МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2022 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС ДЛЯ ГЕНЕРАЦІЇ ОБ'ЄКТНИХ МОДЕЛЕЙ НА ОСНОВІ ВХІДНИХ ДАНИХ КОРИСТУВАЧА

Спеціальність 122 «Комп'ютерні наук»

122 – БКР – 401.2181017

Виконав студент 4-го курсу, групи 401
_____ І. А. Ларченко
«13» червня 2022 р.

Керівник: д-р пед. наук, проф.
_____О. П. Мещанінов
«13» червня 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20___р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Ігорю Анатолійовичу Ларченко.

1. Тема кваліфікаційної роботи «Прикладний програмний інтерфейс для генерації об'єктних моделей на основі вхідних даних користувача».

Керівник роботи Мещанінов О. П., д-р пед. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «___» _____ 20___ р. № ____

2. Строк представлення кваліфікаційної роботи студентом «___» _____ 20__ р.

3. Вхідні (початкові) дані до роботи: методи застосування принципів об'єктно орієнтованого програмування при розробці корпоративних застосунків; бібліотеки та технології для розробки корпоративних застосунків.

Очікуваний результат роботи: функціонуючий прикладний програмний інтерфейс для генерації об'єктних моделей згідно специфікації користувача.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– огляд на сучасні технології та фреймворки, які використовуються для розробки багатошарових корпоративних додатків;

– пошук аналогів розроблюваної системи;

– огляд методів і технологій генерації вихідного коду для подальшого його компілювання;

– тестування розроблюваної системи на коректність функціонування.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини:

- Проаналізувати наявні умови праці в робочому приміщенні;
- Визначити достатній рівень показників для робочого приміщення, де проводяться роботи з розробки системи;
- Визначити основні правила техніки безпеки при роботі з комп'ютерною технікою.

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	О. В. Макарова	

Керівник роботи __д-р пед. наук, проф. Мещанінов О. П._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Ларченко І. А.
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_23_» _____листопада____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Прикладний програмний інтерфейс для генерації об’єктних моделей на основі вхідних даних користувача.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	26.10.2021	27.10.2021	
2	Отримання завдання на виконання БКР	23.11.2021	23.11.2021	
3	Складання календарного плану на період виконання БКР	09.12.2021	10.12.2021	
4	Отримання завдання на переддипломну практику	18.05.2022	18.05.2022	
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	23.05.2022	04.06.2022	
6	Оформлення звіту з переддипломної практики	04.06.2022	06.06.2022	
7	Налаштування середовища для розробки системи	28.03.2022	29.03.2022	
8	Формування базових функціональних вимог до системи	01.04.2022	04.04.2022	
9	Створення генератора класів для об’єктних моделей	22.05.2022	26.05.2022	
10	Створення генератора веб-сервісів згідно об’єктної моделі	26.05.2022	28.05.2022	
11	Оформлення звіту БКР	10.05.2022	16.06.2022	
12	Попередній захист БКР на засіданні комісії кафедри	30.05.2022	31.05.2022	
13	Консультація з керівником БКР	04.06.2022	10.06.2022	
14	Виправлення помилок в звіті	26.05.2022	01.06.2022	
15	Подання БКР рецензенту	16.06.2022	18.06.2022	
16	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	20.06.2022	22.06.2022	
17	Захист БКР перед екзаменаційною комісією (ЕК)	27.06.2022	29.06.2022	

Розробив студент _____Ларченко І. А._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи __д-р пед. наук, проф. Мещанінов О. П ___ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 2022 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили
Ларченка Ігоря Анатолійовича

Тема: «Прикладний програмний інтерфейс для генерації об’єктних моделей на основі вхідних даних користувача»

Об'єкт роботи – методи генерації вихідного коду об’єктних моделей.

Предмет роботи – прикладний інтерфейс для генерації об’єктних моделей.

Метою бакалаврської кваліфікаційної роботи є проєктування, розробка та тестування прикладного програмного інтерфейсу для генерації сутностей та об’єктних моделей на основі введених користувачем даних про ці сутності.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, трьох розділів, висновків та додатків.

У першому розділі проведено аналіз предметної області, визначено аналоги розроблюваної системи та розглянуто принципи роботи подібних прикладних інтерфейсів.

У другому розділі проаналізовано сучасні технології для розробки систем корпоративного рівня та визначено технологічний стек, який використано для розробки інтерфейсу в який ввійшли такі технології: Java, Spring, Hibernate, PostgreSQL, XML.

У третьому розділі описано процес проєктування системи, виділено нюанси розробки, проведено тестування системи та описано покроковий алгоритм роботи з програмним інтерфейсом.

В результаті розроблено прикладний програмний інтерфейс для генерації об’єктних моделей з можливістю гнучкого конфігурування моделей, які піддаються генерації.

Бакалаврська кваліфікаційна робота містить 77 сторінок, 30 рисунків, 5 таблиць, 26 використаних джерел та 5 додатків.

Ключові слова: прикладний програмний інтерфейс, об’єктно-орієнтоване програмування, генерація об’єктних моделей, веб-сервіси, реляційні бази даних.

ABSTRACT

bachelor's degree qualification work of student of group 401 BSNU named after Petro Mohyla

Larchenko Ihor Anatolyovich

Topic: «Application programming interface for generating object models based on user`s input»

Object of work - methods of generating the source code of object models.

The subject of the work is an application interface for generating object models.

The purpose of the bachelor's thesis is to design, develop and test an application software interface for generating entities and object models based on user-entered data about these entities.

The work consists of a professional section and a special section on labor protection. The explanatory note consists of an introduction, three chapters, conclusions and appendices.

In the first section the analysis of the subject area is carried out, analogues of the developed system are defined and the principles of work of similar application interfaces are considered.

The second section analyzes modern technologies for the development of enterprise-level systems and identifies the technology stack used to develop the interface, which includes the following technologies: Java, Spring, Hibernate, PostgreSQL, XML.

The third section describes the system design process, highlights the nuances of development, system testing and describes a step-by-step algorithm for working with the software interface.

As a result, an application programming interface has been developed for generating object models with the ability to flexibly configure the models to be generated.

The bachelor's thesis contains 77 pages, 30 figures, 5 tables, 26 sources used and 5 appendices.

Keywords: application programming interface, object-oriented programming, object model generation, web services, relational databases.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД – система керування базами даних

API – прикладний програмний інтерфейс
ASCII – американський стандартний код для інформаційного обміну
DOM – об'єктна модель документа
EMF – фреймворк моделювання Eclipse
HTML – мова гіпертекстової розмітки
HTTP – протокол гіпертекстової передачі даних
MVC – модель-представлення-контролер
ORM – об’єктно-реляційна проекція
RDBMS – система керування реляційними базами даних
REST – передача репрезентативного стану
SQL – мова структурованих запитів
XAML – розширювана мова розмітки застосунків
XML – розширювана мова розмітки
XSD – визначення схеми XML

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«Прикладний програмний інтерфейс для генерації об’єктних моделей на основі вхідних даних користувача»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 401.21810117

Виконав студент 4-го курсу, групи 401

_____*І. А. Ларченко*_____

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Керівник: _____*д-р пед. наук, проф.* _____

(наук. ступінь, вчене звання)

_____ *О. П. Мещанінов* _____

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Миколаїв – 2022

ВСТУП

З розвитком цифрових технологій все більше сфер нашого життя переходять стають залежними від комп’ютерної техніки. Усе що ми можемо собі уявити може бути представлене у вигляді упорядкованих структур даних: книжки, електронні документи, наукові статті, автомобільна техніка тощо. Даний фактор суттєво впливає на наше суспільство і нам набагато простіше знайти ту чи іншу інформацію в мережі.

Через це виникає потреба в швидкому процесі абстракції об’єктів з реального світу в інформаційний простір. Частково дана потреба задовольняється за допомогою сервісів-генераторів об’єктних моделей. Їх основна ціль – за мінімальних витрат створювати комплексні моделі для їх зберігання в пам’яті комп’ютера та подальшій реалізації бізнес-логіки з ними.

Метою даної роботи є розробка та тестування прикладного програмного інтерфейсу для генерації об’єктних моделей з можливістю подальшого розширення його функціональної бази.

Об’єктом дослідження даної роботи є методи генерації об’єктних моделей.

Предметом дослідження роботи є прикладний програмний інтерфейс для генерації об’єктних моделей на основі вхідних даних користувача.

Актуальність роботи виявляється в високому попиті різних сфер бізнесу на перенесення процесів та структур в цифровий вигляд. Переважна більшість сучасних компаній мають своє відображення в цифровому світі.

Основні завдання, які повинні бути розв’язані в ході даної роботи:

- проведення аналізу сучасних методів генерації вихідного коду;
- огляд існуючих аналогів розроблюваного програмного інтерфейсу;
- визначення сучасних технологій для розробки корпоративних застосунків середнього рівня;
- проєктування, розробка та тестування прикладного програмного інтерфейсу для генерації об’єктних моделей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд на предметну сферу

Що таке взагалі генератор? Поняття генератору в різних джерелах трактується по різному. В технічній та фізико-математичній сфері генератор визначений як пристрій, який виробляє яку-небудь продукцію, виробляє електроенергію або трансформує один вид енергії в інший [1]. Це може бути автомобільний генератор, паровий генератор чи генератор електромагнітних хвиль. В сфері інформаційних технологій поняття генератору є доволі поширеним та різностороннім, але воно далеко не відступило від понять з інших галузей. Основне призначення генераторів в галузі інформаційних технологій є трансформація даних з одного виду в інший. Яскравими прикладами таких генераторів виступають генератори псевдовипадкових чисел, генератори звітів або документації, генератори паролей.

Розглянемо поняття генератора більш детально на прикладі генератора псевдовипадкових чисел. Виклик будь-якої функції генерації псевдовипадкового числа не просто генерує число з пустого місця. Для отримання результату функція вимагає в якості вхідного параметру ще одне число, яке виступає в якості насіння (англ. seed). В більшості генераторів псевдовипадкових чисел в якості насіння використовується поточний час системи в мілісекундах. Вхідний параметр в ході роботи складної логіки генератора багатократно трансформується складними математичними та логічними функціями і на виході ми отримуємо згенероване число [2].

Приблизно так само працюють генератори звітів або документацій. На вхід програма-генератор отримує деякий шаблон та звітні дані. В ході своєї роботи отримані аргументи компонуються згідно шаблону та зберігають готовий звіт у файлову систему чи одразу відправляють на друк.

Також, існують доволі складні генератори, основною функцією яких є генерація вихідного коду. Такі генератори містять в собі комплексну систему, яка виконує функції подібні до роботи компіляторів чи інтерпретаторів різних мов

програмування. В якості вхідних параметрів такі системи приймають величезну кількість метаданих, які в ході роботи генератора проходять декілька етапів перед видачою готового коду:

– введення метаданих;
– перевірка цілісності та валідація введених метаданих;
– компонування та інтерпретація даних в зрозумілий для системи формат;
– генерація вихідного коду на основі шаблонів;
– збереження згенерованих файлів або негайна їх компіляція та виконання.

Найбільшого поширення такі генератори набрали серед мов програмування які підтримують об’єктно-орієнтовану парадигму. Таке поширення зумовлено тим, що будь-яка структура в об’єктно-орієнтованій мові може бути описана об’єктом, який в свою чергу є набором примітивів, які є його атрибутами. На рисунку 1.1 зображено приклад об’єктів в вигляді діаграми класів.

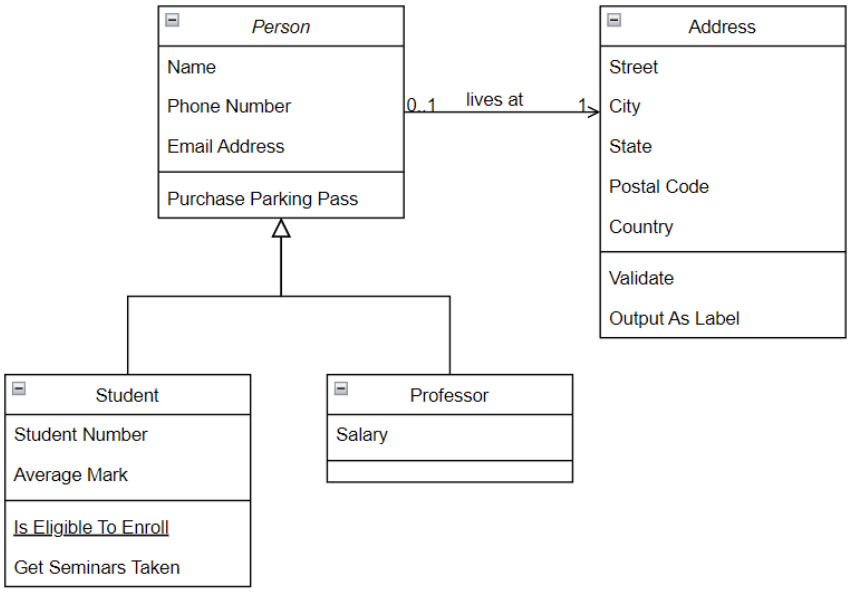

Рисунок 1.1 – Діаграма класів об’єктних моделей

Об’єктні моделі представлені на попередньому рисунку складаються здебільшого з простих структур, таких як рядки, цілі числа, числа з плаваючою крапкою тощо. Також на діаграмі зображено взаємозв’язки класів, наприклад клас Person є предком класів Student та Professor і в той же час має асоціацію до класу

Address. Опис такої структури класів будь-якою об’єктно-орієнтованою мовою не займе багато часу і може бути автоматизована завдяки генераторам вихідного коду.

Що собою представляють метадані, які згадувались вище? Метадані – це дані, які містять в собі інформацію про склад чи структуру того чи іншого об’єкта [3]. Метаданими класу, наприклад, виступає інформація про його атрибути, тип даних цих атрибутів, зв’язки з іншими об’єктами тощо. В якості прикладу метаданих розглянемо клас Professor. На рисунку 1.2 продемонстровано загальну структуру метаданих для даного класу.

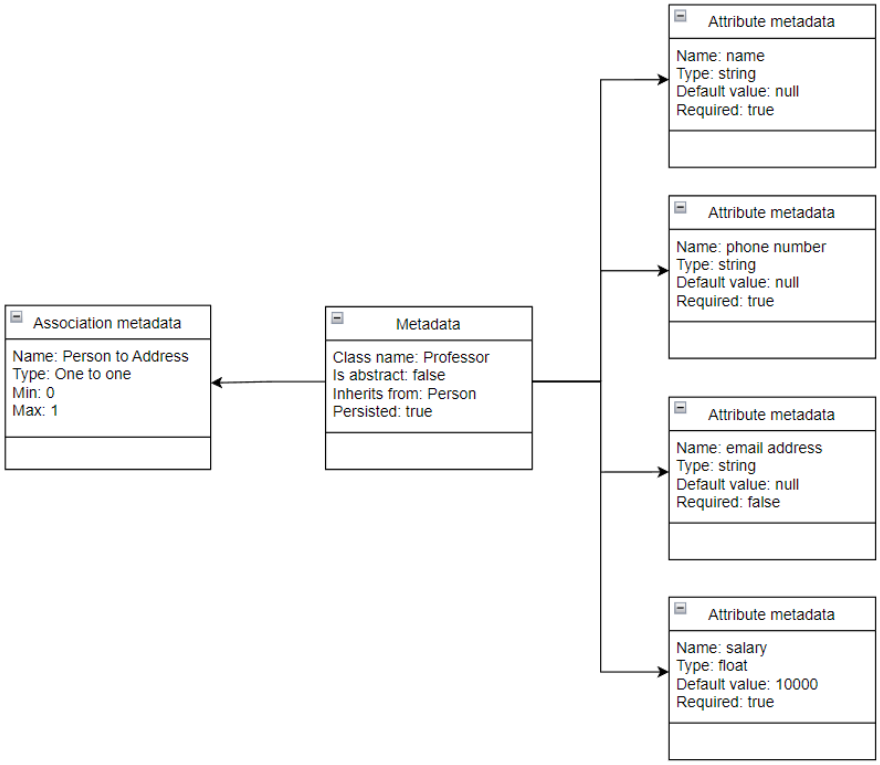

Рисунок 1.2 – Загальна структура метаданих для класу Professor

Метадані для даного класу містять, безпосередньо назву класу (Class name), чи є він абстрактним типом (Is abstract), від якого класу він успадковується (Inherits from) та чи зберігається він у базі даних (Persisted). Крім того метадані містять інформацію про атрибути класу (Attributes metadata) а саме: ім’я поля (Name), тип даних (Type), значення за замовчуванням (Default value) і чи поле необхідно заповнювати (Required). Окремо також міститься інформація про відношення класу до інших класів (Association metadata).

Основною перевагою метаданих є те, що їх можна зберігати та використовувати у такому вигляді, який зрозумілий комп’ютеру. Саме тому їх також можна перевіряти на цілісність та правильність програмно.

Після визначення метаданих об’єктних моделей наступним етапом генерації є об’єднання метаданих з шаблоном. Шаблон – це своєрідний каркас із заздалегідь прописаним вихідним кодом, який при об’єднанні з метаданими формує повністю працюючий програмний код, який вимагає лише компіляції або інтерпретації.

1.2 Пошук та аналіз наявних аналогів розроблюваної системи генерації об’єктних моделей

Згідно з джерелом [4] існує величезна кількість різносторонніх генераторів коду, які виконують функцію генерації вихідного коду згідно введеного користувачем або самим розробником шаблону. Більшість з представлених джерелом рішень є системи для прискорення процесу розробки інших програмних застосунків. Тобто, інші розробники можуть використовувати генератори коду для прискорення процесу розробки своїх програм. Також, варто відзначити, що переважна більшість існуючих генераторів вихідного коду є складовою або окремою частиною інших більш комплексних продуктів.

Розглянемо такий проєкт як T4. Text Template Transformation Toolkit (також відомий як «T4») — це безкоштовна структура генерації тексту на основі шаблонів із відкритим вихідним кодом . Вихідні файли T4 зазвичай позначаються розширенням ".tt".

T4 використовується розробниками як частина програми або інструментальної рамки для автоматизації створення текстових файлів з різними параметрами. Ці текстові файли можуть бути будь-якого текстового формату, наприклад коду (наприклад , C#), XML , HTML або XAML .

T4 використовує користувацький формат шаблону, який може містити код .NET і рядкові літерали . Це аналізується інструментом командного рядка T4 на код .NET, компілюється та виконується. Результатом виконаного коду є текстовий файл, згенерований шаблоном. T4 також можна повністю запускати в програмах

.NET за допомогою класу Text Transformation, який усуває потребу кінцевому користувачеві встановлювати Visual Studio.

T4 використовується в Microsoft в ASP.NET MVC для створення представлень і контролерів, ADO.NET Entity Framework для генерації сутностей і ASP.NET Dynamic Data . [5]

Серед окремих рішень, які сфокусовані на генерації вихідного коду слід виділити такий проєкт як CodeSmith Generator.

CodeSmith Generator — це інструмент для розробки програмного забезпечення, який допоможе вам швидше виконувати роботу. Технічно це керований шаблонами генератор вихідного коду, який автоматизує створення загального вихідного коду програми для будь-якої мови (C#, Java, VB, PHP, ASP.NET, SQL тощо) [6]. CodeSmith Generator містить багато корисних шаблонів, а також цілі набори шаблонів для створення перевірених архітектур. Постачальник даного продукту запевняє, що користувач може легко змінити будь-які шаблони або написати власний, щоб створити свій код саме так, як вам потрібно.

Синтаксис CodeSmith Generator майже ідентичний ASP.NET. Отже, якщо користувач знайомий з ASP.NET, він зможете швидко вивчити синтаксис шаблону. Також, користувач може використовувати мови C# або VB.NET у своїх шаблонах, а створені шаблони можуть виводити будь-яку мову на основі ASCII (текстової).

Основними перевагами даного рішення, за словами розробника, є:

- зменшення повторюваного коду;
- генерація коду за менший час з меншою кількістю помилок;
- створення послідовного коду, який відповідає стандартам користувача;
- створення власних шаблонів для будь-якої мови.

Також, доволі популярним рішенням генерації коду мовою Java є Acceleo. Даний застосунок є плагіном до інтегрованої середи розробки Eclipse. Acceleo — це технологія на основі шаблонів, що включає інструменти для створення власних генераторів коду [7]. Це дозволяє автоматично створювати будь-який вихідний код з будь-якого джерела даних, доступного у форматі EMF [8].

Головні переваги, які виділяє розробник:

– висока здатність до налаштування
– сумісність
– легкий старт

Ще одним прикладом генератора цілих сервісів є проєкт Bootify [9]. За допомогою даного рішення користувач може на основі схеми бази даних створити повністю функціонуючий сервіс, який готовий до запуску одразу після генерації. Проєкт використовує сучасні рішення корпоративної Java для генерації вихідного коду сервісів, а саме: Hibernate, Spring framework, REST Web Services та багато інших. На рисунку 1.3 зображено приклад генерації об’єктної моделі через користувацький інтерфейс застосунку.

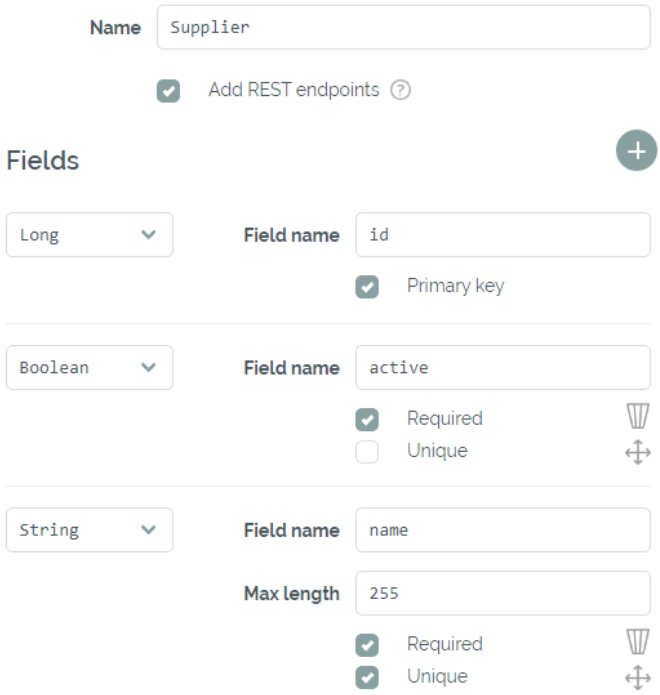

Рисунок 1.3 – Приклад створення структури об’єктної моделі

Після того, як користувач повністю опише об’єктні моделі, застосунок почне генерувати класи, таблиці для бази даних та веб сервіси для доступу до даних через протокол HTTP з використанням REST API. Розглянемо даний приклад більш детально.

Взагалі даний сервіс дозволяє доволі гнучко побудувати необхідні об’єктні моделі та побудувати взаємозв’язки між ними. Користувач може згенерувати наступні типи сутностей: класи та перелічення. В якості метаданих для класів та

перелічень виступають дані про назву класу, наявність REST веб сервісів, характеристики атрибутів, відношення до інших класів, тощо. Далі метадані об’єднуються зі заздалегідь створеними шаблонами класів сутностей, репозиторіями для доступу до бази даних, веб сервісів, SQL скриптів і компанують це все у один архів з готовим до запуску проєктом. Саме згідно такого сервісу будуть формуватися технічні вимоги для розроблюваної системи.

Окремо серед рішень даного типу варто виділити такий проєкт як aicodoo. Це плагін для інтегрованої середи розробки який на основі описаної структури класу розробником генерує повністю готовий до запуску сервіс мовою програмування Java з готовими таблицями бази даних, веб сервісами для доступу до даних за допомогою HTTP протоколу. Його особливість заключається в тому, що він генерує код не на основі шаблонів, а за допомогою алгоритмів машинного навчання. Від користувача потребується лише описати назву класу, основні атрибути і з контекстного меню середи розробки викликати відповідну функцію та налаштувати генерацію коду відповідним чином. Розробник дає користувачеві можливість тренувати нейронну мережу на готових проєктах з системи керування версіями GitHub, що дозволяє генерувати гнучкі сервіси та програмні застосунки.

1.3 Постановка задачі та формування технічного завдання програмного інтерфейсу для генерації об’єктних моделей

Провівши огляд на предметну область та проаналізувавши існуючі аналоги розроблюваної системи, можна перейти до етапу формування функціональних вимог до генератора об’єктних моделей. Для спрощення процесу розробки системи розіб’ємо її на кілька модулів. Перш за все необхідно створити схему, за якою будуть формуватися та зберігатися метадані об’єктних моделей. Це дозволить зберігати метадані структуровано та зрозуміло як для користувача так і для комп’ютера. Найкращим форматом для збереження великої кількості даних є формат XML (eXtensible Markup Language). Завдяки даному формату можна описати складно структуровані об’єкти та структури та зберігати їх у файловій системі. Завдяки деревовидній структурі, зчитування та обробка даних XML файлів є доволі простою та інтуїтивно зрозумілою.

Структура метаданих об’єктних моделей повинна містити усі необхідні дані, які в подальшому будуть використані при генерації SQL скриптів, класів та сервісів для доступу до об’єктів цих класів. На рисунку 1.4 представлено базову структуру файлів з метаданими.

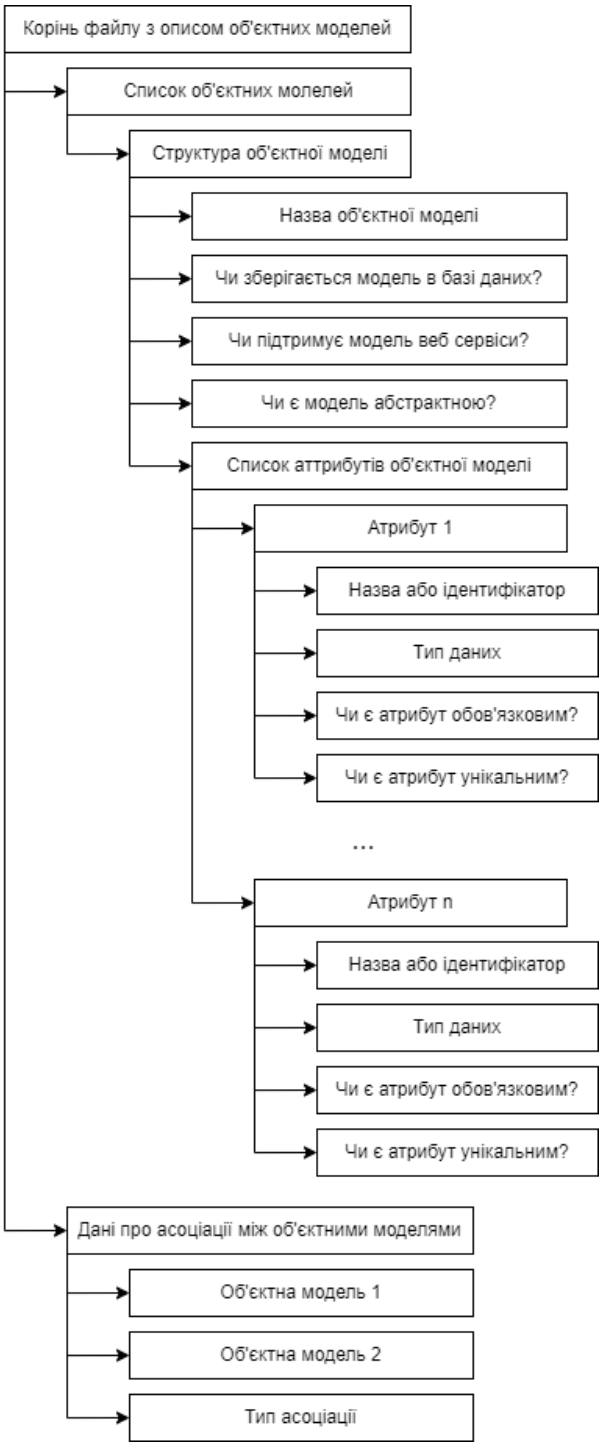

Рисунок 1.4 – Базова структура файлу з вхідними метаданими

Як видно з рисунку файл з описом об’єктних моделей складається з двох основних частин: список з даними про структуру об’єктних моделей та дані про асоціації між даними моделями. Розглянемо кожну з частин більш детально.

Структура файлу з об’єктними моделями містить основні дані про модель, яка буде генеруватися. Перш за все це назва, за якою вона буде ідентифікуватися. Назва повинна буди унікальною, щоб уникнути помилок при генерації моделей. Наступна характеристика моделі визначає, чи потрібно генерувати таблиці та SQL скрипти для зберігання та доступу до об’єктної моделі в реляційних базах даних. В залежності від наступного атрибуту, генератор створюватиме веб сервіси для доступу до об’єктних моделей, які зберігаються в базі даних. Останній загальний атрибут визначає, чи є об’єктна модель абстрактною.

Список атрибутів містить набір основних полів, які будуть згенеровані в об’єктній моделі. Кожне поле повинен мати наступні атрибути: назва поля, тип даних, чи повинен атрибут бути унікальним та обов’язковим до введення.

Друга частина файлу з об’єктними моделями містить дані про асоціації між моделями. Асоціації розділені з об’єктними моделями для того, щоб при генерації об’єктів не виникало помилок про те що об’єкти задіяні в асоціації ще не існують. Тобто, генерація об’єктних моделей починається з генерації класів, і тільки після цього генеруються асоціації між цими класами. Дані про асоціацію містять такі атрибути як об’єкт-джерело асоціації, об’єкт-ціль асоціації та тип асоціації. До типів асоціацій відноситься наступні: один до одного, багато до одного, багато до багатьох.

Після створення файлу зі структурою моделей необхідно провести попередню його валідацію. У випадку некоректності введених даних користувач має отримати повідомлення зі вказівками де саме були допущені помилки. Крім того, валідація повинна проводитися в декілька етапів: на етапі опису моделі та на етапі генерації.

Розглянувши базову структуру файлу з об’єктними моделями можна перейти до розробки специфікації для генератора об’єктних моделей. Сервіс-генератор повинен виконувати такі основні функції: генерація, безпосередньо, класів

об’єктної моделі, створення таблиць та SQL скриптів для бази даних, реалізація веб сервісів доступу до об’єктів за протоколом HTTP. Додатково можна згенерувати веб сторінки для перегляду та редагування даних з браузеру у вигляді таблиць. Розглянемо кожен з цих аспектів більш детально.

Перш за все, генератор повинен згенерувати класи на основі введених метаданих. Для цього необхідно заздалегідь створити шаблон класу, який буде гнучко маніпулювати метаданими та згенерує коректний файл з вихідним кодом класу.

Наступним етапом є формування таблиць та скриптів для бази даних. Для цього генератору, так само як і в випадку генерації класів, необхідно заздалегідь сформувати шаблони з SQL кодом, які так само об'єднуючись з метаданими будуть видавати функціонуючі SQL файли.

При створенні веб сервісів необхідно заздалегідь визначитись з підходом який буде використовуватись для доступу до даних через протокол HTTP. В сучасних реаліях найбільш поширеним інтерфейсом доступу до даних є REST. REST (або Representational State Transfer) це архітектурний стиль який регулює передачу даних та об’єктів через мережу та протокол HTTP. Його основна перевага в тому, що завдяки набору HTTP методів можна реалізувати CRUD (create, read, update, delete) операції з об’єктами які зберігаються на сервері [10]. Доволі поширеним форматом передачі даних через протокол REST є JSON (JavaScript object Notation). Формат даних в такому вигляді доволі просто серіалізувати в інших додатках, адже він підтримується всіма популярними мовами програмування. Веб сервіси повинні також генеруватися но основі шаблонів та підтримувати всі основні операції даного архітектурного підходу. На рисунку 1.5 зображено загальну схему сервісу на основі REST.

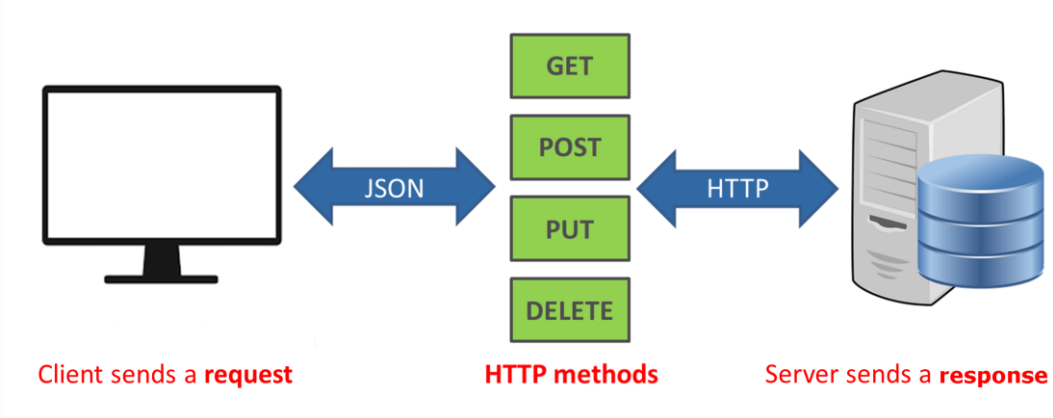

Рисунок 1.5 – Загальна схема роботи сервісів на основі REST API

Для спрощення процесу перегляду та редагування об’єктів можна згенерувати прості веб-сторінки з таблицями для перегляду даних цих об’єктів.

Після процесу генерації об’єктних моделей усі вихідні файли, отримані в ході роботи сервісу повинні бути відкомпільовані та повинен бути сформований кінцевий архів із сервісом, який буде повністю готовий до запуску та роботи.

Тестування роботи прикладного програмного інтерфейсу для генерації об’єктних моделей повинне проходити в кілька етапів: перший етап – етап після генерації моделей для перевірки їх цілісності та коректності та на етапі функціонування сервісу для того, щоб перевірити коректність роботи клієнт-серверної частини сервісу.

Висновки до розділу 1

В даному розділі було розглянуто основні поняття та терміни, які стосуються генерації вихідного коду на основі вхідних даних. Було з’ясовано, що переважна більшість генераторів розроблена для мов програмування, які основані на об’єктно-орієнтованій парадигмі. Це обумовлено тим, що представляти моделі в вигляді об’єктів набагато легше та ефективніше. Було розглянуто основні принципи та методи які використовуються в процесі генерації файлів чи тексту з вихідним кодом, а саме: використання шаблонів в якості скелету для коду, використання метаданих як спосіб збереження інформації про дані. Було досліджено, що перед етапом генерації вихідного коду метадані повинні бути належним чином провалідовані: перш за все вони повинні пройти перевірку на цілісність та повноту їх введення.

Було розглянуто сучасні аналоги розроблюваного програмного інтерфейсу. В ході огляду готових проєктів було розглянуто різні підходи та методи, які використовуються для генерації вихідного коду. Проаналізувавши існуючі системи, було зроблено висновки про мінімальні функціональні вимоги розроблюваної системи.

При розробці функціональних вимог було зроблено наголос на простоту реалізації, можливість подальшого масштабування системи та коректність генерації об’єктних моделей. Крім того, було створено специфікацію структури метаданих об’єктних моделей, було приблизно описано поступовий процес генерації об’єктних моделей. Також, було сформовано план тестування прикладного програмного інтерфейсу для генерації об’єктних моделей. Нижче перелічені основні вимоги до програмного інтерфейсу:

- стандартизоване введення та зберігання метаданих об’єктних моделей;
- гнучкі шаблони для генерації коду;
- швидкий процес генерації;
- створення компоненту для компіляції та компонування вихідного коду;

2 ТЕХНОЛОГІЇ ТА МЕТОДИ ДЛЯ РЕАЛІЗАЦІЇ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСТУ ГЕНЕРАЦІЇ ОБ’ЄКНТНИХ МОДЕЛЕЙ

2.1 Огляд на сучасні технології для розробки корпоративних систем

Успіх будь-якого проєкту в сфері інформаційних технологій на 70% залежить від технологічного стеку, який був обраний для його реалізації. Можна розробити досконалий дизайн, прозоро описати функціональні вимоги, але якщо обрані технології не надають повний інструментарій для реалізації цих вимог, то проєкт в такому разі приречений на провал.

На сьогоднішній день існує велика кількість методів та технологій для реалізації систем різного рівня складності. Вибір технології, яка буде використовуватись в ході реалізації залежить від специфіки розроблюваної системи і проводиться за відповідними критеріями. Основними критеріями вибору технології для створення тієї чи іншої системи є:

- можливість використання технології в різних системах;
- простота використання даної технології;
- гнучкість методів для вирішення тих чи інших задач.

Для вирішення поставленої в роботі задачі, перш за все необхідно провести аналіз мов програмування, які користуються популярністю та мають обширний інструментарій для розробки сучасних систем. Згідно з дослідженням компанії Tiobe [11], в трійку найбільш популярних мов програмування для розробки програмних застосунків на 2022 рік є Python, C, та Java. На рисунку 2.1 представлено діаграму з динамікою зміни популярності мов програмування серед розробників протягом 20-ти років.

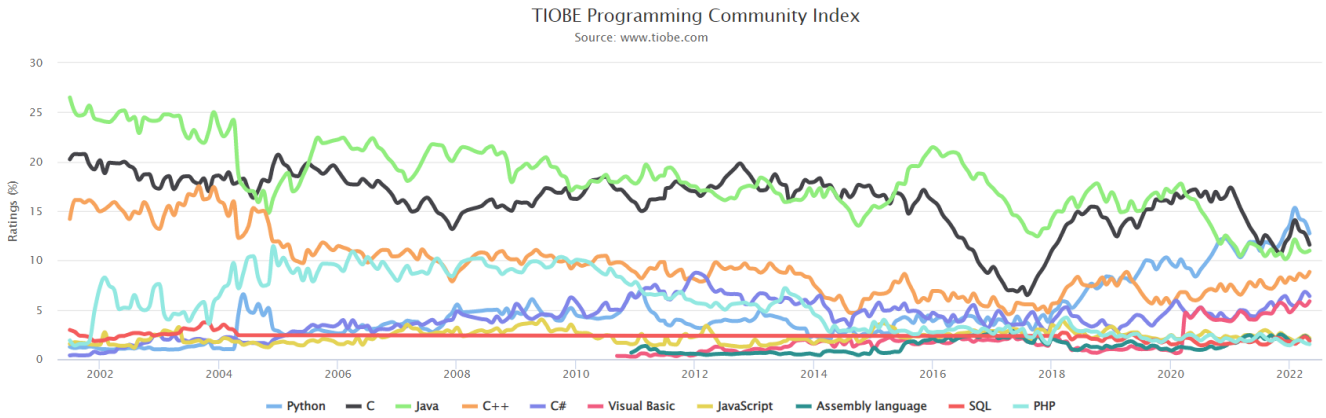

Рисунок 2.1 – Динаміка популярності різних мов програмування протягом двадцятирічного періоду

Слід мати на увазі, що в переважній кількості випадків різні технологічні стеки використовують в зв’язці. Наприклад, використати в проєкті корпоративного рівня лише методи, які надає Java неможливо, адже в ході виконання програми неможливо буде зберігати дані поза межами оперативної пам’яті. Тому в корпоративній розробці обов’язково використовуються реляційні бази даних чи більш сучасні нереляційні аналоги.

C# — сучасна, об'єктно-орієнтована та суворо типізована мова програмування. C# дозволяє розробникам створювати широкий спектр безпечних і надійних програм, які працюють на базі .NET. C# має свої коріння в сімействі мов C і буде відразу знайомий програмістам C, C++, Java та JavaScript.

C# — це об'єктно-орієнтована мова програмування, яка надає мовні конструкції для безпосередньої підтримки цієї концепцій, роблячи C# природною мовою для створення та використання програмних компонентів. З моменту свого виникнення в C# були додані функції для підтримки нових робочих навантажень і нових методів проєктування програмного забезпечення. За своєю суттю C# є об'єктно-орієнтованою мовою, тобто розробник самостійно визначає типи та їх поведінку.

Функцій C# допомагають створювати надійні та довговічні програми. Автоматична очистка ресурсів відновлює оперативну пам’ять, зайняту непотрібними в конкретний момент часу використаними об’єктами. Типи, які допускають значення NULL, захищають від змінних, які не посилаються на

виділені об’єкти. Обробка винятків забезпечує структурований і розширений підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Синтаксис мовного інтегрованого запиту (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела. Підтримка мови для асинхронних операцій забезпечує синтаксис для побудови розподілених систем. C# має уніфіковану систему типів. Усі типи C#, включаючи примітивні типи, такі як int і double, успадковуються від одного кореневого типу об’єкта. Усі типи мають набір загальних операцій. Цінності будь-якого типу можна зберігати, транспортувати та використовувати узгоджено. Крім того, C# підтримує як визначені користувачем типи посилань, так і типи значень. C# дозволяє динамічно розподіляти об'єкти та вбудовано зберігати полегшені структури. C# підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типів. C# надає ітератори, які дають змогу реалізаторам класів колекції визначати користувацьку поведінку для клієнтського коду.

C# робить акцент на контролі версій, щоб програми та бібліотеки могли розвиватися з часом сумісним чином. Аспекти дизайну C#, на які безпосередньо вплинули міркування щодо версій, включають окремі віртуальні модифікатори та модифікатори перевизначення, правила вирішення перевантаження методів і підтримку явного оголошення членів інтерфейсу.

Серед недоліків даної технології варто виділити те, що вона в цілому орієнтована на платформу Windows і цей фактор безпосередньо впливає на кросплатформеність розроблюваних застосунків. Також, основною проблемою цієї мови програмування є недостатня обернена сумісність коду. Це означає що якщо в ході розробки вийшла нова версія даної мови, зазвичай для переносу проєкту на нові рейки потрібно повністю переписувати весь вихідний код, або залишатися на попередніх версіях.

Окрім базового функціоналу, який надає дана мова програмування, існує велика кількість бібліотек різного рівня, які допомагають розробникам створювати комплексні системи. Найбільш популярним фреймворком C# є ASP. NET.

ASP.NET розширює платформу .NET інструментами та бібліотеками спеціально для розробки веб-програм.

Нижче перелічено основні речі, які ASP.NET додає до платформи .NET:

- Базовий фреймворк для обробки веб-запитів на C# або F#
- Синтаксис шаблону веб-сторінки, відомий як Razor, для створення динамічних веб-сторінок за допомогою C#
- Бібліотеки для поширених веб-шаблів, наприклад Контролер перегляду моделі (MVC)
- Система аутентифікації, яка включає бібліотеки, базу даних і шаблонні сторінки для обробки входу, включаючи багатофакторну автентифікацію та зовнішню аутентифікацію за допомогою Google, Twitter тощо.
- Розширення редактора, що забезпечують виділення синтаксису, доповнення коду та інші функції спеціально для розробки веб-сторінок.

Крім того, існує також Entity Framework, який реалізовує технології ORM (object-relational mapping). Основна перевага даного підходу в тому, що використання ORM автоматично зв’язує класи з таблицями в базі даних, що в свою чергу зменшує витрати в часі на розробку та конфігурацію розроблюваного застосунку.

Розглянемо мову програмування та її фреймворки, які є аналогічними до розглянутої вище.

Java є широко використовуваною мовою програмування для кодування веб-додатків. Він був популярним вибором серед розробників понад два десятиліття, сьогодні використовуються мільйонами сервысів. Java — це багатоплатформна, об’єктно-орієнтована та мережево-орієнтована мова, яку можна використовувати як саму платформу. Це швидка, безпечна та надійна мова програмування для кодування всього, від мобільних додатків і корпоративного програмного забезпечення до додатків для великих даних і технологій на стороні сервера.

Оскільки Java є вільною та універсальною мовою, вона створює локалізоване та розподілене програмне забезпечення. Деякі поширені сфери використання Java включають:

- Розробка ігор: багато популярних мобільних, комп’ютерних і відеоігор створено на Java. Навіть сучасні ігри, які інтегрують передові технології, такі як машинне навчання або віртуальна реальність, створені за допомогою технології Java.
- Хмарні обчислення: Java часто називають WORA (Write Once Run Anywhere) – пишіть один раз і запускайте будь-де, що робить її ідеальною для децентралізованих хмарних програм. Хмарні постачальники вибирають мову Java для запуску програм на широкому діапазоні базових платформ.
- Великі дані: Java використовується для механізмів обробки даних, які можуть працювати зі складними наборами даних і величезними обсягами даних у реальному часі.
- Штучний інтелект: Java — це потужний центр бібліотек машинного навчання. Його стабільність і швидкість роблять його ідеальним для розробки додатків зі штучним інтелектом, таких як обробка природної мови та глибоке навчання.
- Інтернет речей: Java використовувалася для програмування веб сервісів для продажу будь чого.

Мова програмування Java не обмежується стандартними бібліотеками та методами для розробки програмного забезпечення. Стек технологій даної мови програмування поділяють на: SE (Standart Edition) та EE (Enterprice Edition). Для корпоративної розробки великих систем зазвичай обирають саме Enterprise Java, адже спектр технологій яке пропонує ця версія задовольняє майже всі сучасні потреби систем різного рівня.

Взагалі Java EE це лише набір стандартних специфікацій для розробки корпоративних застосунків. Конкретні реалізації даних специфікацій містяться в фреймворках Java. Найбільш популярним корпоративним фреймворком Java, на сьогоднішній день є Spring. На рисунку 2.2 зображено результати дослідження компанії Snyk [12] щодо популярності корпоративних фреймворків Java.

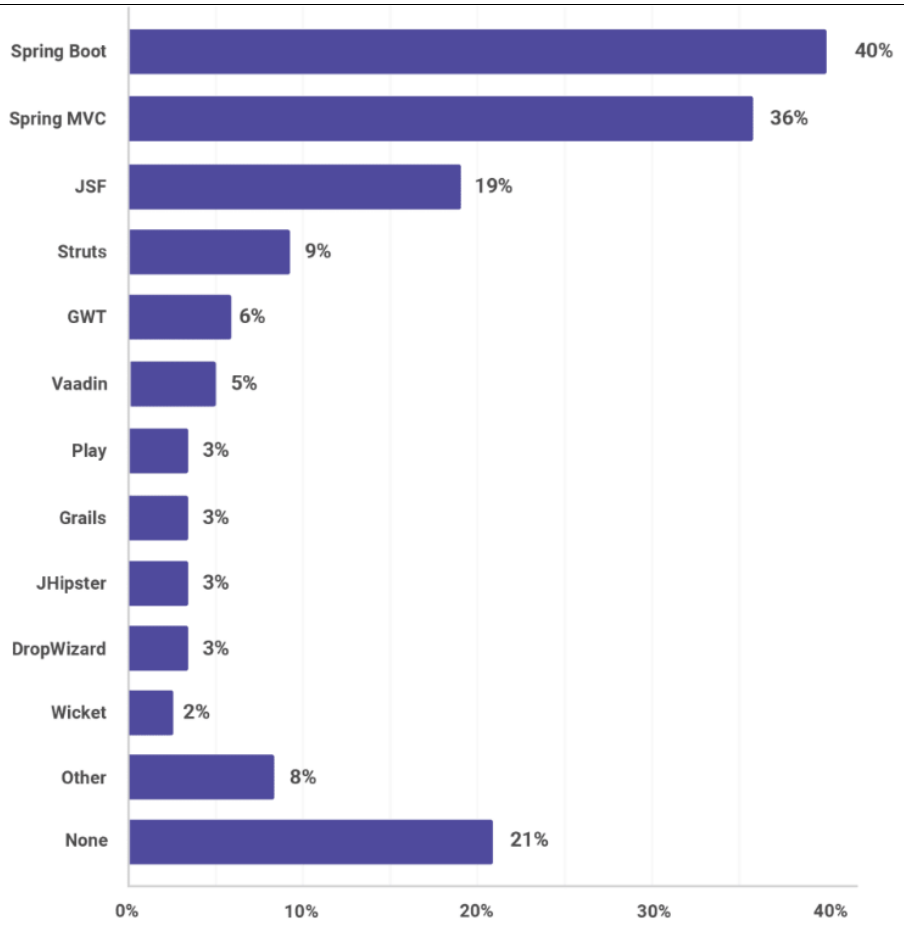

Рисунок 2.2 – Популярність фреймворків Java

Spring Framework надає комплексну модель програмування та конфігурації для сучасних корпоративних програм на базі Java - на будь-якій платформі розгортання.

Ключовим елементом Spring є інфраструктурна підтримка на рівні програми: Spring зосереджується на «підготовці» корпоративних додатків, щоб розробники могли зосередитися на бізнес-логіці на рівні програми без зайвих зв’язків із конкретними середовищами розгортання.

Spring надає такі можливості для розробки застосунків:

– Основні технології: впровадження залежностей, події, ресурси, i18n, перевірка, прив’язка даних, перетворення типів, SpEL, AOP.
– Тестування: фіктивні об’єкти, фреймворк TestContext, Spring MVC Test, WebTestClient.
– Доступ до даних: транзакції, підтримка DAO, JDBC, ORM, маршалінг XML.

- Веб-фреймворки Spring MVC і Spring WebFlux.
- Інтеграція: віддалений доступ, JMS, JCA, JMX, електронна пошта, завдання, планування, кеш.

Основним недоліком технологій на основі Java вважають їх надмірну надлишковість. Це витікає з того. Що основним принципом Java э зворотня сумісність коду. Код написаний на попередній версії Java буде так само працювати і на наступних версіях.

Невід’ємною частиною сучасної системи є інтеграція з базами даних. Найбільшою популярністю на сьогоднішній день користуються реляційні бази даних. Завдяки гнучкості реалізації та загальній структурі даних технологій можливо посвністю задовольнити потреби сучасних корпоративних додатків. На рисунку 2.4 представлено результати опитувань тієї ж компанії Snyk щодо того, які СКБД розробники найчастіше використовують для розробки сучасних корпоративних систем на мові програмування Java.

З розвитком комп’ютерних мереж, протягом останнього десятиліття інтернет став невід’ємною частиною життя кожного. Компанії чи інституції різного рівня так чи інакше мають свою інтернет сторінку в всесвітній павутині.

Як видно на діаграмі від агенства Statista [13] (рисунок 2.3), до 2025 майже 76 мільярдів пристроїв буде підключений до Інтернету.

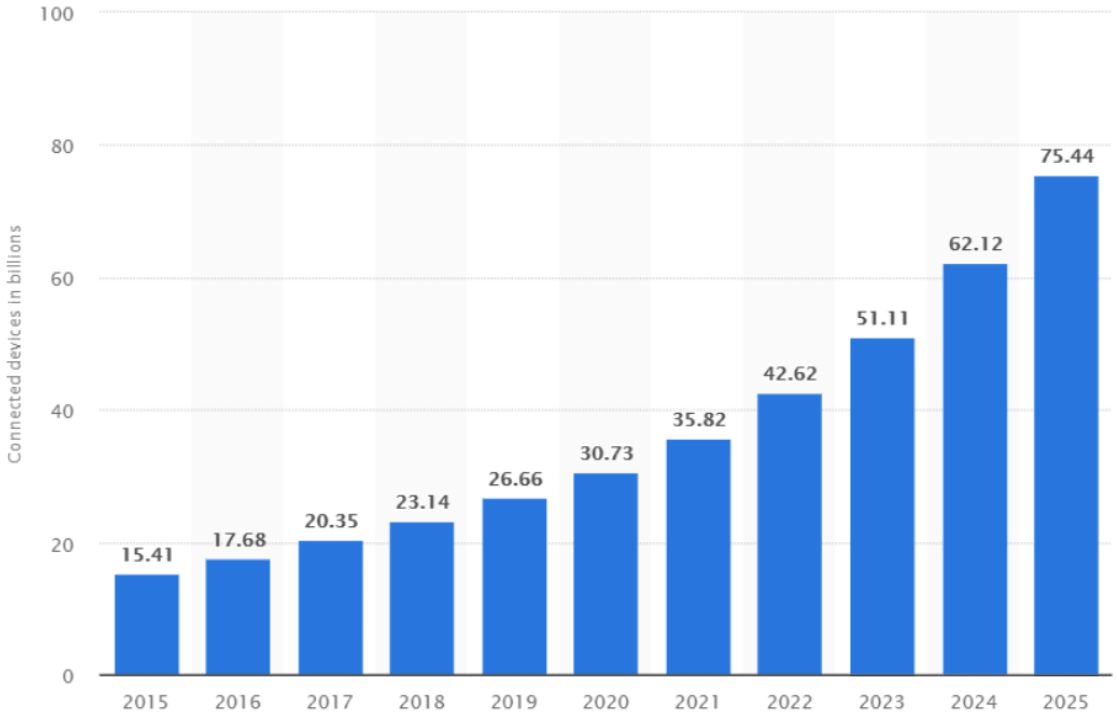

Рисунок 2.3 – практична та прогнозована кількість девайсів підключених до мережі інтернет

Це дуже впливає на подальші тенденції веб-розробки. Адже зі зростанням цього показника збільшується ефективність синхронізації девайсів.

Сьогодні не так багато сайтів, смартфонів та додатків мають інтеграцію так званого інтернету речей. Серед них можна виділити Google Home та Uber. Наприклад, можна викликати машину через програму, а підтвердити завершення поїздки голосовими командами після прибуття додому.

Чим більше користувачів цифрових пристроїв мають доступ до мережі інтернет, тим більше зростає попит на сучасні веб технології. Найбільш популярною технологією передачі даних через інтернет є протокол HTTP.

HTTP (HyperText Transfer Protocol) – це протокол, який дозволяє отримувати різні ресурси, наприклад HTML-документи. Протокол HTTP є основою обміну даними в Інтернеті. HTTP є протоколом клієнт-серверної взаємодії, що означає ініціювання запитів до сервера самим одержувачем, як правило, веб-браузером (web-browser). Отриманий підсумковий документ (може) складатися з різних піддокументів, що є частиною підсумкового документа: наприклад, з окремо

отриманого тексту, опис структури документа, зображень, відео-файлів, скриптів та багато іншого[14].

На основі даного протоколу було засновано такі сучасні архітектурні підходи для створення програмних застосунків як REST та SOAP. Завдяки фунціям, які надає протокол HTTP реалізація та використання застосунків на основі цих архітектурних підходів є стандартизованою та доволі простою, в залежності від розроблюваної системи.

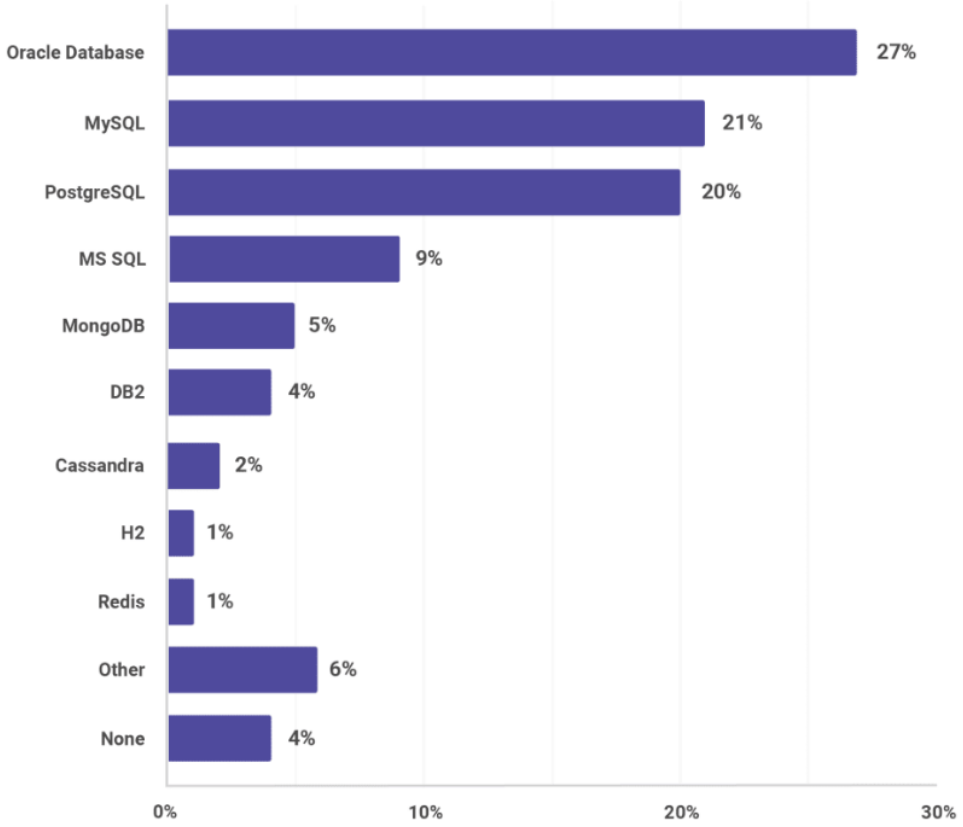

Рисунок 2.4 – Популярність різних СКБД в Java застосунках

З діаграми видно, що в трійку найбільш популярних СКБД входять Oracle, MySQL та PostgreSQL. Розглянемо що з себе представляють реляційні бази даних на основі PostgreSQL.

PostgreSQL — це сучасна реляційна база даних корпоративного класу з відкритим кодом, яка підтримує запити як SQL (реляційні), так і JSON (нереляційні). Це високостабільна система керування базами даних, що підтримується більш ніж 20-річним розвитком спільноти, що сприяло її високому рівню стійкості, цілісності та коректності. PostgreSQL використовується як основне

сховище даних або сховище даних для багатьох веб-, мобільних, геопросторових та аналітичних додатків. Остання основна версія — PostgreSQL 12.

PostgreSQL має багату історію підтримки розширених типів даних і підтримує рівень оптимізації продуктивності, який є звичайним для його комерційних аналогів баз даних, таких як Oracle і SQL Server.

Таблиці даних, які використовуються в реляційній базі даних, зберігають інформацію про пов’язані об’єкти. Кожен рядок містить запис з унікальним ідентифікатором, відомим як ключ, і кожен стовпець містить атрибути даних. Кожен запис призначає значення кожній ознаці, завдяки чому зв’язки між точками даних легко ідентифікувати.

Стандартним інтерфейсом користувача та прикладної програми (API) реляційної бази даних є мова структурованих запитів. Оператори коду SQL використовуються як для інтерактивних запитів на інформацію з реляційної бази даних, так і для збору даних для звітів. Для забезпечення точності та доступності реляційної бази даних необхідно дотримуватися визначених правил цілісності даних.

Кожна таблиця, яку іноді називають відношенням, у реляційній базі даних містить одну або кілька категорій даних у стовпцях або атрибутах. Кожен рядок, також званий записом або кортежем, містить унікальний екземпляр даних - або ключ - для категорій, визначених стовпцями. Кожна таблиця має унікальний первинний ключ, який ідентифікує інформацію в таблиці. Зв'язок між таблицями можна встановити за допомогою зовнішніх ключів - поля в таблиці, яке посилається на первинний ключ іншої таблиці.

2.2 Вибір технологій для розробки прикладного програмного інтерфейсу

Провівши аналіз предметної сфери, визначивши функціональні вимоги до прикладного програмного інтерфейсу, зробивши висновки щодо сучасних технологій для розробки систем корпоративного рівня можна перейти до використання даних технологій в розробці системи.

В якості мови програмування для розробки програмного інтерфейсу було обрано Java. Це обумовлено тим, що Java та її фреймворки надають широкий спектр методів для вирішення поставленої задачі.

Технологією для зберігання, опису та передачі через мережу метаданих об’єктних моделей було обрано XML. XML - це мова розмітки подібна до HTML. Розшифровується як (англ. Extensible Markup Language - Розширювана Мова Розмітки) і є рекомендацією спільноти W3C [15] як мова розмітки загального призначення. На відміну від інших мов розмітки, XML сам по собі не визначений (це означає, що ви повинні самі визначати теги, що використовуються). Це досягається за допомогою створення так званої схеми документу XSD (XML Schema Definition). Вона задає стандарти для описання такого XML документу, який повинен выдповыдати вимогам розробника.

Для опису шаблонів та створення каркасів вихідного коду об’єктних моделей було використано Apache Freemarker. Apache FreeMarker — це механізм шаблонів: бібліотека Java для створення текстового виводу (веб-сторінки HTML, електронні листи, файли конфігурації, вихідний код тощо) на основі шаблонів та змінених даних. Шаблони написані мовою шаблонів FreeMarker (FTL), яка є простою спеціалізованою мовою (а не повноцінною мовою програмування, як PHP). Зазвичай для підготовки даних використовується мова програмування загального призначення (наприклад, Java) (видача запитів до бази даних, виконання бізнес-розрахунків). Потім Apache FreeMarker відображає підготовлені дані за допомогою шаблонів. У шаблоні ви зосереджуєтесь на тому, як представити дані, а поза шаблоном ви зосереджуєтесь на тому, які дані представити. [16] На рисунку 2.5 зображено загальну схему того, яку функці. Виконує даний фреймворк.

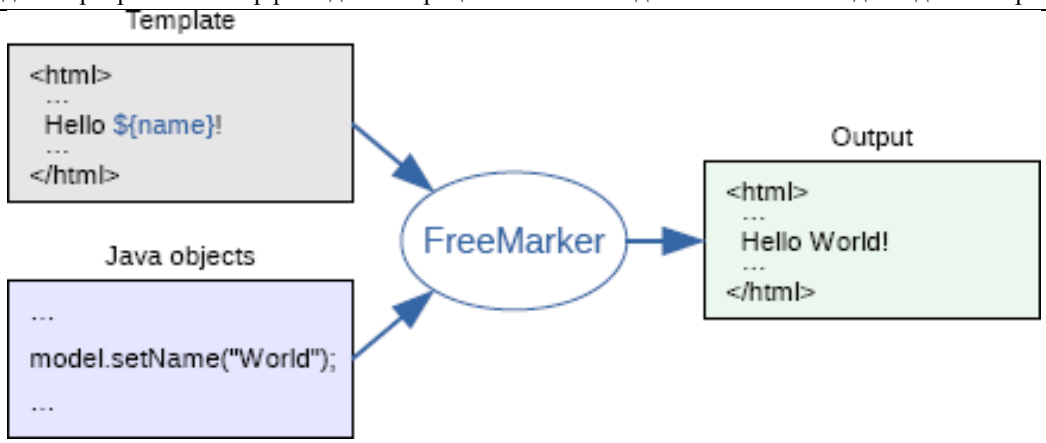

Рисунок 2.5 – Схема роботи шаблонізатора Freemarker

В якості системи керування базами даних було обрано PostgreSQL. Це обумовлено тим, що дана система розповсюджується за відкритою ліцензією та не вимагає встановлення додаткових модулів та бібліотек.

Для налагодження доступу до даних через Java код, до системи було додано популярний Java фреймворк Hibernate. Hibernate ORM дозволяє розробникам легше писати програми, дані яких перебувають у процесі застосування. Як фреймворк об’єктно-реляційного відображення (ORM), Hibernate займається збереженням даних, оскільки він застосовується до реляційних баз даних (через JDBC). На додаток до власного API, Hibernate також є реалізацією специфікації Java Persistence API (JPA). Таким чином, його можна легко використовувати в будь-якому середовищі, що підтримує JPA, включаючи програми Java SE, сервери додатків Java EE тощо.

З метою реалізації доступу до даних через мережевий протокол HTTP, в якості фреймворку для реалізації REST API було використано технології які надає фреймворк Spring, а саме такі його компоненти:

- Spring Boot;
- Spring Data JPA;
- Spring MVC;
- Spring Web Services.

Висновки до розділу 2

В даному розділі було розглянуто основні технології, які можливо використати в процесі реалізації прикладного програмного інтерфейсу для генерації об’єктних моделей. В процесі аналізу було визначено найпопулярніші технології, бібліотеки та фреймворки для реалізації системи. На основі даних досліджень різних компаній та агентств було визначено, які технології використовуються розробниками в системах корпоративного рівня.

В результаті дослідження, проведеного в даному розділі було визначено стек технологій для розробки прикладного програмного інтерфейсу. В якості мови програмування було обрано мову Java.

Крім того, для розробки було обрано наступні технології та фреймворки для реалізації системи:

- СКБД PostgreSQL;
- Spring Framework з наступними компонентами: Spring Core, Spring MVC, Spring Data JPA, Spring Boot та Spring Web Services;
- ORM фреймворк Hibernate
- шаблонізатор Apache FreeMarker
- розширювана мова розмітки XML.

3 МОДЕЛЮВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ ДЛЯ ГЕНЕРАЦІ ОБ’ЄКТНИХ МОДЕЛЕЙ

3.1 Проєктування структури прикладного програмного інтерфейсу

Структурно прикладний програмний інтерфейс для генерації об’єктних моделей можна поділити на кілька основних модулів:

- модуль для введення та зчитування файлу з метаданими;
- модуль для генерації файлів з вихідним кодом;
- модуль компоновки та компіляції та виконання файлів з кодом.

Перш за все, необхідно визначитись зі структурою файлу з метаданими, які будуть використані в якості вхідних даних для генерації об’єктних моделей. Як було визначено в розділі 2 даної роботи, в якості вхідних даних об’єктних моделей необхідно використати XML структуру. Для забезпечення вірності та стандартизованості введення метаданих об’єктних моделей найкраще підходить так звана XML Schema Definition або XSD.

XSD — це мова опису структури документа XML. Його також називають XML Schema. При використанні XML Schema XML парсер може перевірити не тільки правильність синтаксису XML документа, але також його структуру, модель змісту та типи даних [17].

Такий підхід дозволяє об'єктно-орієнтованим мовам програмування легко створювати об'єкти в пам'яті, що, безперечно, зручніше, ніж розбирати XML як звичайний текстовий файл.

Крім того, XSD може бути легко розширений і дозволяє підключати готові словники для опису типових завдань, наприклад веб-сервісів, таких як SOAP.

Файл з метаданими можна розбити на три основні частини:

- безпосередньо, дані про об’єктні моделі;
- дані про перелічення (enumerations);
- дані про асоціації між об’єктними моделями.

В частині файлу з описом об’єктних моделей визначено список моделей, які будуть генеруватись, їх основні атрибути та характеристики. На рисунку 3.1 зображено структуру частини з об’єктними моделями в вигляді дерева.

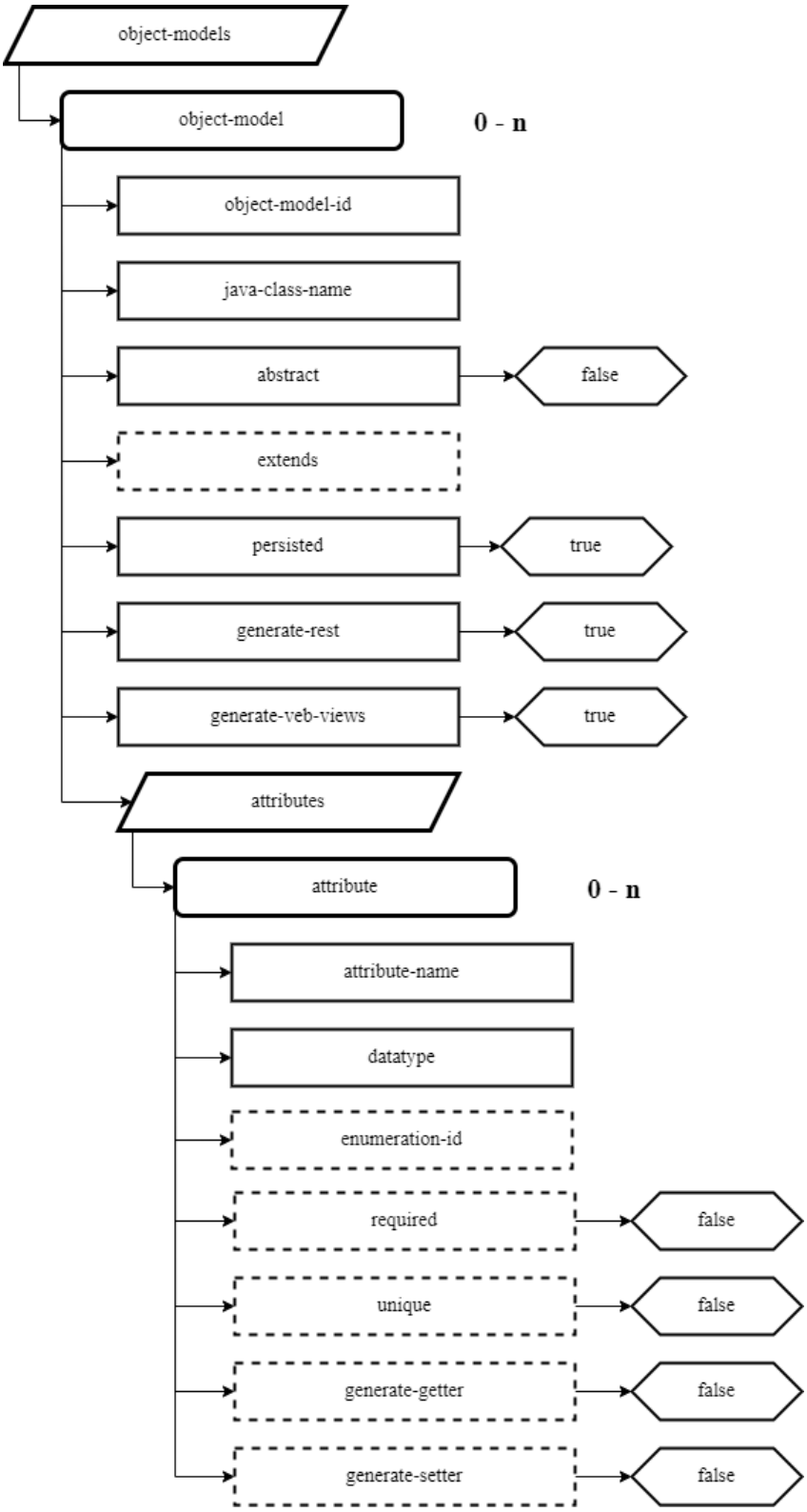

Рисунок 3.1 – Структура частини метаданих з описом об’єктних моделей

На попередньому рисунку елементи в вигляді паралелограма – це списки з комплексними типами. Елемент object-models містить в собі колекцію комплексних типів object-model. В свою чергу, елемент object-model містить в собі велику кількість різних даних. В таблиці 3.1 перелічені усі елементи, їх призначення та значення за замовчуванням.

Таблиця 3.1 – Перелік елементів та їх призначення

№	Назва	Призначення	Значення за замовчуванням	Чи є атрибут обов’язковим?
1	object-attribute-name-id	Символьний ідентифікатор конкретної об’єктної моделі	-	Так
2	java-class-name	Назва класу об’єктної моделі, яка буде використана в ході генерації	-	Так
3	abstract	Чи є об’єктна модель абстрактною?	false	Так
4	extends	Ідентифікатор іншої об’єктної моделі, від якої поточна унаслідується	-	Ні
5	persisted	Чи зберігатиметься об’єктна модель в базі даних?	true	Так
6	generate-rest	Чи будуть генеруватися REST веб-сервіси для даної моделі?	true	Так
7	generate-web-views	Чи будуть генеруватися веб-сторынки з таблицями для даної моделі?	true	Так

Крім того, в елементі object-model міститься підсписок з основними атрибутами даної моделі. В таблиці 3.2 також представлено перелік основних піделементів та їх призначення.

Таблиця 3.2 – Перелік елементів типу attribute та їх призначення

№	Назва	Призначення	Значення за замовчуванням	Чи є атрибут обов’язковим?
1	attribute-name	Символьний ідентифікатор конкретного атрибуту	-	Так
2	datatype	Тип даних поточного атрибуту	-	Так
3	enumeration-id	Ідентифікатор переліченння, яке використовується в якості datatype	-	Ні
4	required	Чи є атрибут обов’язковим?	false	Ні
5	unique	Чи є атрибут унікальним?	false	Ні
6	generate-getter	Чи буде генеруватися метод get?	true	Ні
7	generate-setter	Чи буде генеруватися метод set?	true	Ні

Ще одна частина файлу з об'єктними моделями містить дані про переліченння, які використовуються в об’єктних моделях. На рисунку 3.2 зображено структуру опису перелічень в вигляді дерева.

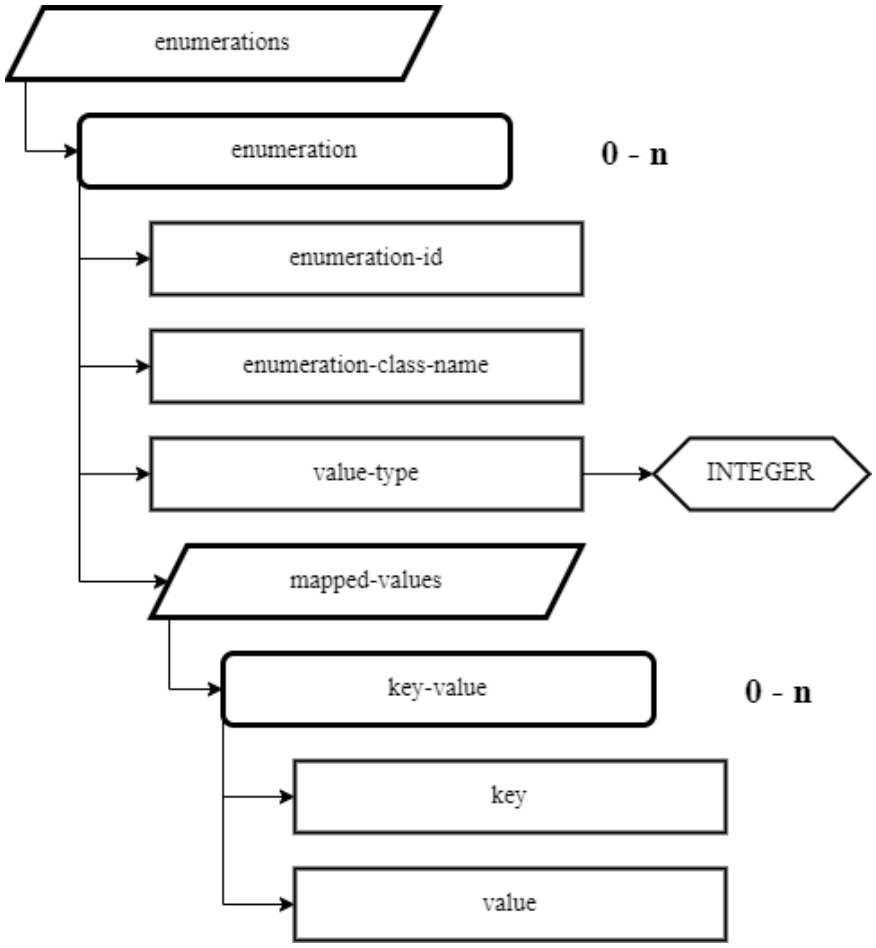

Рисунок 3.2 – Структура частини метаданих з описом перелічень моделей

За своєю структурою перелічення є набагато простішим типом, тому і елементів для їх опису набагато менше. В таблиці 3.3 наведено перелік основних елементів.

Таблиця 3.3 – Перелік елементів типу enumeration та їх призначення

№	Назва	Призначення	Значення за замовчуванням	Чи є атрибут обов’язковим?
1	Enumeration-id	Символьний ідентифікатор перелічення	-	Так

Продовження таблиці 3.3

2	enumeration-class-name	Ім’я класу перелічення	-	Так
3	value-type	Тип даних перелічення (рядок або ціле число)	INTEGER	Так
4	mapped-values	Список зі значеннями перелічень	-	-
5	key-value	Конкретне значення перелічення	-	-
6	key	Значення-ключ перелічення	-	Так
7	value	Конкретне значення перелічення	-	Так

Наступною і останньою частиною файлу з об’єктною моделлю є частина з асоціаціями між різними об’єктами. Асоціації можуть бути різного типу, наприклад, один-до-одного, багато-до-одного, один-до-багатьох, багато-до-багатьох. На рисунку 3.3 наведено приклад деревовидної структури частини файлу з асоціаціями.

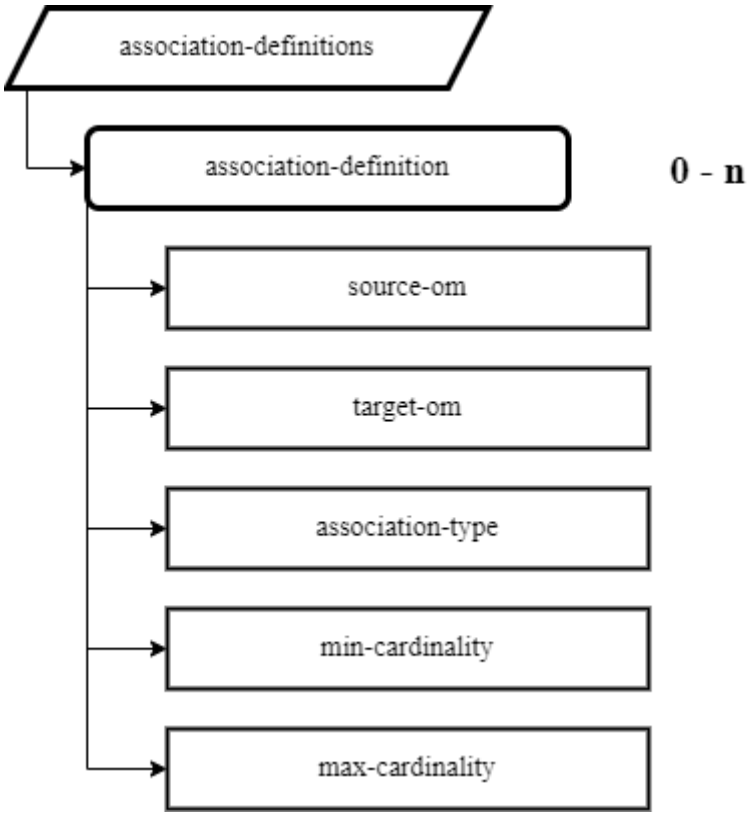

Рисунок 3.3 – Структура частини метаданих з описом асоціацій моделей

Асоціації не містять в собі ніяких списків, лише невелику кількість даних про тип асоціації та об’єктні моделі, на які розповсюджується дані асоціації. В таблиці 3.4 представлено дані про кожен елемент даної частини об’єктної моделі.

Таблиця 3.4 – Дані про елементи метаданих з асоціаціями

№	Назва	Призначення	Значення за замовчуванням	Чи є атрибут обов’язковим?
1	source-om	Об’єктна модель, яка є джерелом асоціації	-	Так
2	target-om	Об’єктна модель, до якої створена асоціація	-	Так
3	association-type	Тип асоціації між об’єктними моделями	-	Так

Продовження таблиці 3.4.

4	min-cardinality	Мінімальна кількість об’єктів, які можуть бути асоційовані до об’єкту-джерела	0	Так
5	max-cardinality	Максимальна кількість об’єктів, які можуть бути асоційовані до об’єкту-джерела	Приблизно 10^{18}	Так

Для забезпечення валідності та цілісності даних генерація об’єктних моделей повинна проводитися в певному порядку. Виходячи з наведених вище даних та таблиць можемо дійти висновку, що генерація об’єктних моделей повинна проводитися в наступній послідовності:

- спочатку генеруються незалежні ні від чого метадані перелічень, адже вони не потребують ніяких додаткових даних;
- наступним етапом генеруються, безпосередньо об’єктні моделі. Це обумовлено тим, що в якості типів даних атрибутів об’єктних моделей можуть бути використані заздалегідь згенеровані перелічення;
- в останню чергу генеруються асоціації, адже для забезпечення умов генерації необхідно щоб класи об’єктних моделей вже були згенеровані.

На наступному рисунку 3.4 зображено блок-схему послідовності роботи механізму зчитування та обробки даних про об’єктні моделі.

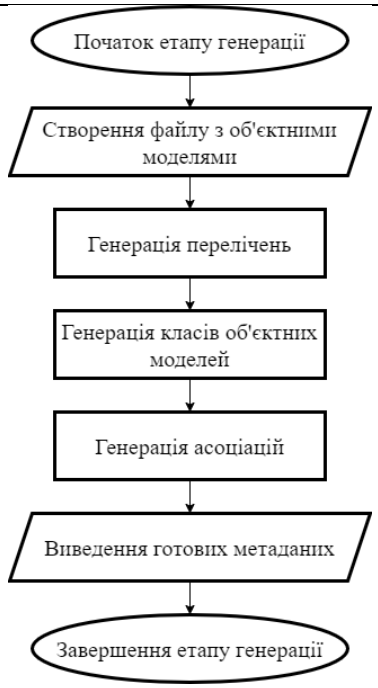

Рисунок 3.4 – Блок-схема процесу генерації метаданих

Визначившись зі структурою файлу, необхідно перенести дані з файлу в пам’ять комп’ютера для подальших процесів. Однією з головних переваг структурованого XML файлу є те, що всі елементи упорядковані за так званою DOM (Document Object Model) [18]. На рисунку 3.5 представлено схему DOM для простого XML документу.

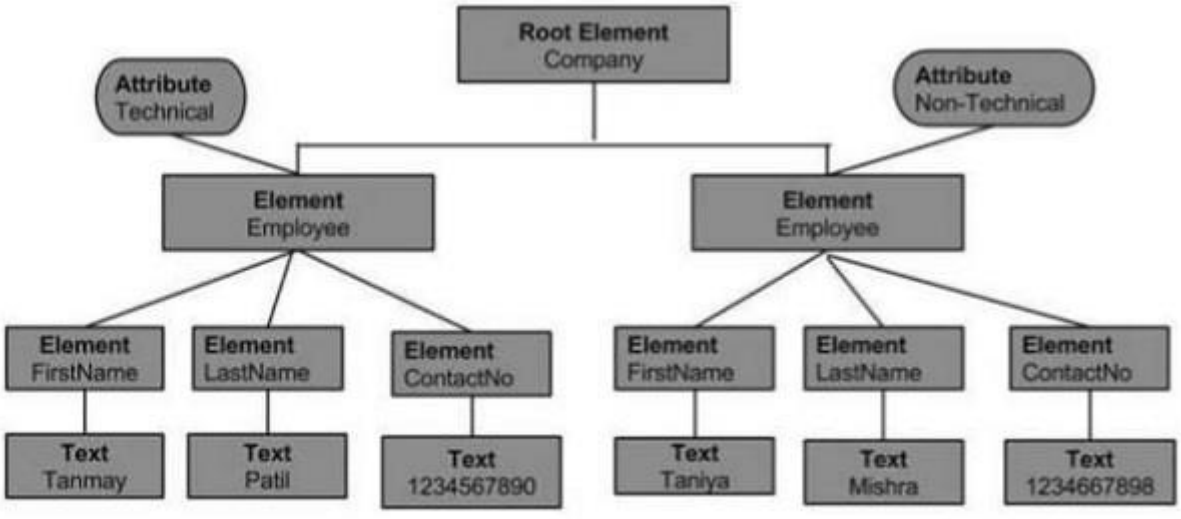

Рисунок 3.5 – Типова структура документу за DOM

Це дозволяє доволі легко зчитувати всі дані з різних елементів програмно та заносити ці дані в пам’ять для подальших обчислень. Єдиною проблемою при

цьому залишається те, що всі дані при даному підході зберігаються в вигляді текстових рядків, тому їх необхідно додатково приводити до необхідних примітивних типів.

Наступним етапом розробки прикладного програмного інтерфейсу є створення шаблонів для класів, сервісів та скриптів для доступу до даних з реляційних баз даних. Як було визначено в ході розділу 2, для створення шаблонів використано фреймворк з шаблонізатором FreeMarker. Як було зазначено раніше, генерація вихідного коду буде також проходити в декілька етапів.

В ході першого етапу повинні генеруватися класи об’єктних моделей з відповідними таблицями для реляційних баз даних. Завдяки гнучкій конфігурації обраних фреймворків Hibernate та Spring даний етап можна скоротити лише до генерації класів.

Шаблони класів завдяки вбудованим функціям фреймворку FreeMarker можна розбити на декілька частин. В першому шаблоні міститимуться дані про клас, його назва, атрибути та основні складові такі як пакет, в якому він міститься та імпорт залежностей з інших класів. В другій частині міститимуться дані про поля конкретного класу, конструктори, методи геттери та сеттери. Третя частина описуватиме залежності з асоціаціями, а саме види асоціацій та класи, які мають асоціації до даної об’єктної моделі.

Наступним етапом генеруватимуться шаблони з класами-сервісами для доступу до даних, які зберігаються на сервері бази даних. Такі сервіси представляють собою класи, які містять в собі основні об’єкти для доступу та з’єднання до бази та основні методи для керування даними таблиці. До таких методів входять наступні:

- методи створення нового екземпляру (create);
- методи зчитування даних з таблиці (read);
- методи редагування існуючих екземплярів (update);
- методи видалення даних з таблиці (delete).

Передостаннім етапом генеруватимуться веб сервіси для доступу до даних через протокол HTTP. Шаблони таких сервісів представляють собою класи,

помічені відповідними метатегами, які містять в собі посилання на сервіс для доступу до даних з бази конкретної об’єктної моделі та методи для редагування та зчитування цих даних. Як і в попередньому етапі ці методи відповідають методам CRUD, але з поправкою на методи, які використовуються в протоколі HTTP: GET (зчитати дані), POST (додати дані), PATCH (оновити дані), DELETE (видалити дані) [19].

Останнім типом шаблонів є шаблони веб-сторінок з таблицями для відображення та проведення операцій над даними в звичному для користувачів вигляді. Дані шаблони складатимуться з файлів гіпертекстової розмітки (HTML) та каскадних таблиць стилів (CSS).

Окремо варто зазначити, що в ході генерації також повинні створюватись шаблони, які міститимуть дані про проєкт. Це необхідно для того, щоб при подальшій компіляції не виникало труднощів. Серед таких файлів повинні бути файли з залежностями проєкту, тобто, з залежностями на фреймворки які необхідні для функціонування системи, файлами з властивостями проєкту для налаштування доступу до бази даних та конфігурації модулів програмного інтерфейсу.

Підбиваючи підсумок, можемо сформувати загальну схему шаблонів та компонентів програмного інтерфейсу, які будуть генеруватися на їх основі. На рисунку 3.6 представлено дану схему.

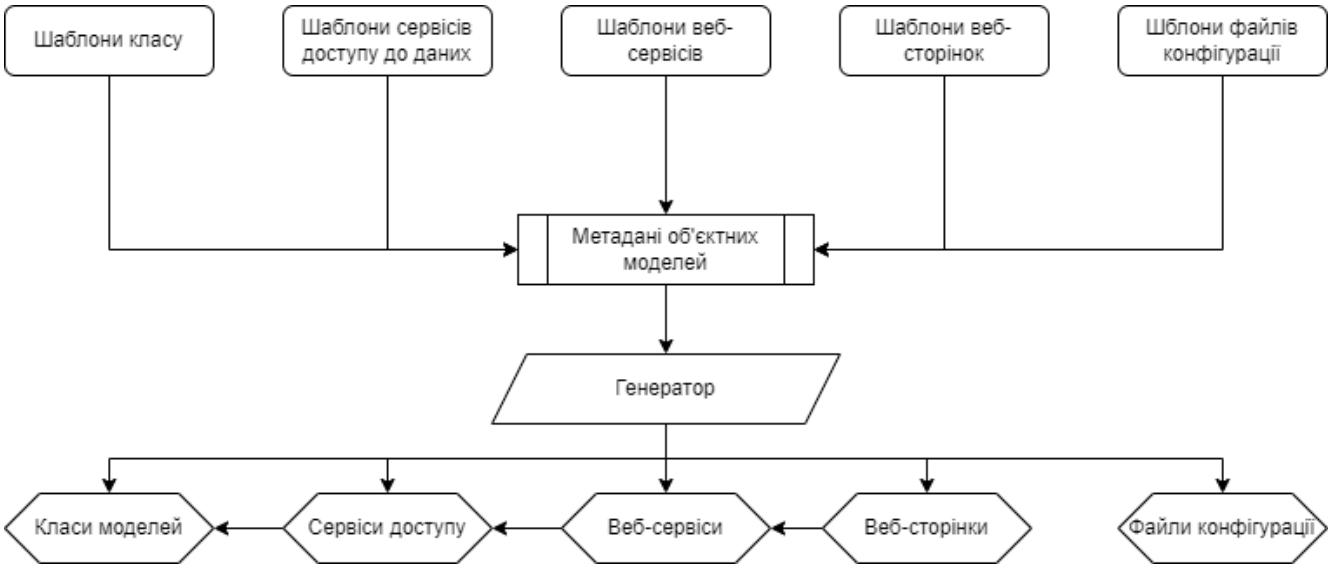

Рисунок 3.6 – Схема перетворення шаблонів в файли з вихідним кодом та залежності між компонентами згенерованого коду

Після того, як відпрацює генератор, на останньому етапі відбуватиметься компоновка усіх згенерованих файлів у архів з проєктом. Завдяки функціям Java для роботи з файловою системою це досягається доволі просто. На етапі конфігурації генератора користувач повинен мати вибір: генерувати готовий до запуску на сервері сервіс чи завантажити архів з вихідним кодом та файлами конфігурації.

У випадку вибору опції генерації готового сервісу, програмний інтерфейс повинен самостійно запустити процес компіляції та збору файлів для запуску на серверній платформі. Дану функцію можна реалізувати за допомогою так званих інструментів збірки, таких як: Maven, Gradle або Ant.

3.2 Програмна реалізація прикладного програмного інтерфейсу

Реалізація прикладного програмного інтерфейсу проводитиметься в кілька етапів, так само, як зазначено в попередньому підрозділі. Першим модулем, який підлягає реалізації є структурний опис файлу з метаданими об’єктних моделей.

Як було зазначено раніше, для забезпечення стандартизованості та валідності введених метаданих необхідно визначити XML схему файлу з метаданими. Файл складається з кількох підчастин, які відповідають за різні елементи об’єктної моделі: перелічення, визначення, безпосередньо, об’єктних моделей та асоціації. В додатку А представлено файл XML схеми, розроблений згідно специфікаціям з таблиць 3.1 – 3.5.

Для того щоб користувач міг скористатися даною схемою для опису об’єктних моделей він повинен завантажити фай зі схемою, або вказати посилання в спеціальному тезі при створенні файлу з об’єктними моделями.

Тепер необхідно реалізувати обробку файлу з метаданими об’єктних моделей в генераторі. Для досягенння цієї цілі найкраще підходить вбудований пакет Java для роботи з документами XML – SAX API. Завдяки методам та класам які надає дана бібліотека можна легко пройтися по структурі документу та зчитати дані з нього.

Наприклад, для зчитування списку елементів необхідно звернутись до кореню документу і потім знайти елемент за назвою тегу. Отримавши список, так

само можна звернутися до дочірніх елементів поточного елементу і, відповідно, отримати необхідні текстові дані. В лістингу 3.1 наведено приклад обходу елементів XML документу з метаданими для зчитування даних про перелічення.

Лістинг 3.1 – приклад обходу дерева XML документу за допомогою SAX API

```
private List<EnumerationMetadata> parseEnums(Document document){
    List<EnumerationMetadata> enumerationMetadataList = new ArrayList<>();
    var nodes = document.getElementsByTagName("enumeration");
    for(int i = 0; i < nodes.getLength(); i++){
        Node node = nodes.item(i);
        if(node.getNodeType() == Node.ELEMENT_NODE){
            Element element = (Element)node;
            String id = element.getElementsByTagName("enumeration-
id").item(0).getTextContent();
            String name = element.getElementsByTagName("enumeration-class-
name").item(0).getTextContent();
            String valueType = element.getElementsByTagName("value-
type").item(0).getTextContent();
            EnumerationMetadata metadata = new EnumerationMetadata(id, name,
valueType);
            var values = element.getElementsByTagName("value");
            for(int j = 0; j < values.getLength(); j++){
                Node valueNode = values.item(j);
                if(valueNode.getNodeType() == Node.ELEMENT_NODE){
                    Element elementMap = (Element)valueNode;
                    String key =
elementMap.getElementsByTagName("key").item(0).getTextContent();
                    String val = elementMap.getElementsByTagName("mapped-
value").item(0).getTextContent();
                    metadata.putValue(key, val);
                }
            }
            enumerationMetadataList.add(metadata);
        }
    }
    return enumerationMetadataList;
}
```

За таким методом можна зчитати всі метадані про об’єктні моделі, занести їх до відповідного об’єкту класу типу CommonObjectMetadata та використати для подальшої обробки в шаблонах.

Неступним етапом необхідно описати шаблони, на основі яких будуть генеруватися класи та веб-сервіси. В лістингу 3.2 приведено приклад шаблону для генерації типів перелічень на основі метаданих, які були отримані на попередньому етапі.

Лістинг 3.2 – Шаблон для генерації перелічень

```
public enum ${enumeration.className} {
    <#if enumeration.valueType=="DEFAULT">
        <#list enumeration.values as key, value>
            #{key},
        </#list>;
    </#if>
```

Продовження лістингу 3.2

```
    <#if enumeration.valueType=="STRING">
        <#list enumeration.values as key, value>
            ${key}("${value}"),
        </#list>
        ;

        private ${enumeration.className}(<#if
enumeration.valueType=="DEFAULT">Integer<#else>String</#if> value) {
            this.value = value;
        }

        private String value;

        public String getValue(){
            return this.value;
        }
    </#if>
}
```

За допомогою спеціальних директив та тегів, можна гнучко описати шаблон для генерації різних частин та модулів сервісу. Наприклад, з використанням тегу <#list> можна перебрати колекцію атрибутів да записати їх в клас в якості полів. За допомогою директиви <#if> можна перевірити деяку умову, від виконання якої буде залежати те, чи буде конкретний шматок шаблону занесено в вихідний файл.

Після опису структури шаблонів, наступним кроком є об’єднання їх з метаданими з XML файлу. Це реалізується доволі просто: в першу чергу необхідно зчитати шаблони з файлової системи проєкту та створити об’єкт класу Template, який на вхід отримує файли шаблонів, конфігурацію та модель, яка буде підсталятися в якості значень, зазначених в директивах даного шаблону. В лістингу 3.3 наведено приклад виконання роботи даного шаблонізатора.

Лістинг 3.3– Згенерований вихідний код на основі шаблону класу

```
@Entity
@Table(name = "test")
public class Test {
    @Id
    @GeneratedValue
    @Column(name = "personId")
    private long personId;
```

Продовження лістингу 3.3

```
    @Column(name = "name")
    private String name;

    public String getName(){
        return this.name;
    }
    public void setName(String name){
        this.name = name;
    }

    @Column(name = "age")
    private int age;

    public void setAge(int age){
        this.age = age;
    }

    @Column(name = "cash")
    private float cash;

    public float getCash(){
        return this.cash;
    }

    public Test(String name, int age, float cash){
        this.name = name;
        this.age = age;
        this.cash = cash;
    }
}
```

Таким самим чином генеруються сервіси для доступу до бази даних, веб-сервіси на основі REST та веб сторінки. Вихідний код веб-сервісу для доступу до даних через протокол HTTP виглядає таким чином, як зображено в додатку Б.

Також в процесі генерації, як було зазначено в попередньому розділі, генеруватимуться файли для конфігурації сервісу. В якості цих файлів виступають pom.xml та application.properties.

Файл pom.xml містить усі основні дані про проєкт, його власника тощо. Крім того, в ньому зберігаються дані про залежності до інших бібліотек чи фреймворків. В додатку В зображено даний файл зі всіма необхідними залежностями до проєкту.

В файлі конфігурації application.properties зберігаються властивості, які використовує фреймворк Spring. До таких даних відносяться реквізити для доступу до сервера бази даних, конфігураційні властивості для різних рівнів логування, властивості для налаштування поведінки ORM фреймворку Hibernate тощо. Структура даного файлу властивостей представлена в додатку Г.

Останнім модулем програми є модуль-компонувальник. Його основна мета – розподілити файли проєкту в правильну структуру та помістити готові файли та папки в архів з проєктом. Даний модуль було реалізовано за допомогою вбудованих функцій в стандартну бібліотеку Java для роботи з файловою системою. Результатом роботи даного модуля є структура проєкту, зображена на рисунку 3.7.

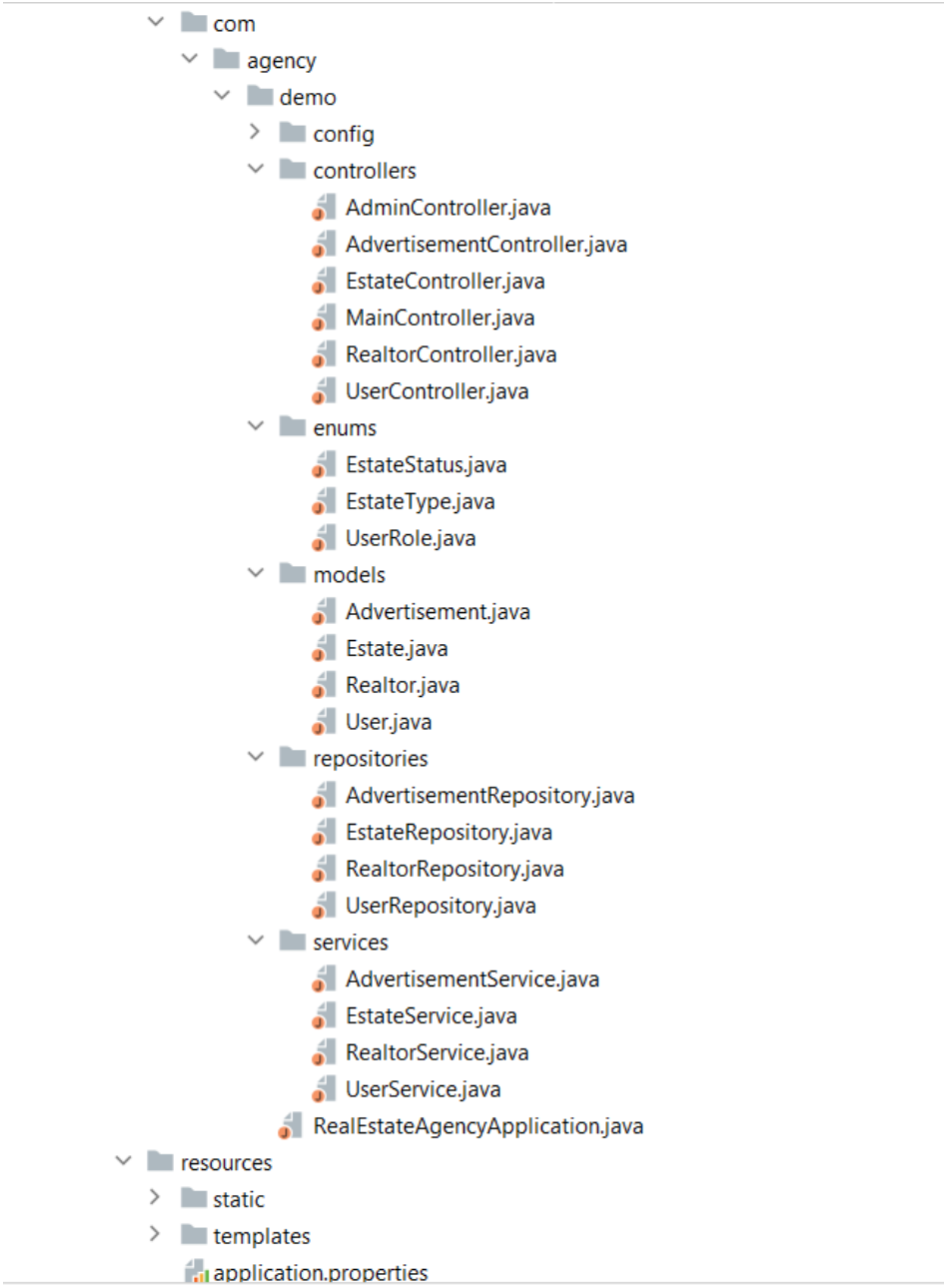

Рисунок 3.7 – Структура архіву з проєктом після генерації всіх компонентів

3.3 Огляд на процес роботи з прикладним програмним інтерфейсом та його тестування.

Одним з головних етапів розробки будь-якого програмного забезпечення є його тестування. В ході даного процесу перевіряється коректність роботи системи в цілому та стресостійкість до нестандартних ситуацій. Для того, щоб перевірити правильність роботи програмного інтерфейсу для генерації об’єктних моделей необхідно пройтися по кожному з пунктів генерації об’єктних моделей.

Перш за все, необхідно визначити файл з об’єктними моделями, який буде генерувати всі необхідні класи та сервіси. На рисунку 3.8 представлено структуру об’єктних моделей, які будуть використані в якості тестових.

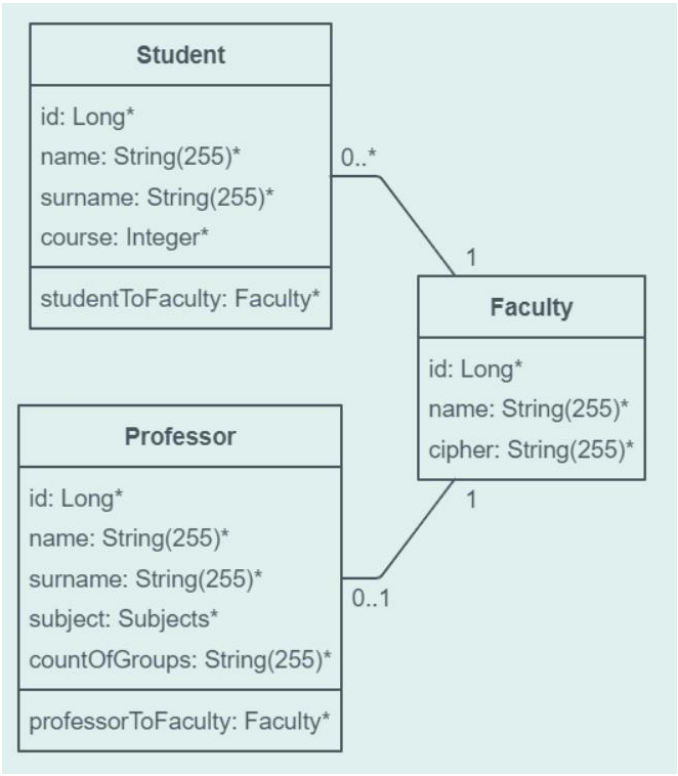

Рисунок 3.8 – структура класів об’єктних моделей для тестової генерації

На даному прикладі зображено три класи об’єктних моделей: студент, професор та факультет. Студент асоціюється до факультету з видом зв’язку багато-до-одного, так само і професор асоціюється до факультету як один-до-одного. Професор також містить в собі атрибут subject, який має тип даних відповідного перелічення.

Тепер необхідно описати дані об’єктні моделі в XML файлі згідно схеми, яка була розроблена в попередньому пункті. В додатку Д описано файл з об’єктними моделями згідно структури, яка зображена на попередньому рисунку.

Наступним кроком, треба запустити процес генерації класів та сервісів, подавши на вхід програмного інтерфейсу описаний вище файл з об’єктною моделлю. Для цього необхідно зайти в інтерфейс сервісу, завантажити файл та зачекати поки пройде процес генерації. В результаті користувач отримає архів з проєктом, в якому містяться усі необхідні класи та файли конфігурації. На рисунку 3.9 представлено структуру проєкту, яку отримано після процесу генерації.

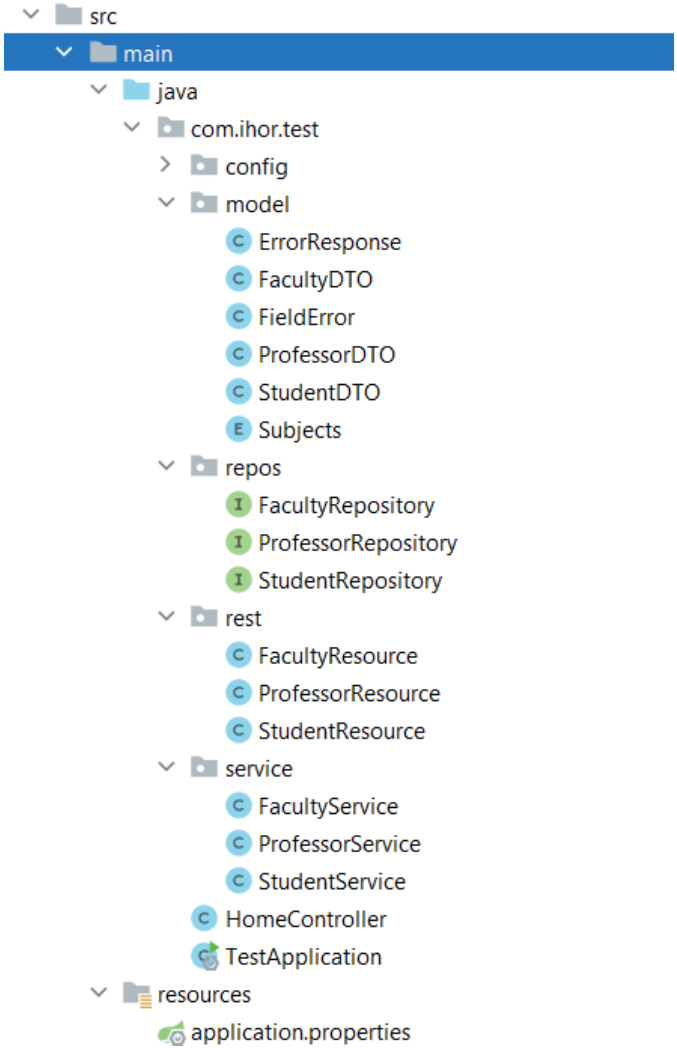

Рисунок 3.9 – Структура згенерованого тестового проєкту

Перед запуском процесу збірки та виконання даного сервісу, перш за все необхідно виконати SQL-скрипт, який створить базу даних в СКБД PostgreSQL.

Після успішного виконання даного скрипта, можна запускати процес збірки та компіляції сервісу. Якщо об’єктні моделі та класи веб-сервіси згенерувались правильно, компіляція пройде успішно, в іншому випадку буде повідомлено про помилку компіляції. Зачекавши кілька секунд, в вікні консолі з’явиться відповідне повідомлення про успішний запуск сервісу як на рисунку 3.10.

```
INFO 2800 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.le
INFO 2800 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
INFO 2800 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 47 ms. Found 3 JPA repo
INFO 2800 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
INFO 2800 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
INFO 2800 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.63]
INFO 2800 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
INFO 2800 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1546 ms
INFO 2800 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
INFO 2800 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.9.Final
INFO 2800 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
INFO 2800 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
INFO 2800 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
INFO 2800 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQL10Dialect
INFO 2800 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.
INFO 2800 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
INFO 2800 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
INFO 2800 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
INFO 2800 --- [  restartedMain] com.ihor.test.TestApplication            : Started TestApplication in 4.594 seconds (JVM running for 6.375)
INFO 2800 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
INFO 2800 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
INFO 2800 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
```

Рисунок 3.10 – Повідомлення про успішний старт сервісу

Фінальний етап тестування це перевірка коректності роботи REST API та веб-сторінок для редагування даних. В першу чергу необхідно перевірити REST веб-сервіси. Для цього використано популярний HTTP-клієнт Postman. На рисунках 3.11 – 3.14 зображено різні методи для отримання чи редагування даних на прикладі об'єктів типу Student.

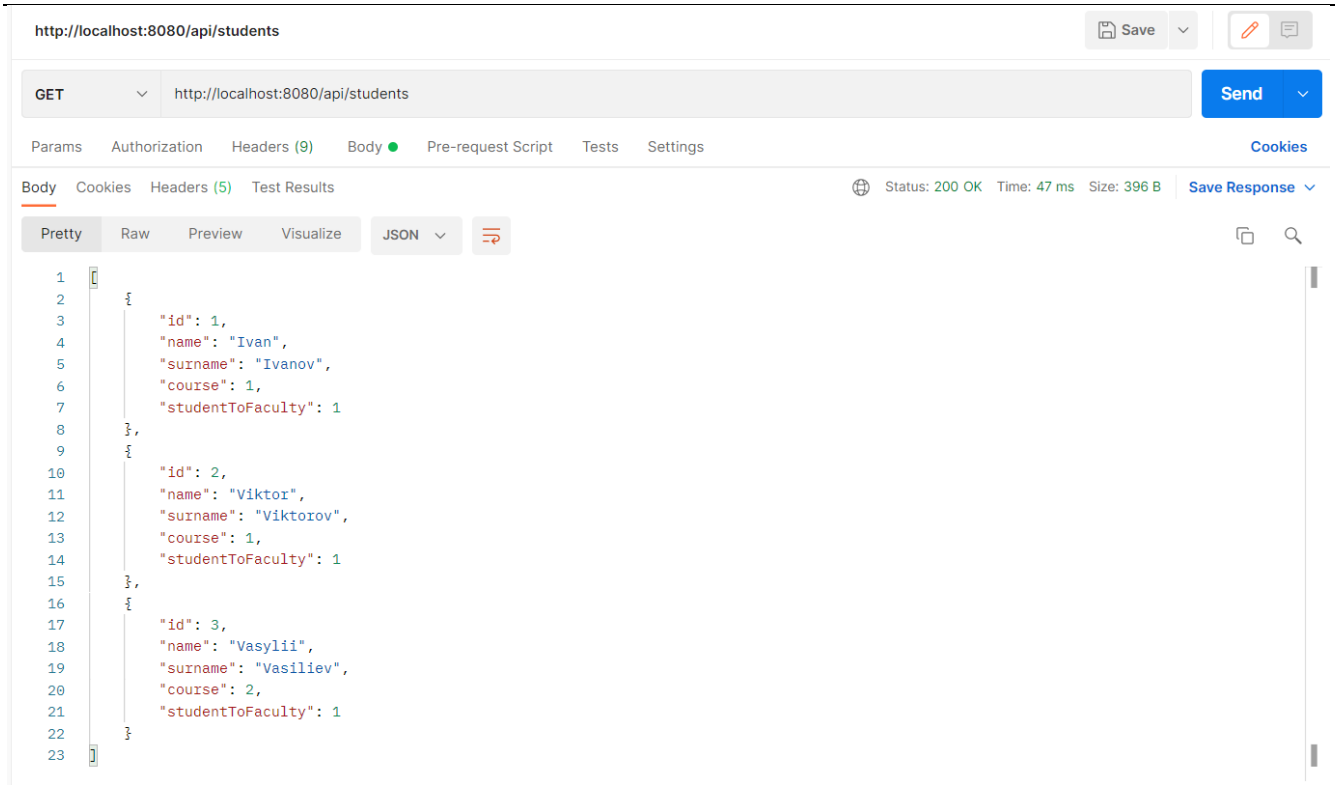

Рисунок 3.11 – Запит методу GET для отримання об’єктів типу Student

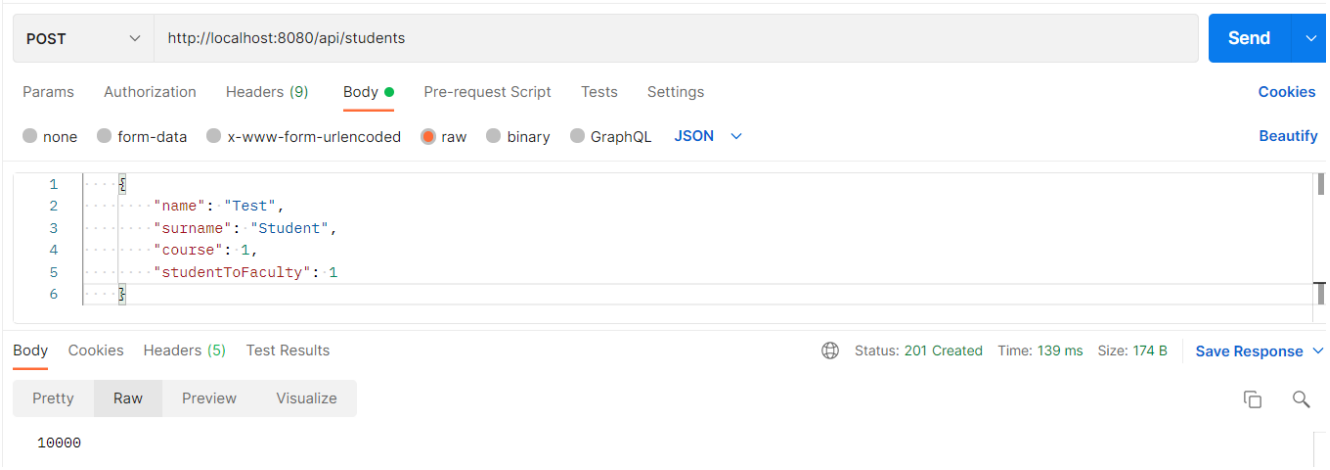

Рисунок 3.12 – Використання методу POST для створення об’єкту типу Student

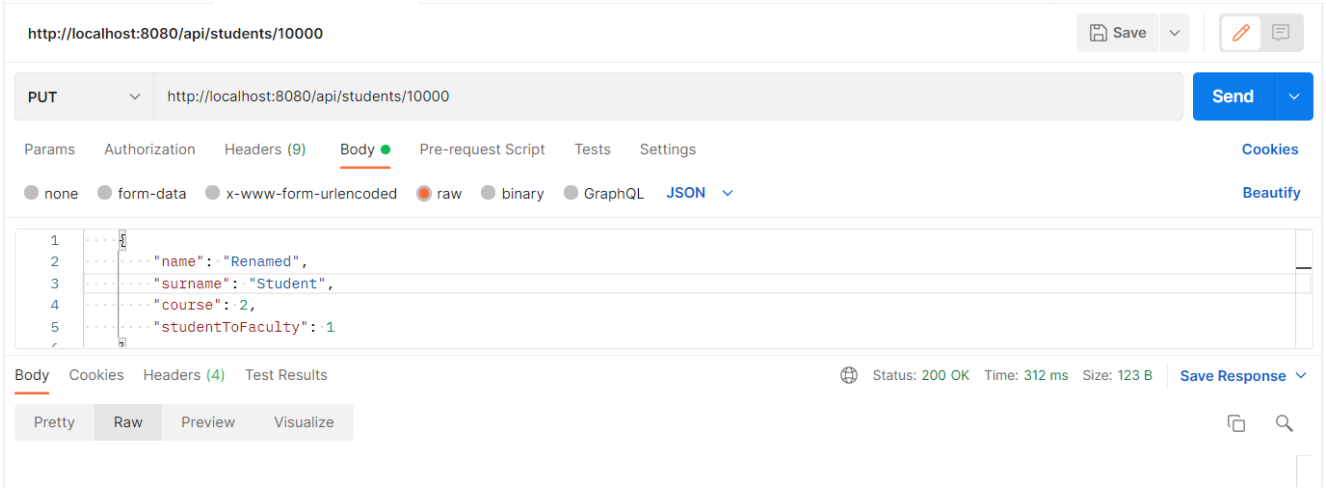

Рисунок 3.13 – Використання методу PUT для оновлення об’єкту типу Student

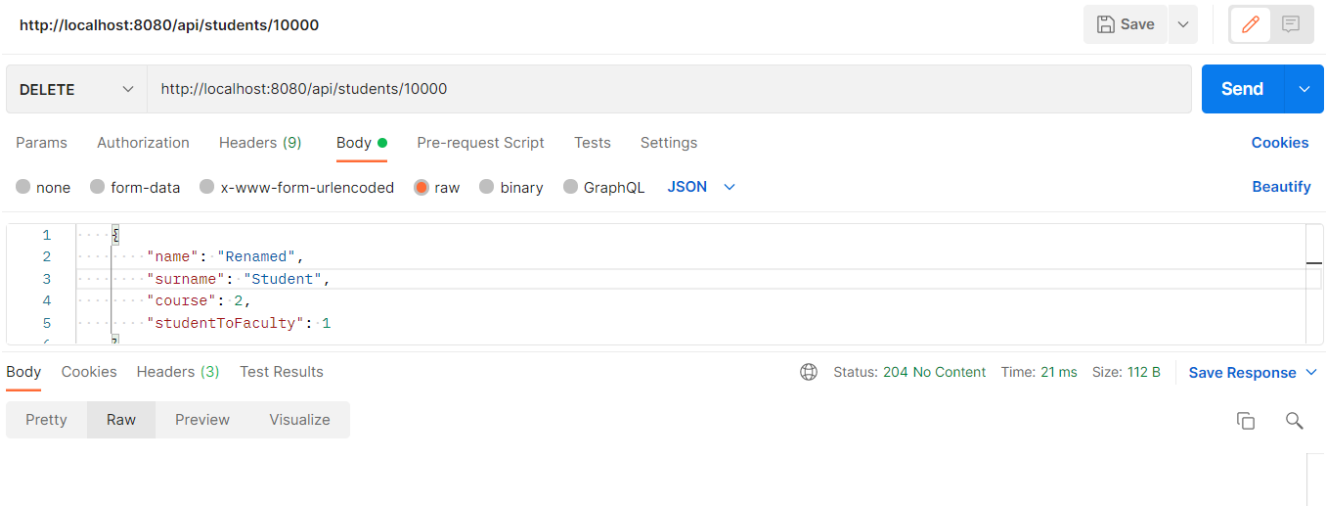

Рисунок 3.14 – Використання методу DELETE для видалення об’єкту типу Student

Важливо перевірити дану функціональність через інтерактивний інтерфейс у вигляді таблиць з даними. Для переходу на перегляд даних у вигляді таблиць необхідно перейти за наступним посиланням: ${HOST}/html/${OM-ID}, де ${HOST} – це хост серверу, а ${OM-ID} – ідентифікатор об’єктної моделі. На наступних рисунках 3.15 - 3.18 зображено таблицю об’єктів типу Student та операції над ними.

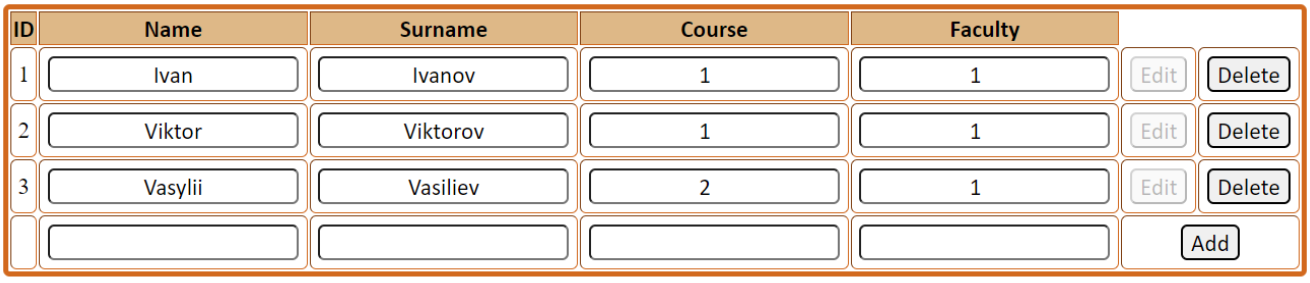

Рисунок 3.15 – Список усіх студентів у вигляді таблиці

ID	Name	Surname	Course	Faculty		
1	Ivan	Ivanov	1	1	Edit	Delete
2	Viktor	Viktorov	1	1	Edit	Delete
3	Vasylii	Vasiliev	2	1	Edit	Delete
	New	Student	4	1	Save	Cancel

Рисунок 3.16 – Додавання нового студента в список

ID	Name	Surname	Course	Faculty		
1	Edited	Ivanov	5	1	Edit	Delete
2	Viktor	Viktorov	1	1	Edit	Delete
3	Vasylii	Vasiliev	2	1	Edit	Delete
4	New	Student	4	1	Edit	Delete
					Add	

Рисунок 3.17 – Редагування студента з ідентифікатором 1

ID	Name	Surname	Course	Faculty		
1	Edited	Ivanov	5	1	Edit	Delete
2	Viktor	Viktorov	1	1	Edit	Delete
4	New	Student	4	1	Edit	Delete
					Add	

Рисунок 3.18 – Видалення студента з ідентифікатором 3

Висновки до розділу 3

В ході даного розділу змодельовано та створено структуру прикладного програмного інтерфейсу для генерації об’єктних моделей. В процесі моделювання структури програмного інтерфейсу використано сучасні технології для зберігання та обробки даних об’єктних моделей.

Для опису структури файлів з об’єктними моделями розроблено XSD схему, яка перевіряє валідність та цілісність введених метаданих об'єктних моделей. Файл розділено на три основні частини, які відповідають за різні структури об’єктних моделей, а саме: перелічення, безпосередньо, об’єктні моделі та асоціації.

Зчитування метаданих з файлу відбувається за допомогою функцій бібліотек Java для роботи з DOM. Після зчитування відбувається генерація в кілька етапів:

генерація класів, генерація сервісів доступу до даних, REST веб-сервісів та веб-сторінок з таблицями.

На завершальному етапі програмний інтерфейс формує архів з файлами класів, сервісів та конфігурацією для подальшої збірки та компіляції в готовий сервіс.

Для перевірки коректності роботи програмного інтерфейсу було проведено тестування основних функцій даної системи. В ході тестування критичних недоліків виявлено не було.

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«Прикладний програмний інтерфейс для генерації об’єктних моделей на основі вхідних даних користувача»

Спеціальність 122 «Комп’ютерні науки»

122 – БКР – 401.21810117

Виконав студент 4-го курсу, групи 401

І. А. Ларченко

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Консультант *ст. викладач*

(наук. ступінь, вчене звання)

О. В. Макарова

(підпис, ініціали та прізвище)

«___» _____ 2022 р.

Миколаїв – 2022

4 ОХОРОНА ПРАЦІ

Для підтримання максимальної працездатності та забезпечення необхідних умов праці необхідно притримуватись відповідних нормативно-правових актів. Однією із основних систем, нормативно правова база яких спрямована на регулювання правових, санітарно-гігієнічних, організаційних та технічних заходів та засобів для збереження життя та здоров’я та забезпечення високого рівня працездатності людини у процесі трудової діяльності є охорона праці. На території України дана система врегульована законом «Про охорону праці» [20]. Закон встановлює загальні вимоги з охорони праці для всіх підприємств та суб’єктів підприємницької діяльності незалежно від форм власності та видів діяльності. Тобто, положення даного закону так само впливають на підприємства, сфера діяльності яких спрямована інформаційні технології.

В сучасних реаліях створення належних умов праці є максимально необхідним, адже забезпечення безпеки життя та здоров’я працівника прямо пропорційно впливає на працездатність людини.

Метою спеціальної частини є створення безпечних і здорових умов праці на робочому місці або у виробничому приміщенні при розробці системи.

Відповідно до мети спеціальної частини, виділено наступні завдання:

- Проаналізувати наявні умови праці в робочому приміщенні;
- Визначити достатній рівень показників для робочого приміщення, де проводяться роботи з розробки системи;
- Визначити основні правила техніки безпеки при роботі з комп'ютерною технікою.

4.1 Аналіз умов праці та визначення шкідливих виробничих факторів в ході розробки системи

Одним із основних факторів, які впливають на здоров’я та працездатність людини при роботі в приміщенні є мікроклімат приміщення. Мікроклімат виробничих приміщень – це умови внутрішнього середовища цих приміщень, що впливають на тепловий обмін працюючих з оточенням. Як фактор виробничого середовища, мікроклімат впливає на теплообмін організму людини з цим середовищем і, таким чином, визначає тепловий стан організму людини в процесі праці [21]. Основними показниками мікрокліматичних умов виробничого приміщення є:

- температура повітря (°С),
- відносна вологість повітря (%),
- швидкість руху повітря (м/с),
- інтенсивність теплового (інфрачервоного) опромінювання (Вт/м^2) від поверхонь обладнання,
- температура поверхні.

Температура приміщення в якому проводилися аналіз та формування вимог до програмного забезпечення та його розробка в середньому становила 22 °С. Згідно даних про санітарні умови виробничих приміщень при виконанні робіт, пов’язаних з нервово-емоційним напруженням [22] даний показник цілком задовольняє оптимальні температурні умови.

Показник відносної вологості в приміщенні прирізних зовнішніх погодних умовах в середньому складав 54%. Порівнюючи даний показник з оптимальним для приміщень в яких виконуються дані типи робіт, можна зробити висновок, що даний відсоток повістю відповідає стандартам.

Швидкість руху повітря в приміщенні є доволі вагомим фактором, який впливає на здоров’я та самовідчуття працівника. Перш за все це стосується протягів і якщо працівник довгий час проведе на інтенсивному потоці повітря, існує висока вірогідність нанести шкоду здоров’ю. В ході виконання кваліфікаційної роботи, виміряти значення швидкості повітря було не можливо.

Інтенсивність теплового опромінення працюючих від нагрітих поверхонь технологічного устаткування, освітлювальних приладів, інсоляція від засклених огороджень не повинна перевищувати 35,0 Вт/м 2 - при опроміненні 50% та більше поверхні тіла, 70 Вт/м 2 - при величині опромінюваної поверхні від 25 до 50%, та 100 Вт/м 2 - при опроміненні не більше 25% поверхні тіла працюючого. Відносна вологість повітря

В ході виконання робіт зі створення програмних продуктів на фахівця можуть впливати такі шкідливі виробничі фактори, як: недостатнє або неякісне освітлення робочого місця, електромагнітне випромінювання від електронних пристроїв, ураження струмом, нервово-психологічні та емоційні навантаження, ураження чи перенапруження очей, інші негативні фактори.

4.2 Аналіз освітленості робочого приміщення

Доволі важливим фактором впливу на здоров’я та працездатність людини є освітленість приміщення. Для визначення того, чи є достатнім забезпечення природного освітлення в приміщенні порівнюють значення фактичної загальної площі вікон в приміщенні до мінімальної [23][24]. Мінімальна площа вікон в приміщенні визначається за формулою 4.1:

$$S_в = \frac{e_N \cdot k_з \cdot \eta_в \cdot S_{підл} \cdot k_{буд}}{\tau_{заг} \cdot r_1 \cdot 100} \qquad (4.1)$$

де e_N - нормований коефіцієнт природного освітлення для розглянутих умов праці,

$\kappa_з$ - коефіцієнт запасу, що приймається при розрахунках природного освітлення,

$\eta_в$ - світлова характеристика вікна,

$S_{підл}$ - площа підлоги виробничого приміщення ,

$\tau_{заг}$ - загальний коефіцієнт пропускання світла,

r_1 - коефіцієнт, що враховує підвищення коефіцієнта природного освітлення за рахунок світла, яке відбивається від внутрішніх поверхонь приміщення,

$K_{буд}$ - коефіцієнт, що враховує вплив протилежної будівлі на освітленість у виробничому приміщенні.

Вхідні дані для розрахунку мінімальної площі вікна для забезпечення природної освітленості наведені в таблиці 4.1.

Таблиця 4.1 – Вхідні дані для розрахунку необхідної площі вікн

№	**Параметр**	**Позначення**	**Розмірність**	**Значення**
1	Нормований коефіцієнт природного освітлення	e_N	%	0,9
2	Коефіцієнт запасу	$\kappa_з$	1	1,2
3	Світлова характеристика вікна	$\eta_в$	1	9.87
4	Площа підлоги	$S_{підл}$	м^2	8.23
5	Загальний коефіцієнт пропускання світла	$\tau_{заг}$	1	2,4
6	Коефіцієнт, що враховує підвищення коефіцієнта природного освітлення	r_1	1	0,621
7	Коефіцієнт, що враховує вплив протилежної будівлі на освітленість	$K_{буд}$	м	1.01

Визначення площі вікна, яке наявне в приміщенні за формулою 4.2:

$$S_{нв} = h * l = 1.6 * 1 = 1.6(м^2) \quad (4.2)$$

Наступним кроком необхідно обчислити значення площі вікна для забезпечення природної освітленості приміщення за формулою 1.1:

$$S_в = \frac{0.9 \cdot 1{,}2 \cdot 9{,}87 \cdot 8{,}32 \cdot 1.01}{2{,}4 \cdot 0{,}621 \cdot 100} = 0{,}6(\text{м}^2)$$

Згідно попередніх обчислень, можна зробити висновки, що того вікна, яке наявне в приміщені більш ніж достатньо для забезпечення природного освітлення в робочому приміщенні для робіт за комп’ютером.

4.3 Охорона здоров’я розробника при роботі за комп’ютером

Крім забезпечення робочого місця необхідними умовами, необхідно також враховувати те, що безперервна робота за комп’ютером несе шкоду здоров’ю працівника. Для розробників програм має бути встановлено перерву для відпочинку:

- тривалістю 15 хвилин через кожну годину роботи за комп’ютером;
- для операторів ЕОМ – 15 хвилин через кожні 2 години;
- для операторів комп’ютерного набору – 10 хвилин після кожної години роботи за комп’ютером.

Якщо виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи за комп’ютером не повинна перевищувати 4 годин.

При 12-годинній робочій зміні регламентовані перерви мають установлюватися в перші 8 годин роботи аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4 годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин.

Розмір робочого місця має становити не менше 6 квадратних метрів. На столі працівника можливо розмістити допоміжні для роботи пристрої а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику. Робочий стілець співробітника має бути підйомно-

поворотним, легко регульованим за висотою та забезпечувати належну підтримку та зручне положення спини і хребта особи. Щодня необхідно проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп’ютера від запиленості [25]. На рисунку 4.1 зображено схему розташування працівника та розміщення предметів за робочим місцем.

Рисунок 4.1 – Розміщення предметів за робочим столом працівника

4.4 Техніка безпеки під час користування комп’ютерними пристроями

Обов’язковою частиною охорони праці є техніка безпеки при роботі в той чи іншій сфері. Загальні положення техніки безпеки перелічені нижче.

Перед початком роботи на ПК користувач повинен:

– пересвідчитися у цілості корпусів і блоків (обладнання) ПК;
– перевірити наявність заземлення, справність і цілість кабелів живлення, місця їх підключення.

Забороняється вмикати ПК та починати роботу при виявлених несправностях. Під час роботи, пересвідчившись у справності обладнання, увімкнути електроживлення ПК, розпочати роботу, дотримуючись умов інструкції з її експлуатації [26].

При роботі за комп’ютером забороняється:

- замінювати і знімні елементи або вузли та проводити перемонтаж при ввімкненому ПК;
- з'єднувати і роз'єднувати вилки та розетки первинних мереж електроживлення, які знаходяться під напругою;
- знімати кришки, які закривають доступ до струмопровідних частин мережі первинного електроживлення при ввімкненому обладнанні;
- користуватися паяльником з незаземленим корпусом;
- замінювати запобіжники під напругою;
- залишати ПК у ввімкненому стані без нагляду.
- По закінченні роботи за комп’ютером:
- кнопкою "ВИМК" відключити електроживлення ПК згідно з інструкцією експлуатації, вийнявши вилку кабелю живлення з розетки;
- впорядкувати робоче місце користувача ПК, прибравши використане обладнання та матеріали у відведені місця;
- Залишаючи приміщення після закінчення роботи, дотримуючись встановленого режиму огляду приміщення, необхідно:
- зачинити вікна, кватирки;
- перевірити приміщення та переконатися у відсутності тліючих предметів;
- відключити від електромережі всі електроприлади, електрообладнання та вимкнути освітлення;

4.5 Пожежна безпека робочого приміщення

При експлуатації ПК необхідно пам'ятати, що первинні мережі електроспоживання під час роботи знаходяться під напругою, яка є небезпечною для життя людини, тому необхідно користуватися справними розетками, відгалужувальними та з'єднувальними коробками, вимикачами та іншими електроприладами. При виявленні в обладнанні ПК ознак несправності (іскріння, пробоїв, підвищення температури, запаху гару, ознак горіння) необхідно негайно припинити роботу, відключити усе обладнання від електромережі.

В разі виникнення пожежі чи надзвичайної ситуації необхідно негайно, за можливості вимкнути електропостачання комп’ютера, викликати відповідну службу. При наявності вогнегасника необхідно якомога скоріше погасити джерело полум’я. При необхідності евакуації необхідно притримуватися плану евакуації з робочого приміщення, який зображено на рисунку 4.2.

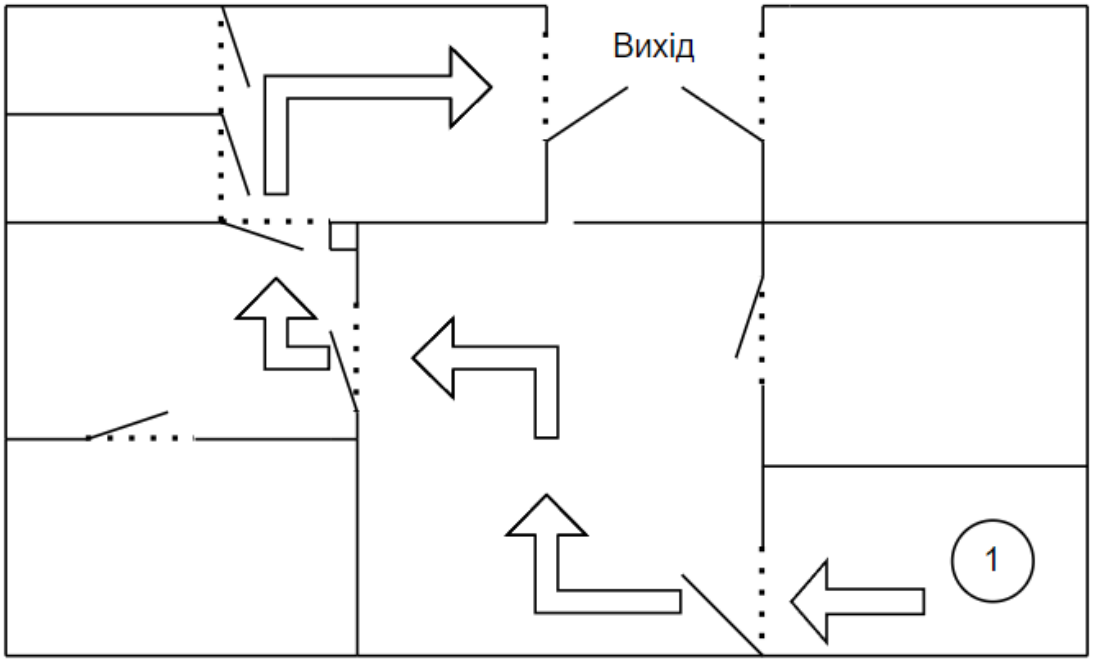

Рисунок 4.2 – План евакуації з робочого приміщення

Висновки до розділу про охорону праці

В даному розділі розглянуто основні положення про охорону праці робітників в сфері інформаційних технологій. Визначено основні вимоги до приміщень, в яких розміщене робоче місце працівника в даній сфері.

В ході першого підрозділу проведено аналіз умов праці в розглянутому робочому місці. Визначено, що приміщення відповідає вимогам та державним стандартам.

В другому підрозділі проведено розрахунки мінімальної площі вікна для забезпечення природної освітленості приміщення. Згідно розрахунків було визначено, що мінімальний розмір вікна для робочого приміщення складає $S_B = 0,6$

$м^2$, що при фактичному значенні площі вікна $S_{НВ} = 1,6\ м^2$ повністю забезпечує природний рівень освітленості.

В ході третього підрозділу визначено основні постулати, які сприяють забезпеченню збереження здоров’я працівника при роботі за комп’ютером.

В четвертому підрозділі розглянуто основні положення з техніки безпеки при роботі за комп’ютером для забезпечення безпеки здоров’ю і життю працівника за комп’ютерною технікою.

В останньому підрозділі розроблено план дій при надзвичайних ситуаціях та створено план евакуації з робочого приміщення.

ВИСНОВКИ

Сучасний світ неможливо уявити без інформаційних технологій, тому і попит на розробку різних систем залишається доволі високим. В даній роботі продемонстровано процес моделювання, розробки та тестування прикладного програмного інтерфейсу для генерації об’єктних моделей на основі вхідних даних користувача.

В першому розділі проаналізовано структуру та модель роботи програм-генераторів вихідного коду. Розглянуто поняття метаданих об’єктних моделей та поняття шаблонів коду. Крім того, в розділі проведено аналіз існуючих аналогів подібних систем, виділено їх сильні та слабкі сторони для уникнення помилок при розробці прикладного програмного інтерфейсу для генерації об’єктних моделей.

В другому розділі проведено загальний огляд на сучасні підходи реалізації систем корпоративного рівня, визначено, які технології користуються високою популярністю серед розробників. На основі цих даних було визначено стек технологій, який використаний в ході розробки програмного інтерфейсу. До його складу входять: Java 8, ORM Hibernate, Spring Framework, Apache FreeMarker, RDBMS Postgresql, XML та XSD.

В ході третього розділу описано процес моделювання прикладного програмного інтерфейсу, а саме визначено структуру файлів з метаданими об’єктних моделей, визначено види шаблонів, які використовуються для генерації вихідного коду. Також в розділі описано процес розробки та тестування програмного інтерфейсу. В процесі розробки пояснюються деякі нюанси розробки системи та реалізована загальна структура застосунку. Після розробки проведено тестування основних функцій програмного інтерфейсу, в ході якого визначено та вилучено основні недоліки. Крім того, в процесі тестування описано приклад використання програмного інтерфейсу.

Підсумовуючи наведені вище тези, можна зробити висновок про те, що головна мета роботи виконана в повній мірі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ГЕНЕРАТОР - тлумачення, орфографія, новий правопис онлайн. *Словник UA*: вебсайт. URL: https://slovnyk.ua/index.php?swrd=%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80 (дата звернення: 01.05.2022).
2. Слеповичев И. И. Генераторы псевдослучайных чисел: навчальний посібник, 2017. – 113 с.
3. Метадані – Вікіпедія. *Wikipedia*: вебсайт. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B4%D0%B0%D0%BD%D1%96 (дата звернення: 02.05.2022).
4. Comparison of code generation tools – *Wikipedia: веб-сайт.* URL: https://en.wikipedia.org/wiki/Comparison_of_code_generation_tools (дата звернення: 03.05.2022).
5. Code Generation and T4 Text Templates – *Docs Microsoft: веб-сайт.* URL: https://docs.microsoft.com/en-us/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2022 (дата звернення: 04.05.2022).
6. CodeSmith Generator – *CodeSmith Tools: веб-сайт.* URL: https://www.codesmithtools.com/product/generator#overview
7. Acceleo | Overview – *Eclipse*: *веб-сайт*. URL: https://www.eclipse.org/acceleo/overview.html (дата звернення: 04.05.2022).
8. Eclipse Modelling Project | The Eclipse Foundation – *Eclipse: веб-сайт.* URL: https://www.eclipse.org/modeling/emf/ (дата звернення: 04.05.2022).
9. Spring Boot Prototypes with Custom Database and REST API – *Bootify.io: вебсайт*. URL: https://bootify.io/ (дата звернення: 04.05.2022).
10. Roy Thomas Fielding Architectural Styles and the Design of Network-based Software Architectures: *дисертація.*University of California, Irvine 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. (дата звернення: 06.05.2022).

11. Tiobe Index – *TIOBE*: *веб-сайт.* URL: http://www.tiobe.com/tiobe-index/ (дата звернення: 07.05.2022).
12. JVM Ecosystem report 2018 - About your Platform and Application – *Snyk*: *веб-сайт*. URL:https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/ (дата звернення: 07.05.2022).
13. The Statistics Portal for Market Data, Market Research and Market Studies – *Statista*: *веб-сайт.* URL: https://www.statista.com/ (дата звернення: 07.05.2022).
14. PostgreSQL: The world's most advanced open source database – *PostgreSQL: веб-сайт*. URL: https://www.postgresql.org/ (дата звернення: 10.05.2022).
15. Extensible Markup Language (XML) 1.0 (Fifth Edition) – *W3C: веб-сайт*. URL: https://www.w3.org/TR/xml/ (дата звернення: 10.05.2022).
16. FreeMarker Java Template Engine – *Apache Freemarker: веб-сайт.* URL: https://freemarker.apache.org/ (дата звернення: 10.05.2022).
17. XSD — умный XML – *Хабр*: *веб-сайт.* URL: https://habr.com/ru/post/90696/ (дата звернення: 10.05.2022).
18. DOM Standard – *DOM*: *веб-сайт*. URL: https://dom.spec.whatwg.org/ (дата звернення: 11.05.2022).
19. Hypertext Transfer Protocol version 2: *веб-сайт.* URL: https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2-17 (дата звернення: 12.05.2022).
20. Про охорону праці: Закон України від 14 жовтня 1992 р. №49. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення 27.05.2022).
21. Інструкція з охорони праці для фахівця з інформаційних технологій – Довідник спеціаліста з охорони праці: веб-сайт. URL: https://www.sop.com.ua/article/1065-nstruktsya-z-ohoroni-prats-dlya-fahvtsya-z-nformatsynih-tehnology (дата звернення 27.05.2022).
22. Санітарні норми мікроклімату виробничих приміщень: Постанова МОЗ України від 01.12.1999 р. №42. URL: https://zakon.rada.gov.ua/go/va042282-99#:~:text=1.1.4.,%D1%82%D0%B5%D0%BC%D0%BF%D0%B5%D1%80%

D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D0%BF%D0%BE%D0%B2%D1%96%D1%82%D1%80%D1%8F%2022%2D24%20%D0%B3%D1%80%D0%B0%D0%B4 (дата звернення 27.05.2022).

23. Природне і штучне освітлення: Державні будівельні норми Україїни В.2.5-28-2006.
24. Практикум із охорони праці: уавчальний посібник / В.Ц. Жидецький, В.С. Джигирей, В.М. Сторожук та ін.; за ред. В.Ц. Жидецького. – Львів: Афіша, 2000. – 350 с.
25. Охорона праці при роботі з комп'ютером – Довідник спеціаліста з охорони праці: веб-сайт. URL: https://www.sop.com.ua/article/183-ohoron-prats-pri-robot-z-kompyuterom (дата звернення 27.05.2022).
26. Охорона праці в галузі інформаційних технологій : навчальний посібник / В.І. Голінько, М.Ю. Іконніков, Я.Я. Лебедєв ; М-во освіти і науки України, Нац. гірн. ун-т. – Д. : НГУ, 2015. – 246 с.

ДОДАТОК А
XSD схема файлу з метаданими

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema      attributeFormDefault="unqualified"      elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="object-model-generation-bundle">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="group-name" type="xs:string" />
        <xs:element name="project-name" type="xs:string" />
        <xs:element name="database-host" type="xs:string" />
        <xs:element name="database-user" type="xs:string" />
        <xs:element name="database-password" type="xs:string" />
        <xs:element name="object-models">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="object-model">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="object-model-id" type="xs:string" />
                    <xs:element name="java-class-name" type="xs:string" />
                    <xs:element name="abstract" type="xs:boolean" />
                    <xs:element name="persisted" type="xs:boolean" />
                    <xs:element name="generate-rest" type="xs:boolean" />
                    <xs:element name="generate-web-view" type="xs:boolean" />
                    <xs:element name="primary-key" type="xs:string" />
                    <xs:element name="attributes">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element maxOccurs="unbounded" name="attribute">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="attribute-name" type="xs:string" />
                                <xs:element name="datatype" type="xs:string" />
                                <xs:element        minOccurs="0"        name="enum-id"
type="xs:string" />
                                <xs:element name="required" type="xs:boolean" />
                                <xs:element name="unique" type="xs:boolean" />
                                <xs:element name="generate-getter" type="xs:boolean" />
                                <xs:element name="generate-setter" type="xs:boolean" />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="enumerations">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="enumeration">
                <xs:complexType>
                  <xs:sequence>
```

```
                    <xs:element name="enumeration-id" type="xs:string" />
                    <xs:element name="enumeration-class-name" type="xs:string" />
                    <xs:element name="value-type" type="xs:string" />
                    <xs:element name="mapped-values">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element maxOccurs="unbounded" name="key-value">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="key" type="xs:string" />
                                <xs:element name="value" type="xs:unsignedByte" />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="association-definitions">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="association-definition">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="source-om" type="xs:string" />
                    <xs:element name="target-om" type="xs:string" />
                    <xs:element name="association-type" type="xs:string" />
                    <xs:element name="min-cardinality" type="xs:unsignedByte" />
                    <xs:element name="max-cardinality" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

ДОДАТОК Б
Файл класу з REST веб-сервісом

```
package com.ihor.test.rest;

import com.ihor.test.model.StudentDTO;
import com.ihor.test.service.StudentService;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import java.util.List;
import javax.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/students", produces =
MediaType.APPLICATION_JSON_VALUE)
public class StudentResource {

    private final StudentService studentService;

    public StudentResource(final StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMapping
    public ResponseEntity<List<StudentDTO>> getAllStudents() {
        return ResponseEntity.ok(studentService.findAll());
    }

    @GetMapping("/{id}")
    public ResponseEntity<StudentDTO> getStudent(@PathVariable final Long id) {
        return ResponseEntity.ok(studentService.get(id));
    }

    @PostMapping
    @ApiResponse(responseCode = "201")
    public ResponseEntity<Long> createStudent(@RequestBody @Valid final StudentDTO
studentDTO) {
        return new ResponseEntity<>(studentService.create(studentDTO),
HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Void> updateStudent(@PathVariable final Long id,
            @RequestBody @Valid final StudentDTO studentDTO) {
        studentService.update(id, studentDTO);
        return ResponseEntity.ok().build();
    }

    @DeleteMapping("/{id}")
    @ApiResponse(responseCode = "204")
    public ResponseEntity<Void> deleteStudent(@PathVariable final Long id) {
        studentService.delete(id);
        return ResponseEntity.noContent().build();
    }

}
```

ДОДАТОК В
Файл з конфігурацією залежностей згенерованого проєкту

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath /><!-- lookup parent from repository -->
    </parent>
    <groupId>com.ihor</groupId>
    <artifactId>test</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>test</name>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-ui</artifactId>
            <version>1.6.8</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

ДОДАТОК Г
Файл з властивостями проєкту

```
spring.datasource.url=jdbc\:postgresql\://localhost\:5432/test
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.dbcp2.max-wait-millis=30000
spring.dbcp2.validation-query=SELECT 1
spring.dbcp2.validation-query-timeout=30
spring.jpa.hibernate.ddl-auto=update
spring.jpa.open-in-view=false
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.properties.hibernate.id.new_generator_mappings=true
springdoc.pathsToMatch=/api/**
```

ДОДАТОК Д
Файл з об'єктними моделями для тестування генерації

```
<object-model-generation-bundle>
    <group-name>com.ihor</group-name>
    <project-name>test</project-name>
    <database-host>localhost:5432</database-host>
    <database-user>postgres</database-user>
    <database-password>postgres</database-password>
    <object-models>
        <object-model>
            <object-model-id>student</object-model-id>
            <java-class-name>Student</java-class-name>
            <abstract>false</abstract>
            <persisted>true</persisted>
            <generate-rest>true</generate-rest>
            <generate-web-view>false</generate-web-view>
            <primary-key>id</primary-key>
            <attributes>
                <attribute>
                    <attribute-name>id</attribute-name>
                    <datatype>LONG</datatype>
                    <required>false</required>
                    <unique>false</unique>
                    <generate-getter>true</generate-getter>
                    <generate-setter>false</generate-setter>
                </attribute>
                <attribute>
                    <attribute-name>name</attribute-name>
                    <datatype>STRING</datatype>
                    <required>true</required>
                    <unique>false</unique>
                    <generate-getter>true</generate-getter>
                    <generate-setter>true</generate-setter>
                </attribute>
                <attribute>
                    <attribute-name>surname</attribute-name>
                    <datatype>STRING</datatype>
                    <required>true</required>
                    <unique>false</unique>
                    <generate-getter>true</generate-getter>
                    <generate-setter>true</generate-setter>
                </attribute>
                <attribute>
                    <attribute-name>course</attribute-name>
                    <datatype>INTEGER</datatype>
                    <required>true</required>
                    <unique>false</unique>
                    <generate-getter>true</generate-getter>
```

```
                <generate-setter>true</generate-setter>
            </attribute>
        </attributes>
    </object-model>
    <object-model>
        <object-model-id>professor</object-model-id>
        <java-class-name>Professor</java-class-name>
        <abstract>false</abstract>
        <persisted>true</persisted>
        <generate-rest>true</generate-rest>
        <generate-web-view>false</generate-web-view>
        <primary-key>id</primary-key>
        <attributes>
            <attribute>
                <attribute-name>id</attribute-name>
                <datatype>LONG</datatype>
                <required>false</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>false</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>name</attribute-name>
                <datatype>STRING</datatype>
                <required>true</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>surname</attribute-name>
                <datatype>STRING</datatype>
                <required>true</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>subject</attribute-name>
                <datatype>ENUM</datatype>
                <enum-id>subjects</enum-id>
                <required>true</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>countOfGroups</attribute-name>
                <datatype>INTEGER</datatype>
                <required>true</required>
```

```
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
        </attributes>
    </object-model>
    <object-model>
        <object-model-id>faculty</object-model-id>
        <java-class-name>Faculty</java-class-name>
        <abstract>false</abstract>
        <persisted>true</persisted>
        <generate-rest>true</generate-rest>
        <generate-web-view>false</generate-web-view>
        <primary-key>id</primary-key>
        <attributes>
            <attribute>
                <attribute-name>id</attribute-name>
                <datatype>LONG</datatype>
                <required>false</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>false</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>name</attribute-name>
                <datatype>STRING</datatype>
                <required>true</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
            <attribute>
                <attribute-name>cipher</attribute-name>
                <datatype>STRING</datatype>
                <required>true</required>
                <unique>false</unique>
                <generate-getter>true</generate-getter>
                <generate-setter>true</generate-setter>
            </attribute>
        </attributes>
    </object-model>
</object-models>

<enumerations>
    <enumeration>
        <enumeration-id>subjects</enumeration-id>
        <enumeration-class-name>Subjects</enumeration-class-name>
        <value-type>INTEGER</value-type>
        <mapped-values>
            <key-value>
```

```
                    <key>MACHINE_LEARNING</key>
                    <value>0</value>
                </key-value>
                <key-value>
                    <key>DATA_STRUCTURES</key>
                    <value>1</value>
                </key-value>
                <key-value>
                    <key>FUZZY_LOGIC</key>
                    <value>2</value>
                </key-value>
            </mapped-values>
        </enumeration>
    </enumerations>

    <association-definitions>
        <association-definition>
            <source-om>student</source-om>
            <target-om>faculty</target-om>
            <association-type>MANY_TO_ONE</association-type>
            <min-cardinality>0</min-cardinality>
            <max-cardinality>n</max-cardinality>
        </association-definition>
        <association-definition>
            <source-om>professor</source-om>
            <target-om>faculty</target-om>
            <association-type>ONE_TO_ONE</association-type>
            <min-cardinality>0</min-cardinality>
            <max-cardinality>n</max-cardinality>
        </association-definition>
    </association-definitions>
</object-model-generation-bundle>
```