’єкт;

– по обраному об’єкту потрібно натиснути лівою кнопкою миші не відпускаючи її;

– з натиснутою кнопкою потрібно перевести об'єкт на бажане місце робочої області;

– по досягненню бажаної позиції ліву кнопку миші можна відпускати.

З точки зору програмної реалізації це відносно простий спосіб. Більшість платформ та мов програмування вже мають реалізовані інтерфейси та API для створення такого підходу взаємодії. Так, наприклад у зв’язку з поширенням популярності, для зручності реалізації, у новій специфікації HTML5 додали оновлений Drag and Drop API. Але все рівно потрібно визначати певний перелік подій, які викликаються під час обробки поведінки методу.

2.1.2 Метод розміщення «Point and Click»

«Point and Click» є одним із підходів розміщення заздалегідь обраного об’єкта чи елементу на спеціально виділеному просторі. Для роботи на графічному інтерфейсі також необхідне використання або маніпулятора-миші, сенсорного екрану чи сенсорної панелі. Переклад підходу з англійської означає «вкажи та натисни (клікни)». Принцип та порядок дій даного методу складається з наступних кроків:

– обрати необхідний елемент для розміщення натиснувши на нього лівою кнопкою миші;

– перевести курсор на місце робочої області, де необхідно розташувати обраний елемент;

– натиснути лівою кнопкою миші, щоб розмістити обраний елемент на робочій області.

При обранні точки розміщення об’єкту, користувач побачить результат розміщення лише по завершенню. Саме розміщення елементу на обраній точці

робочого простору виконується по зафіксованій ключовій точці обраного об’єкту. В якості такої точки зазвичай виступає центр елементу або його верхній лівий куток.

Для переміщення обраного об’єкту по робочій поверхні необхідно використовувати один з маніпуляторів ведення курсору – маніпулятор-миша, сенсорний екран чи сенсорна панель.

Даний метод є одним з найпростіших з точки зору програмної реалізації. Для його роботи потрібно розробити спосіб для зчитування координат положення курсору під час натискання по робочій поверхні та перевірити, чи був обраний візуальний елемент для розміщення. Після цього потрібно отримати дані про розташування ключової точки обраного елементу та зіставити її з координатами курсору.

2.1.3 Координатне розміщення

Координатне розміщення, або розміщення по координатній сітці – це один з варіантів переміщення або розташування елементу на координатній площині. Головною передумовою для роботи даного методу є реалізація робочого простору з координатною сіткою, або визначення поточних координат курсору, тобто можливість визначити позицію точки для розміщення елементу. Порядок дій для розташування елементу на робочій поверхні складається з наступних кроків:

– обрати необхідний елемент для розміщення натиснувши на нього лівою кнопкою миші;

– визначити на координатній площині необхідні значення по обох осях;

– змістити фокус на поля вводу кординат;

– ввести обрані значення координат;

– підтвердити вказані значення та розмістити елемент.

Для використання цього методу не потрібна наявність будь-яких периферійних пристроїв для наведення курсору.

Простий спосіб для реалізації та подібний до «Point and Click», тим що замість зчитування координат курсору при клікі, тут координати потрібно зчитувати зі спеціальних полів, де користувач їх вказав при підтвердженні вибору. Обов’язково потрібно додавати валідацію значень координат, бо це може призвести до некоректної поведінки.

2.1.4 Автоматичне розміщення

Останнім методом для розміщення елементів на робочій площині є підхід з автоматичним розміщенням. Суть підходу полягає в тому, що сервіс, який надає можливість проектування на площині самостійно виконує розташування елементів. Тобто після того, як користувач обрав елемент, вказав необхідні налаштування для нього та підтвердив свій вибір, система автоматично його додасть на робочу область. При цьому будуть враховані вже існуючі елементи, зв’язки між ними та інші фактори, що впливають на місце розміщення.

Розміщення об’єкту на робочій площині виконується автоматично засобами програмного забезпечення. Використання додаткових інструментів для розміщення не потрібні.

Користувачу необхідно обрати елемент, який він захоче додати на робочу область та виконати його необхідні налаштування. По підтвердженню вибору, система автоматично замість користувача розмістить його на робочій поверхні з урахування існуючих елементів.

Найскладніший спосіб для реалізації, адже для нього потрібно писати окремий алгоритм для автоматизованого розміщення елементів на робочій поверхні. Потрібно враховувати обраний елемент, його розміри, існуючі елементи схеми та їх поточне розміщення на робочій поверхні. А при наявності стрілок чи зв’язків необхідно ще враховувати реалізацію алгоритму оптимізації розміщення з метою зробити якомога менше перетинів та перекривань.

2.2 Огляд форматів представлення схеми даних у вигляді конфігураційного файлу

Конфігураційні файли є невід’ємною частиною будь-якого програмного засобу. Вони призначені для налаштування параметрів, ініціалізації деяких комп’ютерних програм чи для збереження системної інформації у структурованому вигляді. Вони використовуються для застосунків, серверних програм і налаштувань ОС.

Деякі програми надають інструменти для створення, змінення та перевірки синтаксису файлів конфігурації; іноді вони мають графічний інтерфейс. Для інших

програм системні адміністратори можуть створювати та змінювати файли вручну за допомогою текстового редактора, оскільки багато з них є простими текстовими файлами. Для серверних процесів і налаштувань операційної системи часто не існує стандартного інструменту, але операційні системи можуть надавати власні графічні інтерфейси.

Деякі програми зчитують файли конфігурації лише під час запуску. Інші періодично перевіряють конфігураційні файли на наявність змін. В окремих випадках користувач може ініціювати примусове читання файлу конфігурації та застосування змін до поточного процесу, або навіть читання довільних файлів як файлів конфігурації. Щодо цього немає чітких стандартів чи домовленостей.

В даному випадку такий конфігураційний файл необхідний для збереження усієї інформації зі створеної схеми бази даних. Під час проектування схеми, усі графічні елементи, їх налаштування та взаємодію, система зберігає в свою базу даних. По завершенню проектування або коли користувач прийме рішення почати генерацію файлів моделей, система виконує генерацію конфігураційного файлу, який перетворить усю заздалегідь зібрану інформацію у відформатований, текстовий формат. Даний файл повинен бути незначним за розміром, незалежно від кількості об’єктів моделі, швидко передаватися на сторонні сервіси, підтримувати базові ієрархічні структури та легко парситись. Адже ключовим його значенням є швидка, легка та повноцінна передача інформації про схему даних.

Існує низка форматів серіалізації загального призначення, які можуть подавати складні структури даних у форматі, який легко зберігати, і вони часто використовуються як основа для конфігураційних файлів, особливо в програмах і бібліотеках з відкритим вихідним кодом і кросплатформових. Специфікації, що описують ці формати, регулярно публікуються. Найбільш відомим їх представниками є XML, JSON та YAML [23, 24].

2.2.1 Формат XML

XML є видом мови розмітки, що розширюється. Такі файли формату, являють собою документи, що використовують теги з метою визначення об'єктів, а також їх

атрибутів. XML формат, на відміну HTML, наділяє користувача можливістю самостійно задавати теги, які застосовує мову XML.

Розширювана мова використовує структуру, в якій документ XML зберігатиме дерево елементів. Деякі з таких елементів будуть мати різні атрибути та інший вміст. XML-файл, а також інші файли розширенням, створеним на основі мови розмітки XML, мають неймовірно широке поширення серед користувачів [25].

Подібний формат файлу зберігає найрізноманітніші види інформації. Застосування документа XML у мережі Всесвітньої павутини служить з метою обміну інформацією. Використовується мова розмітки, що розширюється, і для обміну даними між програмними комплексами, варто відзначити, що саме для цього розробниками, свого часу, було створено розширення XML.

Схеми XML мають безліч вбудованих типів даних. Найчастіше використовуваними є такі типи:

– xs:string;
– xs:decimal;
– xs:integer;
– xs:boolean;
– xs:date;
– xs:time.

Щоб забезпечити спільне використання даних у форматі XML, стандартний набір імен тегів та атрибутів елементів має бути узгоджений з іншими користувачами, щоб усі члени групи використовували та застосовували теги аналогічним чином. Одним із способів забезпечити це є застосування DTD-файлів (Document Type Definition). DTD-файл є набором елементів і атрибутів, якими можуть користуватися члени робочої групи. Він також визначає правила знаходження елементів у ієрархічній структурі.

Серед переваг використання цієї мови для зберігання інформації про елементи схеми бази даних можна віднести наступне:

– ієрархічна структура XML підходить для опису практично будь-яких типів документів;

– підтримка атрибутів, тегів та областей імен розширює можливості мови для опису структур даних;

– W3C підтримує XML, який підтримується основними постачальниками програмного забезпечення. Він також використовується у дедалі більшій кількості галузей;

– мова розмітка містить готові реалізації парсерів для всіх сучасних мов програмування;

– існує стандартний механізм перетворення XSLT, реалізації якого інтегровані в браузери, операційні системи, web-сервери;

– значна кількість доступних кодувань;

– є стандартом передачі даних в SOAP-сімействі протоколів.

Але окрім переваг, даний формат представлення структури даних містить і достатню кількість недоліків, а саме:

– розмір документа XML значно більше, ніж документа в альтернативних текстових форматах передачі даних і особливо у форматах даних, оптимізованих для конкретного випадку використання;

– складність внесення змін в структуру власноруч, важка читабельність;

– надмірність XML може вплинути на ефективність програми. Зростає вартість зберігання, обробки та передачі даних;

– важко отримати значення, необхідно провести парсинг задля його отримання;

– для великої кількості завдань не потрібна вся потужність синтаксису XML і можна використовувати значно простіші та продуктивніші рішення [26].

Отже, про XML як формат представлення даних можна зробити висновок, що це дуже всесвітньо відомий, потужний, та поширений інструмент, який надає користувачу великий набір можливостей для представлення даних у потрібному вигляді, на кшталт атрибутів, тегів, областей імен тощо. Але його «багатогранність» в деяких випадках обертається мінусом, адже це призводить до значних проблем з парсингом файлу, його великих розмірів, складної валідації та роботи з ним в цілому.

2.2.2 Формат JSON

За представлення структурованих даних на основі синтаксису JavaScript відповідає стандартний текстовий формат під назвою JSON, абревіатура якого розшифровується як JavaScript Object Notation.

JSON – текстовий формат даних, який використовується практично у всіх скриптових мовах програмування, однак його витоки знаходяться у JavaScript. Він має схожість із літерним синтаксисом даної мови програмування, але може використовуватися окремо від неї. Багато середовищ розробки добре справляються з його читанням та генеруванням. JSON знаходиться в стані рядка, тому дозволяє передавати інформацію через мережу. Він перетворюється на об'єкт JS, щоб користувач міг прочитати ці дані. Здійснюється це методами мови програмування, але сам JSON методів немає, лише властивості.

Незважаючи на те, що ім’я формату даних пов’язане з мовою програмування JavaScript, текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам С-подібних мов, таких як C, C++, C#, Java, JavaScript, Perl, Python та багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними.

JSON заснований на двох структурах даних:

– об’єкт. Невпорядкований набір пар ключ-значення. У різних мовах ця концепція реалізована як об’єкт, запис, структура, словник, хеш, іменований список чи асоціативний масив;

– масив. Впорядкована колекція значень, перерахованих через кому. У більшості мов це реалізовано як масив, вектор, список чи послідовність.

Це – універсальні структури даних. Майже всі сучасні мови програмування підтримують їх у будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, має ґрунтуватися на цих структурах. Ці структури можуть зберігати значення як в подібних форматах, формуючи ієрархію даних, так і зберігати значення у вигляді примітивів:

– number;

– string;

– boolean;

– null [27].

До плюсів цього формату представлення даних можна віднести наступне:

– дуже простий у використанні;

– JSON досить швидкий, оскільки споживає дуже мало місця у пам'яті, що особливо підходить для великих графів об'єктів чи систем;

– бібліотека JSON має відкритий вихідний код та безкоштовна для використання;

– швидке виконання процесів серіалізації та десеріалізації;

– бібліотека JSON не вимагає будь-якої іншої бібліотеки для обробки;

– рушії браузерів містять вбудовані методи та функції для роботи з JSON форматом;

– немає жодних проблем з крос-браузерною підтримкою;

– не займає багато місця, є компактним у написанні та швидко компілюється;

– створення текстового вмісту зрозуміло людині, у реалізації, а читання із боку середовища розробки немає ніяких проблем. Читання може здійснюватися і людиною, оскільки нічого складного у поданні даних немає.

До мінусів відноситься:

– підтримується лише ASCII та UTF-кодування;

– неможливість створення власних структур чи посилань з метою перевикористання;

– немає підтримки простору імен, отже, погана розширюваність;

– неможливість залишити коментарі в коді, специфікація не дозволяє.

Отже, можна зробити висновок, що JSON набагато простіший та «легший» формат представлення даних у порівнянні з XML. Він пропонує значно простіший, але достатній функціонал для представлення ієрархії даних. Але це дає значний приріст у швидкості створення, валідації, опрацювання та парсингу файлу цього формату. Окрім цього, JSON-формат є одним з найбільш вдалих рішень для веб-платформи, завдяки його рівню крос-браузерності та вбудованих інструментів роботи в рушії браузерів та мову програмування JavaScript.

2.2.3 Формат YAML

YAML або «YAML Ain't Markup Language» це мова для зберігання інформації у форматі зрозумілій людині. Його назва розшифровується як «Ще одна мова розмітки». Однак, пізніше розшифровку змінили на «YAML не мова розмітки», щоб відрізняти його від справжніх мов розмітки. Завдяки своїм можливостям серіалізації ця мова є

гідною заміною мовами, як JSON і XML. До речі, YAML v1.2 є суворим надмножиною JSON.

Мова схожа на XML і JSON, але використовує більш мінімалістичний синтаксис при збереженні аналогічних можливостей. YAML зазвичай застосовують для створення конфігураційних файлів у програмах типу Інфраструктура як код (Iac), або для керування контейнерами у роботі DevOps.

Найчастіше за допомогою YAML створюють протоколи автоматизації, які можуть виконувати послідовності команд, записані в YAML-файлі. Це дозволяє вашій системі бути більш незалежною та чуйною без додаткової уваги розробника. YAML підтримує стандартні типи:

- int;
- float;
- boolean;
- string;
- null.

Серед передумов створення даної мови були принципи простоти, універсальності та гнучкості використання. Тому до його переваг можна віднести:

– зручність сприйняття та освоєння людиною. Визначення структур даних YAML дуже прості для розуміння, користувачам легко сприймати навіть складні структури даних, описаних цією мовою;

– незалежність мови програмування. YAML підтримує списки (масиви) та словники (асоціативні масиви, хеші) у формі, зрозумілій для різних мов;

– однозначність. YAML однозначно представляє структури серіалізованих даних, тому немає особливої потреби у коментарях та документації. Оскільки структури даних однозначні, для читання та запису YAML можна легко використовувати сценарії автоматизації;

– швидкість обробки. YAML швидко завантажується та легко обробляється в пам'яті;

– додаткові можливості. Підтримка розширюваних типів даних, якорів та посилань, мапінг типів із збереженням порядку ключів.

Але незважаючи на такий перелік сильних сторін, мова як формат представлення даних також містить і мінуси:

– програє JSON у швидкості парсингу та серіалізації;

– слабко адаптований для роботи з веб, зокрема при передачі даних на стороні front-end-у, на відміну від JSON та XML, які є перевіреними форматами передачі даних;

– велика кількість спеціальних символів, домовленості про структуру роблять підтримку YAML-а складною, парсери – складними;

– потрібно дуже ретельно слідкувати за відступами в файлі, які за специфікацією визначають ієрархію значень. Це особливо ускладняється тим, що даний формат не підтримує табуляцію для відступів, лише подвійний пробіл;

– через специфіку синткасису мови, можуть виникати потенційні проблеми з оформленням списків та конвертацією булевих значень при десеріалізації [28].

По завершенню розгляду формату представлення даних YAML, можна зробити такий висновок, що YAML був створений з метою замістити JSON формат – підтримує повністю його функціонал, бо є його надмножиною, виправив деякі недоліки свого «нащадка» та додав нові можливості, став ще більш лаконічним. Але незважаючи на це, сам синтаксис мови став складнішим, адже додалась значна кількість системних символів та їх комбінацій для забезпечення нового функціоналу, що вплинуло на його складність створення, а головне – на швидкість серіалізації та десеріалізації.

2.3 Огляд способів збереження конфігураційних даних для генерації коду ORM-моделей

Одною з головних функцій в проектованій системі є процес автоматичної генерації ORM-моделей. Під ORM-моделлю мається на увазі програмний код, який репрезентує таблицю бази даних та надає функціонал для маніпуляцій з її даними, та даними інших таблиць. Зазвичай таких моделей генерується для кожної таблиці бази даних, хоча деякі бібліотеки для створення віртуальних баз даних вміють конвертувати допоміжні таблиці зв’язків «багато-до-багатьох» у вигляді спеціального зв’язку, якщо вона не несе смислового навантаження.

Більшість мов програмування дають користувачу змогу розробити віртуальну базу даних за допомогою окремих ORM-бібліотек. При чому таких бібліотек може

бути навіть не одна для мови програмування. Моделі в рамках модулю пишуться з використанням усіх правил та обмежень мови програмування, до якої він відноситься, але кожен модуль сам вирішує яку структуру визначення моделі чи імена складних типів даних використовувати.

Для кожної ORM-бібліотеки буде створений окремий програмний сервіс для проведення генерації, а тому немає сенсу у підтримці його універсальності під усі наявні ОРМ-рішення. Це забезпечить його автономність, а сам сервіс буде значно легшим і простішим у розумінні в плані експлуатації, супроводження та підтримки.

Тому, для роботи програмних сервісів по генерації коду моделей, потрібно забезпечити їх усією необхідною інформацією, необхідною для цього. Для цього є сенс організувати єдине сховище даних, в якому буде міститись уся інформація. У якості прикладу такої інформації можна привести наступне:

- синтаксис оголошення класу чи функції-конструктору;
- ключові слова для назви властивостей та їх параметрів доступу;
- символи-роздільники, символи завершення рядку та інші спец.символи;
- розширення вихідного файлу;
- підтримувані значення типів даних та мапінг типів даних схеми до типів даних бібліотеки;
- синтаксис та мапінг для конвертації міжтабличних зв’язків;
- синтаксис виконання експорту.

Це лише базовий та дуже абстрактний набір даних, які потрібно зберігати на стороні сервісу, щоб успішно виконати генерацію файлу. В якості потенційних варіантів сховищ для збереження інформації такого типу було висунуто наступні:

- реляційна база даних;
- документоорієнтована база даних;
- конфігураційний локальний файл.

В якості вимог до майбутнього сховища можна виділити необхідність у посиланнях на інші значення чи записи, мати можливість зберігати складні ієрархії та структури даних та швидко виконувати операції зчитування значень. Окрім цього, можна зазначити те, що в окрім, як операцій зчитування, інші операції виконуватись не будуть, БД сервісу виступає виключно як довідник.

2.3.1 Реляційна база даних

Першим серед запропонованих рішень в якості сховища для системних даних сервісу генерації моделей є реляційна база даних. Даний тип баз даних є одним з найбільш розповсюджених серед програмних засобів, який можна навіть вважати класичним.

Реляційна база даних – це тип бази даних, що зберігає інформацію в електронних таблицях і здійснює пошук даних в одній таблиці на підставі визначених ключових полів іншої таблиці.

Під ключовими полями, чи просто ключами мається на увазі стовпець чи декілька стовпців, що додається до таблиці і дозволяє встановити зв’язок із записами в іншій таблиці. Існують ключі двох типів: первинні і зовнішні. Первинний ключом є одне або кілька полів (стовпців), комбінація значень яких однозначно визначає кожний запис у таблиці. Зовнішній ключ – це одне або кілька полів (стовпців) у таблиці, що містять посилання на поле або поля первинного ключа в іншій таблиці.

Зв’язок встановлюється між двома загальними полями (стовпцями) двох таблиць. Існують зв’язки з відношенням «один-до-одного», «один-до-багатьох» і «багато-до-багатьох». Відношення «один-до-багатьох» створюється в тому випадку, коли тільки одне з полів є полем первинного ключа або унікального індексу. Відношення «один-до-одного» створюється в тому випадку, коли обидва поля є ключовими або мають унікальні індекси. Відношення «багато-до-багатьох» фактично є двома відносинами «один-до-багатьох» з третьої таблицею, первинний ключ якої складається з полів зовнішнього ключа двох інших таблиць [29].

СКБД мають зумовлену схему, тобто дані зберігаються у рядках (записах) та стовпцях (атрибутах) із суворою структурою. Кожен запис зазвичай містить значення для кожного атрибуту, що призводить до чітких залежностей між різними точками даних.

Реляційні бази зазвичай масштабуються по вертикалі, що означає, що дані зберігаються одному сервері, а масштабування виконується шляхом додавання більшої кількості ресурсів (CPU, GPU і оперативної пам'яті). Однак перемикання з невеликих машин на більші може призводити до простою. Масштабування між кількома серверами (горизонтальне масштабування) може виявитися складним

завданням, оскільки для цього будуть потрібні зміни структури даних та додаткові інженерні зусилля.

Реляційні бази даних демонструють високу продуктивність при інтенсивних операціях читання/запису з невеликими та середніми наборами даних. Вони також забезпечують підвищену швидкість пошуку даних за рахунок додавання індексів у поля даних для запитів та об'єднання таблиць. Однак у разі збільшення обсягу даних і запитів користувачів продуктивність може знизитися.

Завдяки інтегрованій структурі та системі зберігання даних БД SQL не вимагають великих інженерних зусиль для забезпечення їх надійного захисту. Вони є гарним вибором для створення та підтримки складних програмних рішень, де будь-яка взаємодія має низку різних наслідків. Однією з основ SQL є відповідність вимог ACID (Атомарність, Узгодженість, Ізоляція, Довговічність). Відповідність вимогам ACID є кращим варіантом, якщо розробляється, наприклад, додаток для електронної комерції або фінансові програми, де є цілісність бази даних вирішальне значення.

Завдяки таким своїм особливостям побудови, представленням даних та їх зв’язку, реляційні бази даних мають наступні плюси використання:

– стандартизація. Стандарти SQL-NN (SQL-89, SQL-92, SQL-99 і т. д.), що мають кілька рівнів повноти реалізації, дозволяють створювати додатки, що переносяться між СКБД різних постачальників;

– теоретична основа. Формально визначає базові поняття моделі, мову опису та операції над відносинами;

– простота маніпуляцій. Однією з важливих переваг реляційного підходу є його простота та доступність для розуміння кінцевим користувачем. Єдиною інформаційною конструкцією є таблиця.

– універсальність. Інформаційне моделювання сутностей реального світу у вигляді набору пов'язаних таблиць є досить добрим підходом у більшості випадків;

– мова запитів. SQL – розвинена стандартизована декларативна мова 4-го покоління, яка дозволяє отримувати інформацію у будь-якому зручному для користувача вигляді;

– робота з даними. Реляційні бази даних дають можливість швидко виконувати операції видалення чи оновлення табличних даних.

Але така сильна акцентованість на стандартизованості, строгості, підтримці цілісності та відповідності вимогам ACID призводить до виникнення наступних слабких сторін:

– повільність. При накопиченні великої кількості даних, доступ до даних сповільнюється у порівнянні з іншими видами БД, зокрема стосується операцій зчитування та запису;

– суворість структури. Даний тип БД не підтримує або слабко підтримує неструктуровані ієрархії даних, наприклад як JSON-об’єкти;

– складність проектування. Не завжди предметну область можна у вигляді сукупності таблиць. Так, у системах автоматизації проектування та автоматизованої розробки програмного забезпечення потрібні набагато складніші структури даних.

У зв’язку з цим можна зробити висновок, що реляційні бази даних це ідеальний варіант у якості основної бази даних на проекті. SQL підійде, якщо потрібна обробка великої кількості складних запитів або рутинного аналізу даних. Особливо якщо велика частина запитів буде націлена на оновлення чи зміну даних. Але якщо база даних потрібна у якості довідника, то РБД не найкращий вибір через свою швидкість роботи зі зчитуванням та записом даних. Так само реляційна БД не підійде, якщо потрібно зберігати різнорідну інформацію, адже таблична репрезентація не націлена на такий формат.

2.3.2 Документоорієнтована база даних

Нереляційна база даних або NoSQL БД – це будь-яка база даних, яка замість табличного представлення використовує різні моделі даних для зберігання та управління. Тут, на відміну більшості традиційних СКБД, не використовується таблична схема рядків і стовпців, а застосовується модель, оптимізована під конкретні вимоги типу даних, що зберігаються.

Нереляційні БД відомі своєю високою продуктивністю, забезпечуючи одночасний доступ до великої кількості користувачів. Можуть зберігати необмежену кількість наборів всіх типів і форм. Вони також досить гнучкі, коли справа доходить до зміни типів даних. Крім того, створення MVP - це чудовий варіант для стартапів з

гнучкою розробкою на основі спринту. NoSQL не вимагає попередньої підготовки до розгортання, що полегшує швидке оновлення структури даних без затримок за часом.

Найбільш поширені типи нереляційних БД:

– типу «ключ-значення»;

– документоорієнтовані бази даних;

– графові бази даних;

– мережеві бази даних.

Через появу СКБД MongoDB, документоорієнтовані бази даних зайняли своє місце на ринку та стабільно утримують його. Документоорієнтована БД є системою зберігання ієрархічних структур даних (документів), що має структуру дерева або лісу. Структура дерева починається з кореневого вузла і може мати декілька внутрішніх та листових вузлів. Листові вузли містять кінцеві дані, які при додаванні заносяться в індекси бази, завдяки яким можна здійснювати швидкий пошук навіть за досить складної загальної структури сховища. Фактично документоорієнтовані БД є складнішою версією сховищ «ключ-значення» - вони досі хороші для систем, які мають безліч зв'язків між елементами, але дозволяють здійснювати вибірку на запит без повного завантаження окремих документів на оперативну пам'ять. Механізми пошуку дозволяють знаходити документи як цілком, так і частини документів, а деревоподібна структура дозволяє організовувати окремі колекції документів одного типу або схожої тематики.

Завдяки неструктурованому, не класичному підходу, документоорієнтовані БД мають такі сильні сторони:

– можливість збереження великих обсягів неструктурованої інформації. У NoSQL немає обмежень на типи даних, що зберігаються, а при необхідності можна додавати нові типи даних;

– NoSQL-бази краще піддаються масштабуванню. Хоча масштабування підтримується і в SQL-базах, це потребує значно більших витрат людських та апаратних ресурсів;

– швидка технологія. NoSQL бази даних не вимагають великого обсягу підготовчих дій, що потрібні для реляційних баз;

– багато NoSQL рішення мають обмежену функціональність, тому що вирішують певні завдання. Тому для роботи з такими базами даних не потрібні

глибокі знання SQL-запитів. Це дуже знижує вхідний поріг для початку роботи з NoSQL сховищами;

– простота роботи. Багато NoSQL рішень, в основному сховища виду «ключ-значення», мають у порівнянні з реляційними базами даних дуже сильно урізану функціональність, яка їм просто не потрібна для виконання поставлених завдань. Це дуже сильно знижує вхідний поріг для початку роботи з NoSQL сховищами;

– власні мови запитів сучасних NoSQL сховищ набагато більше підходять для виконання простих маніпуляцій із базою даних.

Окрім переваг, документоорієнтовані рішення мають і низку недоліків, а саме:

– програма сильно прив'язується до конкретної СКБД;

– обмежена ємність вбудованої мови запитів. SQL дуже потужний і складний інструмент для операцій з даними та складання звітів. Практично всі мови запитів та методи API сховищ NoSQL були створені на основі тих чи інших функцій SQL, але вони мають значно меншу функціональність;

– процес створення реляційного сховища включає етап проектування моделі даних. Для NoSQL рішень необов’язково визначати схему бази даних перед початком її використання, тому в процесі розробки можна натрапити на непередбачені труднощі з її структурою [30].

Тому можна зробити висновок, що документоорієнтовані бази даних не є найкращим рішенням для використання в якості основної бази даних програмного забезпечення через обмеженість функціоналу мови запитів. А довільна структура даних може позначитись на зручності роботи зі сховищем. Але це є прекрасним вибором для тих частин проекту, де метою є збереження даних з динамічною чи з різнорідною структурою та можливістю виконувати прості маніпуляції з ними.

2.3.3 Конфігураційний локальний файл

Останнім способом для збереження даних для проведення генерації є конфігураційний локальний файл. Як вже було згадано вище такі файли призначені для збереження параметрів налаштування, ініціалізації деяких програмних засобів чи для представлення важливої інформації у структурованому та зручному для аналізу

вигляді. Локальним він вважається тому що на відміну від розгорнутих серверів баз даних зберігається локально в репозиторії проекту.

Проведемо огляд сильних та слабких сторін використання конфігураційного файлу у якості сховища даних та оцінимо чи підходить він для поставленої задачі. До сильних сторін можна віднести наступне:

– швидкість звернення. Через те, що файл зберігається локально, то швидкість звернення до нього займає мінімальну кількість часу;

– швидкість розгортання. На відміну від баз даних, для конфігураційного файлу не потрібно розгортати окремі сервери чи сервіси;

– збереження ресурсів. Для такого файлу не потрібно виділяти окремі апаратні ресурси для його розгортання;

– простота використання. Для отримання значень з файлу не потрібно використовувати жодну мову запитів, усі значення зазвичай отримуються по ключу.

До слабких сторін можна віднести:

– слабка підтримка посилань та зв’язків між даними. У конфігураційному файлі складно реалізувати посилання між значеннями, чи перевикористовувати існуючі записи;

– відсутність можливості робити запити. Конфігураційний файл не підтримує будь-який запитів чи операцій, окрім читання. На відміну від БД, при читанні неможливо передавати фільтри чи інші модифікатори;

– потреба у парсингу. Для того, щоб почати роботу з конфігураційним файлом, його потрібно перетворити у читабельний для мови програмування вигляд, що буде забирати певний час.

Отже, можна підвести підсумок, що формат конфігураційного файлу забезпечує гарну швидкість звернення для отримання даних через його локальне розміщення, а також економить час та ресурси на його розгортання. Але в той же час, при кожному зверненні до конфігураційного файлу його потрібно парсити, щоб перетворити в структури даних мови програмування, що використовується. А сам процес отримання даних може бути ускладнений через відсутність відповідних інструментів, на кшталт мови запитів.

3 ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

3.1 Мета та призначення системи

Розроблювана веб-платформа повинна забезпечити IT-фахівців єдиною цілісною платформою з можливістю проведення моделювання схеми даних для усіх сучасних реляційних баз даних та подальшою автоматизованою генерацією коду для заздалегідь обраної ORM-бібліотеки. Завдяки авторизації в системі, користувач повинен мати виконувати збереження та експорт результатів проектування. Процес підготовки схем даних повинен надати зручний та інтуїтивно-зрозумілий функціонал для користувача, а процес генерації коду – забезпечити якість коду та швидкість його створення.

3.2 Функціональні вимоги до системи

Основні функціональні вимоги зі сторони користувача:

а) можливість реєстрації з заповненням форми;

б) можливість авторизації на веб-платформі

1) авторизація з використанням даних реєстрації;

2) авторизація з використанням сторонніх сервісів «OAuth2» (Google, Apple, Github).

в) спроектувати схему даних на логічному рівні з використанням графічних форм створення діаграм;

г) обрати одну з підтримуваних реляційних СКБД та спроектувати схему даних на фізичному рівні з використанням графічних форм створення діаграм;

д) провести генерацію коду для ORM-моделей

1) обрати ORM-бібліотеку зі списку підтримуваних;

2) налаштувати параметри для виконання генерації файлів: шаблон назви файлу моделі, розширення (якщо дозволяє мова та бібліотека), тип архівування файлів тощо;

3) підтвердити налаштування та запустити процес генерації;

4) завантажити результат генерації файлів у вигляді архіву;

е) можливість виконання маніпуляцій з проектами (лише для авторизованих користувачів). Під проектом мається на увазі віртуальна сутність, яка логічно об’єднує в собі усі результати роботи користувача.

1) переглянути усі створені проекти користувачем, дату їх створення, створені моделі даних, історія генерації ORM-моделей тощо;

2) створити проект для збереження результатів роботи: результати логічного та фізичного проектування рівня даних, фіксація результатів генерації;

3) створити копію існуючого проекту з усіма результатами роботи;

4) видалити проект із раніше створених користувачем.

ж) Експорт результатів моделювання схеми даних

1) експорт схем даних у вигляді зображення форматів jpeg чи png;

2) експорт схем даних у вигляді конфігураційного файлу.

Основні функціональні вимоги зі сторони системи:

– повернути результати генерації на основний сервіс та автоматично розпочати завантаження архіву згідно з налаштуваннями, визначеними користувачем перед запуском процесу генерації;

– виконати генерацію конфігураційного файлу з текстовим представленням схеми даних, створеної користувачем;

– визначити сервіс, який повинен виконувати генерацію файлів моделей та направити йому конфігураційний файл з описом схеми даних;

– виконати генерацію моделей обраної ORM-бібліотеки згідно з налаштуваннями, визначеними користувачем перед запуском процесу генерації;

– зберігати результати роботи користувачів до бази даних та до файлових сховищ;

– забезпечити процес автоматичного конвертування логічної моделі даних в фізичну та зворотній процес;

– надавати повноцінну інформацію про усі ключові елементи, структури, типи даних відповідно до обраної РБД;

– забезпечити необхідні графічні елементи та поля для їх налаштування з метою зручного моделювання схеми даних.

3.3 Нефункціональні вимоги до системи

Атрибути якості:

а) логічна структура серверів системи повинна підтримувати load-balancing з метою оптимізації використання ресурсів;

б) веб-застосунок системи повинен безперебійно та коректно працювати на наступних браузерах:

1) Microsoft Edge 12>;
2) Firefox 28>;
3) Chrome 29>;
4) Safari 14.1>;
5) Opera 17>.

в) веб-застосунок повинен підтримувати локалізацію для заздалегідь визначених мов інтерфейсу;

г) структура застосунку повинна підтримувати можливість масштабування (як горизонтального, так і вертикального).

Обмеження:

– система повинна підтримувати коректне відображення інтерфейсу для екранів з роздільною здатністю в межах від 1280×1024 до 2560×1440;

– швидкість інтернет-підключення (Мбіт/с) клієнту повинна бути не менше 5 Мбіт/с для забезпечення комфорту роботи;

– оперативна пам’ять клієнту повинна бути не менше 2 Гб для забезпечення комфорту роботи;

– сервер додатку має бути розгорнутий на Linux-подібній ОС.

Вимоги до інтерфейсів:

– дані між клієнтом та сервером повинні передаватись у форматі JSON;

– для виконання запитів повинен використовуватись лише RESTful підхід;

Вимоги до даних:

– внутрішні дані системи повинні зберігатись в реляційній базі даних PostgreSQL версії 14+;

– для збереження мультимедіа та звітів необхідно використовувати окреме об’єктне сховище Amazon S3;

– вести журнал передзаписів для забезпечення цілісності даних (WAL).

3.4 Вихідна інформація

Під час використання системи користувач має можливість отримати наступну вихідну інформацію:

– схема даних логічного рівня представлення у вигляді зображення (формати .jpeg, .png);

– схема даних фізичного рівня представлення у вигляді зображення (формати .jpeg, .png);

– текстове представлення схеми даних комбінованого представлення у вигляді конфігураційного файлу формату JSON;

– архів зі згенерованими файлами ORM-моделей у форматі, що підтримується відповідною ORM-бібліотекою. Формати архіву – .rar, .zip.

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ВИРІШЕННЯ ЗАДАЧІ

4.1 Опис методу генерації конфігураційного файлу

Першим етапом опису методу генерації конфігураційного файлу, що буде містити текстове представлення схеми даних, яка була розроблена користувачем є визначення формату представлення конфігураційного файлу. На етапі огляду методів та технологій в якості кандидатів для представлення текстових даних було обрано формати XML, JSON та YAML. Кожен з них мав свої переваги та недоліки, особливості використання та особливості синтаксису. В результаті, серед цих варіантів було обрано формат JSON. Причина вибору пояснюється тим, що це зручний, простий для освоєння та використання формат, який забезпечує високу швидкість генерації, парсингу та передачі. Цей формат прекрасно працює з веб-системами, а його інтегрованість в більшість мов програмування робить реалізацію підходу генерації простою.

Наступним етапом опису є визначення вихідної структури конфігураційного файлу. Ця структура повинна бути єдиною та незмінною для будь-яких РБД, що були використані на етапі моделювання. Така уніфікованість забезпечить швидкість розробки та використання даних з конфігураційного файлу, адже не буде необхідності створювати різні алгоритми парсингу під різні бази даних, а подібна структура більшості реляційних баз даних тільки сприятиме цьому. Загальну структуру конфігураційного файлу наведено в лістингу 4.1.

Лістинг 4.1 – Схема конфігураційного файлу

```
{
  "metadata": {
    "createdAt": "09-12-2022T22:08:37Z",
    "userId": "USER_ID",
    "rdb": {
      "version": "14.5.1",
      "type": "postgresql"
    },
    "orm": {
      "version": "6.0",
```

```
    "type": "sequelize"
  },
  "pl": {
    "type": "nodejs",
    "version": "18.12.1",
    "ext": ".js"
  }
},
"schemaData": {
  "tables": [{
    "_id": "TABLE_ID",
    "tableName": "user",
    "modelName": {
      "singularName": "user",
      "pluralName": "users"
    },
    "fields": [{
      "_id": "FIELD_ID",
      "domainType": "string ",
      "dataType": "varchar",
      "name": "email",
      "isPrimaryKey": true,
      "isNull": false,
      "isAutoIncrement": true,
      "defaultValue": "kyrylo.dolhopolov@nure.ua"
    }],
    "constraints": [{
      "_id": "CONSTRAINT_ID",
      "type": "unique",
      "fields": ["FIELD_ID_1", "FIELD_ID_2"]
    }]
  }],
  "relations": [{
    "_id": "RELATION_ID",
    "type": "ONE_TO_MANY",
    "sourceTableId": "TABLE_ID",
    "targetTableId": "TABLE_ID",
    "onDelete": "cascade",
    "onUpdate": "restrict"
```

```
    }]
  },
  "settings": {
    "templates": {
      "pkConstraint": "pk_{TABLE.NAME}_{FIELDS.NAME}",
      "fkConstraint": "fk_{SOURCE_TABLE.NAME}_{TARGET_TABLE.NAME}",
      "uniqueConstraint": "uq_{TABLE.NAME}_{FIELDS.NAME}",
      "indexConstraint": "idx_{TABLE.NAME}_{FIELDS.NAME}",
      "file": "{MODEL.NAME}",
      "archive": "generated_models"
    },
    "indetation": {
      "type": "space",
      "count": 2
    },
    "export": "named",
    "archive": "rar"
  }
}
```

При аналізі структури файлу можна виділити три основні компоненти:

– «metadata»;

– «schemaData»;

– «settings».

Як і у більшості випадків, так і тут, об’єкт «metadata» містить інформацію, яка використовується для опису даних, які містяться в щось на зразок веб-сторінки, документа або файлу. Інше представлення поняття метаданих – це коротке пояснення або підсумок того, що є даними. В конфігураційному файлі, компонента «metadata» зберігає ключову інформацію про сам конфігураційний файл, користувача системи, який його згенерував та ключову інформацію про обрані технології – ORM-бібліотеку, реляційну базу даних та мову програмування. Детальний опис ключів компоненти наведено в таблиці 4.1.

Таблиця 4.1 – Опис структури компоненти «metadata»

Ключ	Належить ключу	Опис значення ключа
createdAt	metadata	Під цим ключем зберігається значення дати та часу, коли користувач згенерував цей конфігураційний файл. Значення текстового типу, що зберігає часову мітку в форматі «YYYY-MM-DDThh:mm:ssZ», затвердженому специфікацією ISO 8601.
userId	metadata	Під цим ключем зберігається значення ідентифікатору користувача, що зберігається в системній базі даних. Значення поля текстового типу та має формат «uuid» версії 4, затвердженому в специфікації RFC 4122.
rdb	metadata	Об’єкт містить інформацію про обрану користувачем реляційну базу даних на етапі моделювання схеми даних фізичного рівня.
version	rdb	Ключ містить значення версії РБД, на основі якої виконувалось моделювання. Поле зберігає значення текстового типу вільного формату.
type	rdb	Ключ «type» містить константне значення-ідентифікатор однієї з підтримуваних системою РБД, яка була застосована для цієї схеми. Поле зберігає значення текстового типу вільного формату.
orm	metadata	Об’єкт містить інформацію про обрану користувачем ORM-бібліотеку для якої необхідно згенерувати файли моделей.
version	orm	Ключ містить значення версії ORM-бібліотеки, для якої планується виконати генерацію моделей. Поле зберігає значення текстового типу вільного формату.
type	orm	Ключ «type» містить константне значення-ідентифікатор ORM-бібліотеки, для якої планується виконати генерацію моделей. Поле зберігає значення текстового типу вільного формату.
pl	metadata	Об’єкт містить інформацію про мову програмування, яка буде використовуватись при генерації файлів моделей.
type	pl	Ключ містить значення версії мови програмування, яка використовуватиметься при генерації моделей. Поле зберігає значення текстового типу вільного формату.
version	pl	Ключ містить значення версії ORM-бібліотеки, яка використовуватиметься при генерації моделей. Поле зберігає значення текстового типу вільного формату.

Кінець таблиці 4.1

Ключ	Належить ключу	Опис значення ключа
ext	pl	Ключ містить значення розширення для файлів обраної мови програмування. Поле зберігає значення текстового типу вільного формату.

Наступною компонентою, що записується до конфігураційного файлу є об’єкт «schemaData». Це є однією з найважливіших частин файлу, адже саме тут зберігається вся інформація про схему даних. Структура сформована таким чином, щоб якомога повно та універсально зберігати інформацію про схему. Зокрема це стосується таблиць, їх полів, зв’язків між ними та обмежень. Детальний опис ключів компоненти наведено в таблиці 4.2.

Таблиця 4.2 – Опис структури компоненти «schemaData»

Ключ	Належить ключу	Опис
tables	schemaData	Поле зберігає масив об’єктів, кожен об’єкт описує таблицю схеми даних. Кількість об’єктів в масиві дорівнює кількості таблиць на схемі. Опис містить інформацію про саму таблицю, її поля та обмеження.
_id	tables	Поле містить значення унікальний ідентифікатор об’єкту таблиці, по якому її можна визначити чи посилатись на неї. Значення поля текстового типу та має формат «uuid» версії 4, затвердженому в специфікації RFC 4122.
tableName	tables	Поле містить назву таблиці безпосередньо в СКБД. Саме цю назву використовує ORM-бібліотека при побудові SQL-запитів для звернення до таблиці. Значення поля текстового типу вільного формату.
modelName	tables	Поле зберігає об’єкт в якому описуються значення назви моделі, яка буде створена для відповідної таблиці.
singularName	modelName	Поле містить назву моделі в однині. Значення поля текстового типу вільного формату.

Продовження таблиці 4.2

Ключ	Належить ключу	Опис
pluralName	modelName	Поле містить назву моделі в множині. Такий формат назви моделей необхідний для деяких ORM, а деякі підтримують множинну назву як опцію. Значення поля текстового типу вільного формату.
fields	tables	Цей ключ містить масив об’єктів, кожен об’єкт описує поле таблиці. Кількість об’єктів в масиві дорівнює кількості полів в таблиці. Опис містить інформацію про назву поля, його тип та обмеження.
_id	fields	Поле містить значення унікальний ідентифікатор об’єкту поля, по якому її можна визначити чи посилатись на нього. Значення поля текстового типу та має формат «uuid» версії 4, затвердженому в специфікації RFC 4122.
domainType	fields	Під цим ключем зберігається значення типу домену даних, яке підтримує це поле в моделі. Зберігається значення текстового типу та повинне належати до одного з допустимих значень: «INTEGER», «DECIMAL», «STRING», «DATETIME», «JSON».
dataType	fields	Під цим ключем зберігається значення типу даних, яке підтримує це поле в таблиці БД. Зберігається значення текстового типу та повинне належати до одного з допустимих значень типів даних, яке підтримує обрана СКБД на етапі проектування схеми.
tableName	fields	Під цим ключем зберігається назва поля в таблиці БД. Саме цю назву використовує ORM-бібліотека при побудові SQL-запитів для звернення до поля таблиці. Зберігається значення текстового типу вільного формату.
modelName	fields	Ключ зберігається назва поля в моделі ORM. Зберігається значення текстового типу вільного формату.
isPrimary	fields	Ключ зберігає значення чи є поле таблиці первинним ключем. Зберігається значення булевого типу. Відразу декілька полів з позитивним значенням формують складений первинний ключ.
isNull	fields	Ключ зберігає значення чи може поле таблиці містити пусте значення NULL. Зберігається значення булевого типу.
isAutoIncrement	fields	Ключ зберігає значення чи може повинне поле таблиці автоматично збільшувати значення поля при додаванні нового запису. Зберігається значення булевого типу.

Продовження таблиці 4.2

Ключ	Належить ключу	Опис
defaultValue	fields	Ключ зберігає значення за замовчуванням для поля таблиці, яке буде зберігатись в екземпляр моделі та в таблицю БД, якщо не було надано конкретного. Значення повинне бути того ж типу даних, що й поле в таблиці.
constratints	tables	Ключ містить масив об’єктів, кожен об’єкт описує обмеження та індекси, які можуть додаватись на поле чи декілька полів. Кількість об’єктів в масиві дорівнює кількості обмежень та індексів на таблиці, окрім первинного ключа.
_id	constratints	Ключ містить значення унікального ідентифікатору об’єкту обмеження, по якому її можна визначити чи посилатись на нього. Значення поля текстового типу та має формат «uuid» версії 4, затвердженому в специфікації RFC 4122.
type	constratints	Ключ містить одне з константних значень типу обмеження, що застосовується до обраних полів таблиці. Зберігається значення текстового типу та повинне належати до одного з допустимих значень: «UNIQUE», «INDEX».
fields	constratints	Ключ містить перелік полів, до яких застосовується обмеження чи індекс. Зберігається значення у вигляді масиву унікальних ідентифікаторів полів.
relations	schemaData	Ключ містить масив об’єктів, кожен об’єкт описує параметри зв’язку між таблицями. Кількість об’єктів в масиві дорівнює кількості зв’язків між таблицями на схемі.
_id	relations	Ключ містить значення унікального ідентифікатору об’єкту зв’язку, по якому його можна визначити чи посилатись на нього. Значення поля текстового типу та має формат «uuid» версії 4, затвердженому в специфікації RFC 4122.
type	relations	Ключ містить одне з константних значень типу зв’язку. Зберігається значення текстового типу та повинне належати до одного з допустимих значень: «ONE_TO_MANY», «ONE_TO_ONE», «MANY_TO_MANY».
sourceTableId	relations	Ключ містить ідентифікатор таблиці, яка є таблицею-батьком в зв’язку. Значення поля текстового типу та повинне відповідати одному з ідентифікаторів поля відповідної таблиці.

Кінець таблиці 4.2

Ключ	Належить ключу	Опис
targetTableId	relations	Ключ містить ідентифікатор таблиці, яка є таблицею-нащадком в зв'язку. Значення поля текстового типу та повинне відповідати одному з ідентифікаторів поля відповідної таблиці.
onDelete	relations	Ключ містить одне з константних значень, яке описує поведінку таблиці при видалені з неї записів. Значення поля текстового типу та повинне належати до одного з допустимих значень: «RESTRICT», «CASCADE», «SET_NULL», «SET_DEFAULT».
onUpdate	relations	Ключ містить одне з константних значень, яке описує поведінку таблиці при видалені з неї записів. Значення поля текстового типу та повинне належати до одного з допустимих значень: «RESTRICT», «CASCADE», «SET_NULL», «SET_DEFAULT».

Останньою компонентою, яка формує конфігураційний файл схеми бази даних є компонента «settings». Вона характеризується тим, що містить ключову інформацію, яка необхідна для безпосереднього процесу генерації моделей. Це можуть бути налаштування визначення імен в моделі чи параметри мови програмування. Головний плюс компоненти в тому, що вона може вільно розширюватись та адаптуватись під нові сервіси ORM-бібліотек без боязку зламати щось для вже існуючих. Дані для нього формуються на перед початком процесу генерації – користувачу відкривається окремий інтерфейс налаштувань, де він може встановити необхідні значення параметрів. Детальний опис ключів компоненти наведено в таблиці 4.3.

Таблиця 4.3 – Опис структури компоненти «settings»

Ключ	Належить ключу	Опис
templates	settings	Ключ зберігає об'єкт, де описано шаблони для формування імен.
pkConstraint	templates	Ключ містить шаблон для формування назви обмеження первинного ключа. Значення поля текстового типу, яке може містити допустимі системні змінні.
fkConstraint	templates	Ключ містить шаблон для формування назви обмеження зовнішнього ключа. Значення поля текстового типу, яке може містити допустимі системні змінні.

Кінець таблиці 4.3

Ключ	Належить ключу	Опис
uniqueConstraint	templates	Ключ містить шаблон для формування назви обмеження унікальності. Значення поля текстового типу, яке може містити допустимі системні змінні.
indexConstraint	templates	Ключ містить шаблон для формування назви індексу. Значення поля текстового типу, яке може містити допустимі системні змінні.
file	templates	Ключ містить шаблон для формування імені обмеження первинного ключа. Значення поля текстового типу, яке може містити допустимі системні змінні.
indetation	settings	Ключ зберігає об’єкт, де описано яким чином виконувати відступи рядків коду у файлі.
type	indetation	Ключ зберігає одне з константних значень, яке визначає символ для виконання відступів. Зберігається значення текстового типу та повинне належати до одного з допустимих значень: «SPACE», «TAB».
count	indetation	Ключ містить значення кількості символів, яке потрібно зробити для відступу. Зберігається значення числового типу, цілого невід’ємного формату.
export	settings	Ключ зберігає одне з константних значень, яке визначає підхід експорту функції-конструктора чи класу моделі в файлі. Зберігається значення текстового типу та повинне належати до одного з допустимих значень: «NAMED», «DEFAULT».

Уся інформація для створення конфігураційного файлу збирається з двох місць системи: з бази даних системи та з форми ініціалізації процесу генерації ORM-моделей. Сам процес формування конфігураційного файлу виконується на основному сервісі системи. Для вище визначеної схеми конфігураційного файлу підготовлено схему валідації. Ця схема валідації використовується по завершенню процесу формування конфігураційного файлу для перевірки його цілісності та правильності. Валідація включає в себе етапи:

- перевірка наявності обов’язкових полів;
- перевірка збереження допустимих значень та типів даних;

– перевірка значень взаємозалежних полів.

В результаті неуспішної валідації, користувача попередять повідомленням на екрані візуально та опишуть причину помилки. Водночас, помилка валідації конфігураційного файлу буде автоматично записана системою разом з самим файлом, щоб адміністратор міг провести аналіз та визначити причину. Якщо ж валідація проведена успішно, то користувача також про це повідомлять та дадуть можливість завантажити конфігураційний файл схеми, а процес генерації моделей автоматично розпочне своє виконання.

4.2 Опис методу генерації файлів моделей

Генерація коду для ORM-моделей складний і заплутаний процес, в якому закладено багато залежностей, потрібно дотримання багатьох передумов та виконання необхідних задач. І здається, що різноманітність ORM-бібліотек для мов програмування повинна тільки ускладнити підготовку алгоритмів для генерації файлів. Звичайно, що різні мови програмування та бібліотеки обумовлюють відмінності хоча б в плані ключових програмних слів, підтримуваних структур даних чи синтаксису. Проте, як і реляційні бази даних, ORM-бібліотеки дуже подібні між собою за принципом дії та дуже схожі по структурі моделі. Тому і алгоритми для побудови моделей будуть дуже подібні за принципом дії, а сховища з конфігураційними даними подібні за своєю схемою. Ця схожість дає змогу узагальнити та уніфікувати етапи процесу генерації коду для всіх бібліотек, в той час як усі технічні відмінності та особливості будуть інкапсульовані в рамках самого сервісу відповідної ORM-бібліотеки чи даних його сховища.

В процесі генерації файлів моделей можна виділити наступні послідовні етапи:

а) отримання конфігураційного файлу схеми бази даних;

б) підготовка папки для збереження файлів моделей;

в) створення файлів для моделей;

г) генерація коду файлу моделі;

1) вставка фрагменту коду з обов’язковими імпортами;

2) вставка фрагменту коду з каркасом класу моделі;

– генерація коду налаштувань моделі;

– генерація коду полів моделі;

– генерація коду обмежень та індексів моделі;

– генерація коду відношень між моделями.

3) вставка фрагменту коду з експортом класу моделі;

д) виконання архівації папки з моделями;

е) повернення результату роботи сервісу генерації [31].

Перед початком опису етапів генерації файлу, необхідно визначитись зі сховищем конфігураційних даних сервісу, зокрема обрати спосіб зберігання та загальну структуру цього сховища. Під конфігураційними даними сервісу мається на увазі шаблони коду, перелік ключових програмних слів та списки співвідношень структур бібліотеки та інші елементи, які необхідні для безпосереднього конструювання коду моделі. Для розробки схеми сховища під цю інформацію, спочатку потрібно обрати один зі способів збереження. В якості варіантів було розглянуто реляційну базу даних, документоорієнтовану базу даних та локальний конфігураційних файл. Серед цих варіантів було надано перевагу документоорієнтованій базі даних, на кшталт MongoDB. Сховища такого типу дозволяють створювати колекції документів будь-якої структури з можливістю посилатись на інші об’єкти. Така гнучкість дасть перевагу у збереженні складних програмних параметрів. Швидкість операцій зчитування та зручність побудови запитів роблять цей базу даних прекрасним інструментом для пошуку та отримання потрібної інформації на етапах генерації коду.

Оглянемо структуру конфігураційної бази даних, її колекції та їх призначення більш детально. Графічне представлення структури бази даних наведено на рисунку 4.1. Всього база даних містить 11 колекцій даних:

– «OrmSettingsTypes». Колекція зберігає всі характеристики та налаштування, які можуть вплинути на код чи його структуру під час генерації. Це можуть бути наприклад версія ORM-бібліотеки, особливості мови програмування, тип представлення моделі тощо;

– «OrmSettingsValues». Колекція містить всі можливі значення налаштувань, визначених в колекції «OrmSettingsTypes»;

– «OrmDataTypes». Колекція зберігає всі можливі значення типів даних, які підтримує бібліотека. Окрім самих типів, вона зберігає і фрагмент коду, який потрібно вставити в файл для відповідного типу даних;

– «DataTypesMapping». Колекція слугує для вираховування типу даних в ORM-бібліотеці для поля на основі типу домену та типу даних поля, отриманих зі схеми бази даних відповідного атрибуту сутності. Детально процес отримання типу даних бібліотеки буде описано пізніше;

– «Constraints». Колекція містить інформацію про обмеження, які повинні дотримуватись системою та параметрами генерації, щоб отримати фрагмент коду, який валідний та працездатний для вказаних параметрів. Саме тут використовуються посилання на характеристики та їх значення з документів колекцій «OrmSettingsTypes» та «OrmSettingsValues» відповідно;

– «RelationTypeMapping». Колекція містить співвідношення між типами відносин, що зберігаються в згенерованій схемі бази даних та підтримуваними бібліотекою. Водночас тут застосовується посилання обмеження, адже колекція зберігає і фрагмент коду, що відповідає відповідному типу зв’язку;

– «TemplateTypes». Колекція містить перелік типів шаблонів коду, які застосовуються при генерації моделі. Це зокрема «import», «export», «class», «classProperty», «field», «index», «constraint», «relation»;

– «Markers». Колекція зберігає значення маркерів коду, які застосовуються у відповідних шаблонах коду. Маркери слугують для полегшення вставки згенерованого коду до шаблону;

– «Keywords». Колекція слугує для збереження ключових слів, які можуть застосовуватись в шаблоні коду. Ключові слова представляють собою відповідні місця вставки назв моделей, обмежень, значень параметрів, типів даних тощо;

– «Properties». Колекція містить програмні назви, які використовуються для зазначення параметрів полів. Наприклад назви параметрів, що відповідають за тип даних, автоінкремент, первинний ключ тощо;

– «Templates». Одна з найважливіших колекцій бази даних, адже зберігає документи з шаблонами коду для тих частин файлу моделі, які зазначені в колекції «TemplateTypes». Кожен шаблон також характеризується своїми обмеженнями використання.

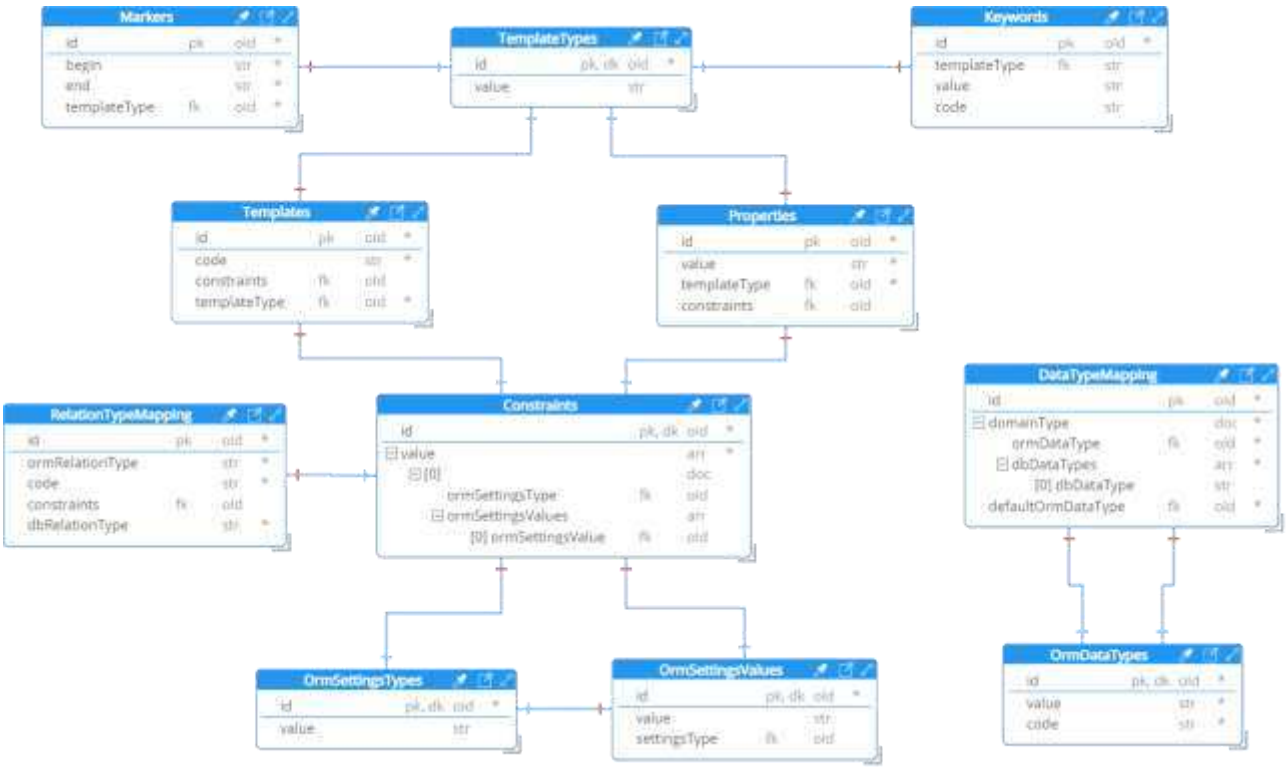

Рисунок 4.1 – Схема конфігураційної бази даних сервісу

Перейдемо до безпосереднього опису етапів генерації файлів моделей. Першим етапом є отримання конфігураційного файлу схеми бази даних. Файл формується на стороні основного сервісу системи. Там же визначається який з окремих сервісів генерації моделей повинен отримати цей файл та ініціювати початок роботи. Вибір кінцевого сервісу виконується на основі обраної ORM-бібліотеки. По завершенню процесу створення файлу схеми він надсилається на визначений сервіс генерації моделей. В якості методу комунікації між компонентами системи та зокрема передачі схеми бази даних використовується REST-підхід передачі даних – один з найпопулярніших підходів для проектування API сервісів [32]. Це не протокол як HTTP і не стандарт як SOAP, це архітектурний стиль, який вводить п’ять обмежень або принципів для комунікації в розподіленої архітектури:

– наявність клієнта і сервера;
– принцип «stateless»;
– єдині інтерфейси та ідентифікатори для ресурсів;
– архітектура розділена на шари;
– кешування відповіді сервера.

Відповідно, якщо створені сервіси в веб-застосунку відповідають цим обмеженням, то вони мають усі підстави називатись RESTful-сервісами. Будь-який протокол, формат даних або фреймворк може бути-використаний для розробки своїх сервісів. Але традиційно саме протокол HTTP і формат JSON найбільш часто використовуються завдяки своїй простоті та популярності в IT. Тоді сервіси матимуть назву RESTful вебсервіси.

Обмеження та принципи носять лише рекомендаційний характер і сам підхід не забороняє користувачу його реалізовувати не дотримуючись загальноприйнятих IT-громадою правил. Втім це може сильно позначитися на зручності використання та подальшій підтримці системи з некласичною реалізацією запитів REST-підходу. Цей підхід підтримує передачу параметрів та даних у одному з можливих типів, а саме:

– «path» параметри. Зазвичай параметри такого типу використовуються для ідентифікації ресурсу. Це потрібно для отримання інформації про ресурс, зміни його даних чи його видалення в цілому. Параметри «шляху» вказуються безпосередньо в URL-рядку запиту;

– «query» параметри. Параметри цього типу вже несуть більш складний характер, ніж просто ідентифікація ресурсу. Зазвичай використовуються для розширення параметрів пошуку, фільтрації, пагінації та інших характеристик для отримання ресурсу. Як і попередній тип, цей вказується безпосередньо в URL;

– «body» параметри. Найбільш потужний тип серед всіх трьох, адже окрім текстової інформації, він дає змогу передавати інформацію в будь-якому форматі, в тому числі і файловому.

Для передачі схеми бази даних підходить лише параметри типу «body». Це пов’язано з двома причинами. Перша полягає в тому, що ні «path», ні «query» параметри не здатні надіслати файл в якості параметру. Якщо ж передавати схему БД у вигляді серіалізованого JSON-у, то виникає причина №2 – максимальний об’єм інформації, який можна передати за допомогою цих типів. Так як ці параметри записуються та зберігаються в URL-рядку, то на них розповсюджується значення максимальної довжини рядку, який підтримується конкретним браузером. Зазвичай це обмеження становить приблизно 2048 символів, тобто 2 кілобайти [33]. Таких обмежень буде недостатньо навіть для передачі схеми даних, що містить опис не більше двох-трьох таблиць. Так, можливо окремі браузери підтримують передачу і

більш довших URL-рядків, але потрібно розуміти, що це не надійний, не швидкий, не зручний і не назначений для цього спосіб передачі значних об’ємів інформації.

Тому залишається лише тип параметрів «body». На відміну від попередніх двох, для цього типу параметрів значення максимального об’єму інформації значно вище. В залежності від веб-браузеру і HTTP-клієнту, що використовується максимальний об’єм інформації може розширюватися аж до 2 гігабайт [34].

Окрім цього, «body»-параметри можуть передавати як текстові дані, так і файл. Через те, що схема бази даних зберігається у форматі JSON, для її передачі може використовуватись обидва підходи, але текстовий формат передачі схеми даних має відразу декілька переваг над передачею у вигляді конфігураційного файлу:

– відсутність необхідності перетворювати файл в бінарний буфер даних та прикріпляти до запиту;

– відсутність необхідності використовувати додаткові обробники чи програмні модулі для зчитування файлу з параметрів запиту;

– текстовий серіалізований формат схеми даних займає менше місця, ніж файл.

При отриманні схеми-опису бази даних користувача за допомогою REST-запиту, сервіс генерації вилучує його з параметрів запиту. Отриманий серіалізований рядок десеріалізується в звичайний формат JSON засобами мови програмування який після цього зберігається локально на стороні сервісу у вигляді файлу. Цей файл надалі буде використовуватись сервісом генерації позбувшись залежності від результату запиту. На цьому етап отримання конфігураційного файлу сервісом завершено.

Далі виконується підготовка локального середовища для генерації файлів, це зокрема стосується папки для збереження файлів моделей. Як було згадано раніше, перед початком генерації, користувач спочатку виконує налаштування параметрів генерації, в тому числі і назву папки, в якій будуть зберігатись заархівовані файли. Теоретично, один і той же сервіс для генерації може використовуватись водночас декількома користувачами. Тому використання для назви папки назву вказану користувачем є небезпечним. В залежності від реалізації це може призвести або до конфлікту при створенні папки з існуючою назвою, чи до спільного використання папки для збереження моделей відразу декількома процесами генерації. Щоб запобігти такій потенційній проблемі, використовується наступний підхід. Замість того, щоб використовувати назву для папки вказану користувачем, система створює

папку зі згенерованим унікальним іменем за допомогою одного з підходів, зазвичай це генерація «uuid»-ідентифікатора. Далі система вже створює ще одну папку, але вже всередині попередньої з назвою, вказаною користувачем і використовує її для збереження моделей. По завершенню процесу генерації файлів саме ця папка буде заархівована та повернена відповіддю на запит на основний сервіс.

Далі починається етап формування моделей. Сервіс звертається до конфігураційного файлу зі схемою бази даних з метою отримання масиву таблиць. Процес формування моделей відбувається ітеративно для кожної таблиці, який складається з декількох послідовних кроків. Спочатку з масиву таблиць береться об’єкт таблиці. Для обраного об’єкту формується відповідний файл. В якості назви файлу використовується шаблон визначений користувачем на етапі налаштування процесу в якому ключові слова заміняються відповідними значеннями. Розширення для файлу також береться з конфігураційного файлу схеми, з компоненти «metadata». На основі комбінації назви та розширення створюється файл у заздалегідь створеній папці, до якого буде записуватись код моделі.

Після створення файлу моделі починається генерація коду моделі. Сервіс отримує з конфігураційного файлу налаштування мови програмування, особливості синтаксису чи інші правила, які впливають на кінцевий код. Використовуючи цю інформацію, він звертається до бази даних з конфігураційними даними з метою отримання фрагменту коду, який відповідає за базові імпорти сторонніх модулів та бібліотек, необхідних для правильної роботи моделі. Отримавши потрібний фрагмент коду з імпортами, він записується до створеного файлу моделі.

Далі до файлу записується наступний, основний фрагмент коду – сама структура моделі. Через те, що ORM-бібліотеки побудовані на принципах об’єктно-орієнтованого підходу, то модель зазвичай представляється у вигляді класу, або рідше у вигляді функції-конструктору. Сервіс аналогічно звертається конфігураційної бази даних та отримує «wireframe» або так званий каркас майбутньої моделі. Цей каркас слугує для того, щоб далі можна було безпроблемно додавати поля, зв’язки та характеристики моделі. Окрім цього, вона вже містить всі необхідні символи-роздільники та ключові слова, щоб компілятор чи інтерпретатор відповідної мови програмування вважав його валідним. Назвою класу слугує назва моделі, вказана в об’єкті таблиці зі схеми бази даних. Далі в цей каркас поміщаються поля-параметри,

які стосуються безпосередньо характеристик моделі. Наприклад це може бути конвертація імені моделі в множинну форму чи можливість автоматичного створення полів для збереження часової мітки створення та останньої зміни запису в таблиці. Всі можливі поля-характеристики моделі які підтримує бібліотека зберігаються всередині бази даних, а потенційні значення для них передаються всередині конфігураційного файлу бази даних.

В каркас моделі також додається програмний опис полів таблиці. Поля отримуються як і таблиці у вигляді масиву об’єктів, кожен з об’єктів поля опрацьовується окремо. Шаблон для створення коду поля так само зберігається в базі даних сервісу. Отримавши цей фрагмент коду, сервіс починає заповнювати його даними, зокрема це стосується назви поля та його характеристик. Окрему увагу потрібно звернути на визначення типу даних поля в моделі, адже кожна ORM-бібліотека містить свої типи даних. Для цього, всі сервіси генерації керуються загальним алгоритмом переведення типу даних в БД до відповідного типу даних моделі, який використовує значення типу даних, типу домену вказаних в об’єкті поля та перелік підтримуваних значень бібліотекою. Принцип роботи алгоритму визначення типу даних ORM-бібліотеки наведено на рисунку 4.2. Визначений тип даних поміщається в фрагмент коду поля. Після опрацювання всіх полів таблиці та генерації коду для них, цей код записується в каркас моделі на визначене місце.

Далі визначаються обмеження та індекси таблиці. Дані по ним, як і вся інформація беруться з конфігураційного файлу. Отримавши перелік обмежень та індексів, для них формується відповідна назва згідно з типом обмеження та шаблоном назви наведеним в конфігураційному файлі. Після генерації назви, сервіс знову звертається до конфігураційного файлу, але на цей раз з метою отримати об’єкти полів. Для цього використовуються ідентифікатори об’єктів полів які збережені у відповідному масиві. Зібравши всю необхідну інформацію, сервіс звертається до бази даних, отримує шаблон коду, який заповнює необхідною інформацією. По завершенню підготовки коду для всіх обмежень та індексів, зібрані фрагменти коду об’єднуються та записуються на визначене місце в каркасі моделі.

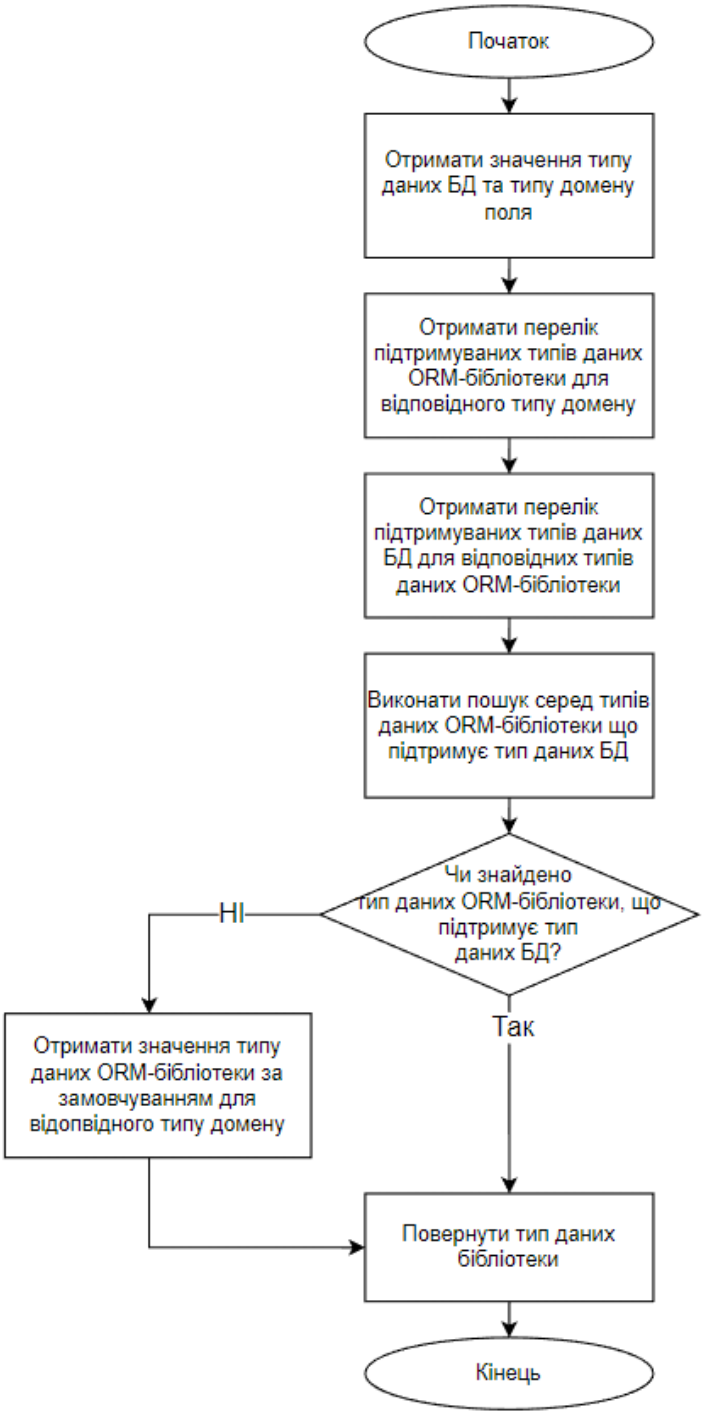

Рисунок 4.2 – Алгоритм вибору типу даних ORM-бібліотеки

Останньою частиною коду, яку необхідно записати до моделі є відношення між таблицями. Сервіс знову звертається до конфігураційного файлу з метою отримати перелік усіх відношень між таблицями. Для кожного відношення сервіс виконує пошук двох об’єктів таблиць – дочірньої таблиці та батьківської таблиці. Пошук

відбувається за допомогою ідентифікатору об’єктів таблиць які зберігаються в описі відношення. На основі інформації з цих об’єктів таблиць формується сама назва відношення, а також визначаються назви моделей, які повинні бути пов’язані. Далі визначається ключове слово, яке описує тип зв’язку. Для цього використовується одна з таблиць конфігураційної бази даних, що зберігає відношення між типами зв’язків в схемі бази даних та типами, які підтримуються ORM-бібліотекою. Наостанок визначається поведінка відношення при операціях видалення та зміни записів таблиці. Для цих операцій також існує системна таблиця в конфігураційній базі даних, яка відповідає за трансформацію типу поведінки зі схеми бази даних в тип, що підтримується бібліотекою. По завершенню опрацювання всіх відношень з конфігураційного файлу, зібрані фрагменти коду комбінуються та записуються в клас моделі у відповідному вигляді.

Останнім, необов’язковим етапом генерації коду є оформлення коду для експорту класу моделі з файлу. Він виконується лише для тих мов програмування, для яких потрібно виконувати писати певний код для експорту класу задля його подальшого використання. Для цього сервіс звертається до конфігураційного файлу та зчитує параметри експорту, якщо такі підтримуються мовою. Далі на основі параметрів експортування та параметрів самої мови програмування, він звертається до бази даних сервісу, щоб отримати необхідний фрагмент коду. В цей фрагмент підставляється назва класу, якщо це необхідно, наприклад при іменованому експорті в JavaScript. В іншому випадку виконується неіменований експорт або експорт за замовчуванням.

По завершенню етапу генерації коду, виконується чистка коду та його форматування. Під чисткою мається на увазі видалення всіх текстових ключових міток, які вказували на місце майбутнього розміщення згенерованих фрагментів коду. Мітки в коді легко знайти та видалити в автоматичному режимі через їх унікальну комбінацію символів і розміщення на окремому рядку. Процес форматування представляє собою застосування зазначених користувачем параметрів відступів. При виконанні генерації, код файлу форматується з використанням двох пробільних символів на один відступ. Сервіс звертається до конфігураційного файлу з метою отримати параметри форматування, які були зазначені користувачем. Якщо параметри збіглися з тим, що застосовувалися при генерації, то процес форматування

пропускається. В іншому випадку виконується пошук символів відступу та їх заміна на потрібну послідовність символів. Генерація коду вважається завершеною після створення файлів моделей для всіх таблиць зі схеми.

Передостаннім етапом створення моделей є архівація папки з файлами. Як згадувалось раніше, сервіс створює дві локальні папки для збереження файлів - перша, з унікальним іменем слугує для захисту від конфлікту імен, а друга, внутрішня папка, слугує папкою-сховищем файлів з вказаною назвою від користувача. Саме друга, внутрішня папка буде архівуватись та передаватись відповіддю. Для архівації використовується один з програмних модулів відповідної мови програмування. Також враховується тип архівування вказаний користувачем, зокрема ZIP чи RAR. Створений архів зберігається всередині папки з унікальним іменем, на рівні з внутрішньою папкою.

Останнім кроком роботи сервісу генерації є повернення результату його роботи. Генерація файлів це складний та тривалий процес, який потребує багато часу для отримання результату. Якщо запиту, що ініціював процес генерації не завершатись до отримання архіву, то є дуже багато вразливостей та ситуацій, що можуть перешкодити цьому. Якщо зникне інтернет-з’єднання, користувач перерве запит чи закриє вкладку браузеру, то весь результат генерації пропаде. В якості альтернативи можна замінити очікування результату запитом на опитування сервісу генерації час від часу на поточний стан генерації, і якщо процес завершено, то отримати результат. Але цей варіант отримання результату також не найкращий, адже не зрозуміло який інтервал між запитами потрібен, а більшість запитів на початку будуть марними. Саме тому для відправки архіву назад на основний сервіс застосовується технологія «webhook». Webhook у веб-розробці це метод збільшення або розширення функціональності веб-застосунку за допомогою користувацьких зворотних викликів [35]. На основному сервісі готується спеціальний запит, який будуть викликати сервіси генерації по завершенню процесу і віддавати архів. Це надасть захист від переривання запиту і запобіжить виконанню зайвих запитів.

ВИСНОВКИ

При розробці програмних застосунків, процес налаштування рівня даних складає основну частину, який в свою чергу займає багато зусиль та часу. Правильно спроектована база даних стає фундаментом майбутнього проекту, а грамотна інтеграція бази даних в проект забезпечує легкість та швидкість майбутнього її використання. Як один із способів інтеграції бази даних в код є використання ORM-бібліотек, які формують віртуальну базу даних з заздалегідь налаштованими інтерфейсами доступу та формування запитів. Та цей спосіб потребує багато часу та уважності, адже для всіх таблиць та зв'язків між ними необхідно сформувати відповідну ORM-модель. Метою цієї роботи було проектування та розробка веб-застосунку, що б дав можливість оптимізувати та покращити процес моделювання схеми даних, а процес створення коду для ORM-моделей – автоматизувати.

Для досягнення мети було визначено перелік таких задач, послідовне вирішення яких призвело до створення повноцінного веб-застосунку з урахуванням усіх вимог та рекомендацій. Першим етапом був аналіз предметної області, який дав представлення про основні аспекти веб-додатку та його складових, визначив методи та технології, які там потрібно використовувати. Окрім цього, було також визначено системи-аналоги за функціоналом, проаналізовано їх сильні та слабкі сторони та враховано на наступних етапах.

По завершенню етапу аналізу системи було детально оглянуто основні складові системи та способи їх технічної реалізації в системі. Зокрема це стосувалось методів візуального проектування, форматів представлення схеми даних у текстовому форматі та способів збереження конфігураційних даних для проведення генерації моделей. Для кожного з методів було визначено його сильні та слабкі сторони, а також основні способи використання.

В результаті проведення етапів аналізу та опису методів і технологій було визначено та сформовано вимоги до майбутнього веб-застосунку. Після програмної реалізації, він може бути застосований як повноцінне рішення для IT-фахівців з метою об'єднання етапів підготовки рівня даних проекту в одному місці, оптимізації процесу проектування схеми даних та автоматизації процесу підготовки віртуальної бази даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке бази даних, їх призначення та види? : веб-сайт. URL: https://futurenow.com.ua/shho-take-bazy-danyh-yih-pryznachennya-ta-vydy/ (дата звернення: 17.11.22).

2. Долгополов К. В. Розробка веб-платформи для генерації ORM-моделей на основі схеми реляційної БД. *Концепт науки XXI: стратегії, методи та наукові інструменти*: матеріали II Міжнародної студентської наукової конференції, м. Черкаси, 21 жовтня, 2022 р. Вінниця, 2022. С.158-159.

3. Григораш М. М. Дерева не ростуть до небес, або Як змінився український ринок IT за останні пів року: веб-сайт. URL: https://speka.media/dereva-ne-rostut-do-nebes-abo-yak-zminivsya-ukrayinskii-rinok-it-za-ostanni-pivroku-v47edv (дата звернення: 17.11.22).

4. Система автоматизації: веб-сайт. URL: https://wiki.tntu.edu.ua/Система_автоматизації (дата звернення: 17.11.22).

5. Автоматизовані інформаційні системи: опис, завдання та особливості: веб-сайт. URL: https://publish.com.ua/it-ta-web/avtomatizovani-informatsijni-sistemi-opis-zavdannya-ta-osoblivosti.html (дата звернення: 17.11.22).

6. Le Hegaret Ph. 100 Specifications for the Open Web Platform and Counting : веб-сайт. URL: https://www.w3.org/blog/2011/01/100-specifications-for-the-ope/ (дата звернення: 19.05.21).

7. Смирнов І. Веб приложение. Разница между сайтом, веб-приложением, SPA и PWA: веб-сайт. URL: https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/ (дата звернення: 17.11.22).

8. Различные виды и типы сайтов и веб-проектов. Типы сайтов: веб-сайт. URL: https://avada-media.ua/tipy-sajtov/ (дата звернення: 17.11.22).

9. Нотации модели сущность-связь (ER диаграммы): веб-сайт. URL: https://pro-prof.com/archives/8126 (дата звернення: 17.11.22).

10. Gordon K. Principles of Data Management : Facilitating Information Sharing / Keith Gordon. – Swindon: British Computer Society, 2007. – 276 с.

11. Стружкин Н. П., Годин В. В. Базы данных: проектирование : учебник для вузов, Москва : Издательство Юрайт, 2023. 477 с.

12. Мартін Ф. UML Основы. Краткое руководство по стандартному языку объектного моделирования. Санкт-Петербург : Символ, 2005. 184 с.

13. Маклаков, С.В. BPwin и Erwin: CASE-средства разработки информационных систем. Москва : Диалог-МИФИ, 1999. 256 с.

14. Roebuck K. Object-relational mapping (ORM): High-impact Strategies – What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors : book. La Vergne, Tennessee, US : Lightning Source Incorporated, 2011. 634 с.

15. Hayes D. ORM Patterns: The Trade-Offs of Active Record and Data Mappers for Object-Relational Mapping : веб-сайт. URL: https://www.thoughtfulcode.com/orm-active-record-vs-data-mapper/ (дата звернення: 19.05.21).

16. Портер М. Конкурентна стратегія. Техніки аналізу галузей і конкурентів. Київ : Наш Формат, 2020. 424 с.

17. Best Architecture Management Software: веб-сайт. URL: https://www.peerspot.com/categories/enterprise-architecture-management (дата звернення: 17.11.22).

18. Pros and Cons of erwin Data Modeler 2022 : веб-сайт. URL: https://www.trustradius.com/products/erwin-data-modeler/reviews?qs=pros-and-cons#reviews/ (дата звернення: 19.05.21).

19. ER диаграмма онлайн | Lucidchart : веб-сайт. URL: https://www.lucidchart.com/pages/ru/примеры/er-диаграмма-онлайн/ (дата звернення: 19.05.21).

20. Pros and Cons of Lucidchart 2022 : веб-сайт. URL: https://www.trustradius.com/products/lucidchart/reviews?qs=pros-and-cons#reviews/ (дата звернення: 19.05.21).

21. SAP PowerDesigner | Data Modeling and Enterprise Architecture : веб-сайт. URL: https://www.sap.com/products/technology-platform/powerdesigner-data-modeling-tools.html/ (дата звернення: 19.05.21).

22. SAP PowerDesigner Reviews & Product Details : веб-сайт. URL: https://www.g2.com/products/sap-powerdesigner/reviews#survey-response-692188/ (дата звернення: 19.05.21).

23. Chaudhari A. YAML vs JSON vs XML | What is the Difference Between Them?: веб-сайт. URL: https://www.csestack.org/yaml-vs-json-vs-xml-difference/ (дата звернення: 17.11.22).

24. Bhalla S. YAML vs JSON vs XML: Which One to Choose?: веб-сайт. URL: https://javascript.plainenglish.io/yaml-vs-json-vs-xml-what-to-choose-4c7a72417ff4/ (дата звернення: 17.11.22).

25. Means S. XML in a Nutshell : book. Sebastopol, California, US: O'Reilly Media, 2004. 600 с.

26. Достоинства и недостатки XML : веб-сайт. URL: https://studfile.net/preview/3021529/ (дата звернення: 19.05.21).

27. Вступ у JSON : веб-сайт. URL: https://www.json.org/json-uk.html (дата звернення: 19.05.21).

28. Денисенко А. YAML: просто о главном: веб-сайт. URL: https://highload.today/yaml/ (дата звернення: 19.05.21).

29. Спіцина Н. М., Шабельник Т.В., Бондаренко С.В. Інформаційні системи і технології : навч. посіб. Донецьк : ДНУЕіТ. 2011.- 290с.

30. Фаулер М. NoSQL: методология разработки нереляционных баз данных : книга. Москва : Вильямс, 2019. 192 с.

31. Долгополов К.В., Імангулова З.А. Метод генерації програмного коду ORM-моделей на основі схем реляційних баз даних. *Вісник Харківського національного автомобільно-дорожнього університету.* 2023. № 100. С.

32. Masse M. REST API Design Rulebook : book. Sebastopol, California, US : O'Reilly Media, 2011. 112 с.

33. What is the maximum length of a URL in different browsers? : веб-сайт. URL: https://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers (дата звернення: 19.05.21).

34. Upload limits for Internet Eplorer, Mozilla firefox, Google Chrome, Opera, IIS and ASP : веб-сайт. URL: https://www.motobit.com/help/scptutl/pa98.htm (дата звернення: 19.05.21).

35. What is a webhook? : веб-сайт. URL: https://www.redhat.com/en/topics/automation/what-is-a-webhook (дата звернення: 19.05.21).