Міністерство освіти і науки України НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка сервіс-форуму для студентів НаУКМА

Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки" 122

Керівник курсової роботи:
ст.викладач Вовк Н.Є.
_____ (підпис)
"____" _____ 2022 р.

Виконала студентка:
Лунякіна В.А.
_____ (підпис)
"____" _____ 2022 р.

Київ 2022

ЗМІСТ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультімедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультімедійних систем,

Доцент., к. ф.-м. н. О.П.Жижерун

_____

(підпис)

“____”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Лунякіній Валерії Андріївні факультету інформатики 4-го курсу

Тема: Розробка сервіс-форуму для студентів НаУКМА

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Анотація
4. Вступ
5. Аналіз предметної області. Постановка завдання курсової роботи
6. Теоретичні відомості
7. Опис реалізації програмного продукта
8. Висновки
9. Перелік використаних джерел

Дата видачі “___” _____ 2022 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання курсової роботи

Тема: Розробка сервіс-форуму для студентів НаУКМА

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	20.10.2022	
2.	Ознайомлення з існуючою інформацією по темі	10.11.2022	
3.	Ознайомлення з існуючими системами-аналогами роботи	23.11.2022	
4.	Початок створення практичної частини	09.03.2022	
5.	Початок написання теоретичної частини	25.04.2022	
6.	Остаточне завершення написання теоретичної частини роботи та розробки практичної частини; корегування	01.05.2022-20.05.2022	
7.	Здача курсової роботи	20.05.2022	

Студент Лунякіна В.А

Керівник Вовк Н.Є.

“_____” _____

Анотація

У роботі розглянуті можливі методи розробки застосунків, проаналізовані види СКБД. Розроблено сервіс-форум для допомоги студентам із пошуком потрібної інформації, на основі фреймворку SpringBoot та мови програмування Java.

ВСТУП

В сучасному світі однією із головних складових є спілкування та комунікація між людьми. Кожного дня ми обмінюємось новою інформацією, дізнаємось щось нове, навчаємось завдяки чомусь. Особливо це є важливою частиною студентського життя.

Усі учні вищих навчальних закладів прагнуть здобути найкращу освіту, закінчити університети із найкращими балами та дійсно стати компетентним у своїй сфері спеціалістами. Але, на жаль, не завжди це вдається легко.

Дуже часто, навіть майже постійно, студенти питають один в одного де можна знайти якусь певну інформацію, що було б краще почитати для засвоєння теми. Це значно краще дізнаватись у людей, які вже навчались на певних дисциплінах, ніж шукати в інтернеті, бо ті люди вже проходили через те та точно знають що дійсно є корисним, а на що навіть не варто витрачати час.

Завжди ці знання передавались “по знайомству”, що значно збільшує час на знаходження потрібної інформації. Бо, окрім того, щоб спитати і людини своє питання, треба буде ще знайти потрібну людину, яка володіє цією інформацією.

Отже, я вирішила якось полегшити життя студентам нашого університету та створити єдиний сервіс-форум, де буде зберігатися вся ця інформація. Будь-хто може зайти на сайт та запитати в інших те, що потрібно, а інші будуть допомагати.

Щоб якось мотивувати користувачів дійсно відповідати та допомагати я хочу додати систему рейтингу. Тобто у всіх буде можливість оцінити якість того чи іншого питання або коментаря. А відповідно до цих оцінок будуть формуватися власні рейтинги користувачів. Та в подальшому це буде впливати як швидко на їх питання будуть відповідати люди.

Таким чином це пришвидшить процес навчання, знаходження потрібної інформації, бо все це буде знаходитись на одній платформі та більше не буде потрібно ходити по мільйонах чатам та шукати потрібну людина. А також це дуже сблизить нашу спільноту студентів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1 Аналіз сучасного стану питання та обґрунтування теми

Складовою частиною студентського життя є комунікація та взаємодія один між одним та найбільшим особливим є передання знань. Зараз технологічний період, тому є багато різних месенджерів, електронні пошти та інші застосунки, за допомогою яких студенти можуть обмінюватись інформацією. Кожного дня вони дізнаються один в одного моменти, які їх цікавлять в навчанні, передають матеріали, які можуть допомогти. Це не є проблемою, але тільки якщо це студенти одного факультету, а то якщо комусь не вистачає певної інформації про предмет із іншої спеціальності, то тут вже постає питання як знайти потрібну людину та отримати бажану інформацію.
Також одним із мінусів такого типу обміну даними є те, що вони можуть загубитись та, якщо студенту буде потрібно швидко щось знайти, то це може зайняти певний час.
Тому я вирішила створити такий сервіс, де вся ця інформація буде зберігатися в одному місці, доступна для всіх та дуже легка для пошуку.

1.2 Постановка завдання

Отже основна ідея даної роботи це написання сервісу, за допомогою якого студенти зможуть створювати питання та відповідати на них.

Основні вимоги до сервісу:

- наявність бекофісу для керування даними

- можливість ставити лайки та дизлайки на питання та коментарі
- можливість створювати нові питання
- можливість додавати коментарі на питання
- передбачення рейтингу користувача, який буде вираховуватись за власною формулою, відповідно до всіх лайків та дизлайків на питаннях та коментарях користувача
- отримання усіх питань відсортованих за рейтингом користувача та датою створення питання
- реєстрація користувачів
- розподілення на ролі такі як, звичайний користувач та адміністратор

РОЗДІЛ 2. Теоретичні відомості

2.1 Аналіз засобів розробки форумів

Зараз у світі існує безліч сайтів-форумів. У кожного є свої переваги, унікальні особливості, кожен намагається чимось привабити користувачів саме для свого продукту. Тому, в залежності від особливостей сервісу розробники обирають ті чи інші засоби розробки.

Це все може залежати від архітектури сервісу, його внутрішньої логіки. Наприклад, якщо форум надає найпростіщі можливості просто відображати нові пости, додавати їх, то це все можно реалізувати за допомогою одного сервісу, де буде фронт частина та звичайні роути, CRUD операції.

У випадку, коли їх внутрішніми моделями застосунку потрібно робити якісь дії, перетворення, то це буде належно виділити у backend частину та не навантажувати frontend зайвою бізнес логікою.

Такі застосунки називають монолітами. Монолітний додаток будується як єдине ціле. Корпоративні програми складаються з трьох частин:

- База даних — складається з багатьох таблиць, як правило, в системі керування реляційною базою даних
- Інтерфейс користувача на стороні клієнта, що складається зі сторінок HTML та/або JavaScript, запущених у браузері)
- Додаток на стороні сервера, який оброблятиме HTTP-запити, виконує логіку домену, витягуватиме й оновлюватиме дані з бази даних, а також заповнювати HTML-подання, які надсилатимуться до браузера.

Це те, що робить монолітну архітектуру монолітною — це єдиний логічний виконуваний файл. Щоб внести будь-які зміни в систему, розробник повинен створити та розгорнути оновлену версію серверної програми.

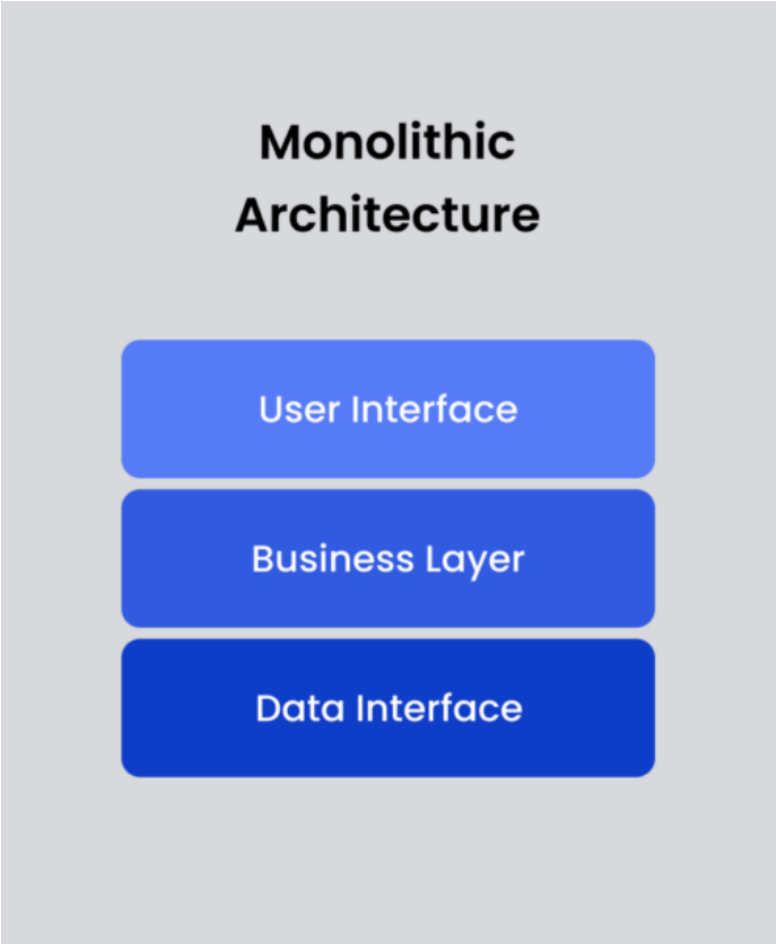


Рисунок 1 (Візуалізація монолітної архітектури застосунку)

Інколи особливості сайту можуть потребувати настільки складних перетворень, що ідеальним рішенням буде використання мікросервісної архітектури. Де кожен окремий сервіс буде відповідати за свою частину логіки та спілкуватися між собою.

Архітектура мікросервісів, яку часто називають мікросервісами, — це архітектурний підхід або стиль розробки додатків. Він передбачає поділ великих програм на менші функціональні блоки, здатні функціонувати та взаємодіяти незалежно.
Програми, що працюють на архітектурі мікросервісів, деконструюються та поділяються на менші блоки (мікросервіси), до яких клієнт може отримати доступ за допомогою шлюзу API. Кожна з цих мікросервісів може бути

розгорнута незалежно один від одного, але при необхідності вони можуть взаємодіяти один з одним.

Розбивка програми на мікросервіси дозволяє пришвидшити розробку, простіше виявляти й усувати помилки, простіше обслуговування, гнучкість, а також вищу доступність і масштабованість.

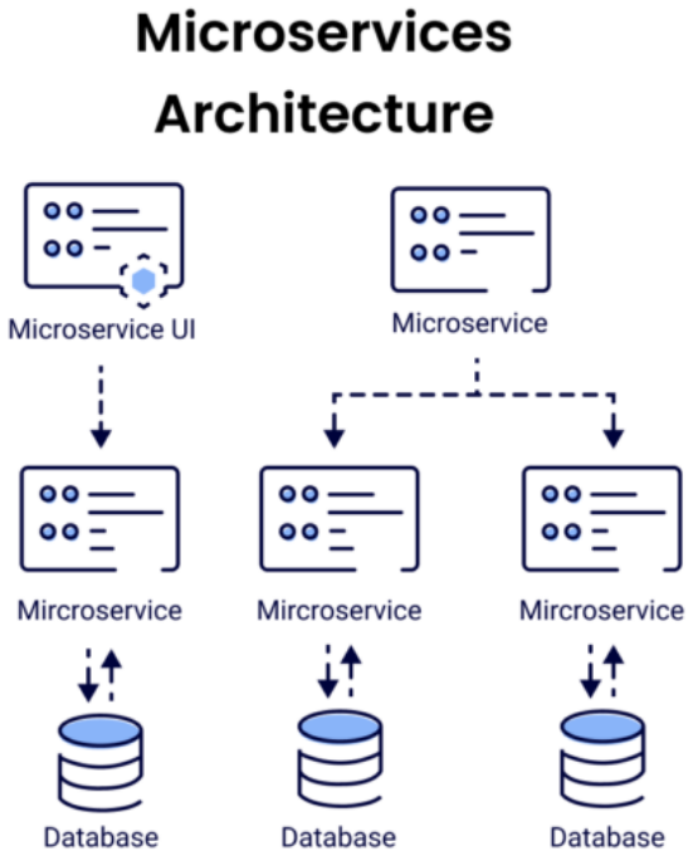


Рисунок 2 (Візуалізація мікросервісної архітектури застосунку)

2.2 Обґрунтування вибору засобів розробки

Проаналізувавши всі особливості свого форуму я вирішила, що найкращий варіант буде використати монолітну архітектуру, де окремим сервісом виділити backend частину.

Для розробки сервісу я обрала мову програмування Java. Вона є строго типізована об’єктно-орієнтованою мовою програмування, що мені дуже підходить.
Також я вирішила використовувати фреймворк Spring Boot. Його використання спрощує написання Spring-сервісів.

Переваги фреймворку:

- Автоконфігурація усіх компонентів у застосунку
- Наявність вже встроєних HTTP-ендпоінтів, які дозволяють отпимувати інформацію про стан сервісу тощо
- Відсутність XML конфігурації
- Швидкий та простий доступ до всіх баз даних
- Готові влаштовані сервіси, які допомагають швидко розгортати застосунки
- Великий вибір плагінів, який пришвидшує процес розробки сервісу

Для розробки frontend частини застосунку є також велика варіація рішень, але я вирішила зупинитися на React та TypeScript.

React - це бібліотека JavaScript, яка надає можливість створювати web-застосунки. Також вона має шаблони, які пришвидшують розробку застосунку.

Переваги React:

- Virtual DOM - Ця характеристика React допомагає прискорити процес розробки додатків і забезпечує гнучкість. Алгоритм полегшує реплікацію веб-сторінки у віртуальній пам’яті React.
- Одностороннє прив’язування даних. Це означає, що React використовує односпрямований потік даних, що змушує розробників використовувати функцію зворотного виклику для редагування компонентів і не дозволяє їм редагувати їх безпосередньо.
- Декларативний інтерфейс користувача. Ця функція робить код React більш читабельним і легше виправляти помилки. React JS є найкращою платформою для розробки інтерфейсів, які є одночасно захоплюючими та привабливими не лише для веб-програм, але й для мобільних додатків.
- Архітектура на основі компонентів. Це просто означає, що користувальницький інтерфейс програми на основі React JS складається з кількох компонентів, кожен з яких має свою особливу логіку, написану на JS. Завдяки цьому розробники можуть передавати дані в програму без впливу на DOM. Компоненти React JS відіграють величезну роль у вирішенні візуалізації та взаємодії програми.
- JavaScript XML або JSX. Це синтаксис розмітки, який описує зовнішній вигляд інтерфейсу програми. Він робить синтаксис схожим на HTML і використовується для створення компонентів React розробниками.

TypeScript — це строго типізована, об'єктно-орієнтована, компільована мова програмування.

Переваги TypeScript:

- Об'єктно-орієнтована мова: TypeScript надає повну функцію об'єктно-орієнтованої мови програмування, таку як класи, інтерфейси, спадкування, модулі тощо. У TypeScript ми можемо писати код для розробки як на стороні клієнта, так і на стороні сервера.
- TypeScript підтримує бібліотеки JavaScript: TypeScript підтримує кожен елемент JavaScript. Це дозволяє розробникам використовувати наявний код JavaScript із TypeScript. Тут ми можемо легко використовувати всі фреймворки, інструменти та інші бібліотеки JavaScript.
- JavaScript є TypeScript: це означає, що код, написаний на JavaScript з дійсним розширенням .js, можна перетворити на TypeScript, змінивши розширення з .js на .ts і скомпільований з іншими файлами TypeScript.
- TypeScript є портативним: TypeScript є портативним, оскільки його можна виконувати в будь-яких браузерах, пристроях або будь-яких операційних системах. Його можна запускати в будь-якому середовищі, де працює JavaScript. Це не є специфічним для будь-якої віртуальної машини для виконання.

2.3 Вибір СКБД

БД (Бази Даних) - концептуально організована сукупність даних, які маюсть залежності одні від одних. Вона може містити таблиці, схеми, процедури, об’єкти, тощо. Самі дані у БД організовуються відповідно до певної моделі, яка є схемою їх організації. Тим самим, сучасний варіант БД буде містити ще засоби обробки, крім самих даних.

СКБД (Системи керування базами даних) - це програмне забезпечення, яке надає можливість взаємодіяти із базою даних, отримувати, оновлювати та видаляти дані.

Існують SQL та NOSQL бази даних.

SQL база даних - реляційна модель баз даних, яка використовує SQL(structured query language) та має завчасно визначену схему даних, яка буде використовуватися. SQL є однією з найбільш універсальних і широко використовуваних мов запитів, що робить його безпечним вибором для багатьох випадків використання. Він ідеально підходить для складних запитів. Однак SQL може бути занадто обмежуючим. Перш ніж працювати з нею, потрібно використовувати попередньо визначені схеми, щоб визначити структуру даних. Усі ваші дані повинні мати однакову структуру. Цей процес вимагає серйозної попередньої підготовки. Якби ви коли-небудь захотіли змінити структуру даних, це було б складно і зашкодило б всій вашій системі.
Бази даних SQL у більшості ситуацій вертикально масштабовані. Ви можете збільшити навантаження на один сервер, додавши більше ємності CPU, RAM або SSD.

SQL бази даних є табличними базами. Є багато популярнів прикладів такі як MySQL, Oracle, PostgreSQL, and Microsoft SQL Server.
Бази даних SQL краще для багаторядкових транзакцій. Також вони зазвичай використовуються для застарілих систем, побудованих на основі реляційної структури.

NoSQL бази даних - це нереляційні бази даних. Вони мають динамічні схеми для неструктурованих даних, і дані зберігаються багатьма

способами. Ви можете використовувати для своїх даних сховище, орієнтоване на стовпці, документообіг, на основі графіків або KeyValue. Ця гнучкість означає, що:

- Ви можете створювати документи без попереднього визначення їх структури
- Кожен документ може мати свою унікальну структуру
- Синтаксис може відрізнятися від бази даних до бази даних
- Ви можете додавати поля по ходу

Бази даних NoSQL мають горизонтальне масштабування. Ви можете обробляти більший трафік за допомогою сегментування, що додає більше серверів до вашої бази даних NoSQL. Горизонтальне масштабування має більшу загальну ємність, ніж вертикальне масштабування, що робить бази даних NoSQL кращим вибором для великих і часто змінних наборів даних.

Бази даних NoSQL — це сховища документів, ключів і значень, графіків або широких стовпців. Наприклад, MongoDB, BigTable, Redis, RavenDB Cassandra, HBase, Neo4j, and CouchDB.
NoSQL краще підходить для неструктурованих даних, таких як документи або JSON.

Вивчивши переваги всіх баз даних я обрала для себе варіант нереляційної БД, а саме MongoDB.

MongoDB -це нереляційна база даних, яка маніпулює документами та не потребує завчасного визначення схеми для даних.

Вона має наступні переваги:

- Безкоштовна

- Має динамічну схему
- Горизонтально масштабована
- Чудова продуктивність із простими запитами
- Можна додавати нові стовпці та поля, не впливаючи на наявні рядки чи продуктивність самої програми

РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТА

3.1 Аналіз технічного завдання

Отже метою є створення застосунку, який буде форумом для студентів НаУКМА, де вони зможуть створювати пости та ділитись інформацією один з одним. Це значно спростить процес навчання та пошуку потрібних знань для того чи іншого предмету.
Сервіс передбачає дві ролі користувачів: звичайний користувач та адміністратор.

Технічні можливості користувача:

- створити аккаунт
- редагувати власні дані
- створювати пости, прив’язані до дисциплін
- додавати коментарі до постів інших користувачів
- ставити лайки та дізлайки на пости
- ставити лайки та дізлайки на коментарі

- переглядати свій власний рейтинг, який залежить від кількості лайків та дизлайків на всіх постах та коментарях, та розраховується за формулою
- зробити пошук по всім постам за назвою
- зробити пошук по всім постам за дисципліною

Технічні можливості адміністратора:

- усі можливості, які доступні для звичайного користувача
- створювати акаунт із правами адміністратора
- редагувати власні дані
- видаляти користувачів
- користуватися бекофісом
- додавати, редагувати, видаляти дисципліни
- редагувати рейтинги користувачів
- видаляти, редагувати пости користувачів
- видаляти, редагувати коментарі користувачів

Також для зручності роботи адміністратора потрібно реалізувати таку частину, як бекофіс.

Бекофіс - підрозділ сервісу, який надає доступ до усіх даних, які використовуються у застосунку.

Можливості бекофісу:

- отримувати усі наявні дисципліни
- створювати, оновлювати, видаляти дисципліни
- отримувати усі наявні пости
- створювати, оновлювати, видаляти пости
- отримувати усі наявні коментарі
- створювати, оновлювати, видаляти коментарі

- отримувати усіх користувачів
- створювати, оновлювати, видаляти користувачів

3.2 Обґрунтування структури програми

Отже, для реалізації будь-якого застосунку спочатку потрібно визначитись із сутностями.

Я виділила 4 головні сутності:

- Дисципліна
- Пост (Питання)
- Коментар
- Користувач

Дисципліна сама по собі є незалежною сутністю, вона у собі не має ніяких зв’язків. Більш детальна модель (Додаток А).

Питання має зв’язки із коментарями, тобто воно зберігає список усіх своїх коментарів. Також зв’язок із користувачем, який є автором питання. Та прив’язано до дисциплін, яких може бути декілька, тобто воно також зберігає у собі список усіх дисциплін, до яких відноситься. Більш детальна модель (Додаток Б).

Коментар має зв’язок із питанням та користувачем, який є автором коментаря. Більш детальна модель (Додаток В).

Користувач має зв’язок із питаннями, тобто зберігає у собі список усіх питань/постів, які він створив. Також має зв’язок із коментарями, містить список усіх його коментарів. Більш детальна модель (Додаток Г).

Також однією із важливою частиною застосунку та логіки є рейтинг. Він є у користувача та розраховується з урахуванням усіх лайків та дизлайків його питань та коментарів.

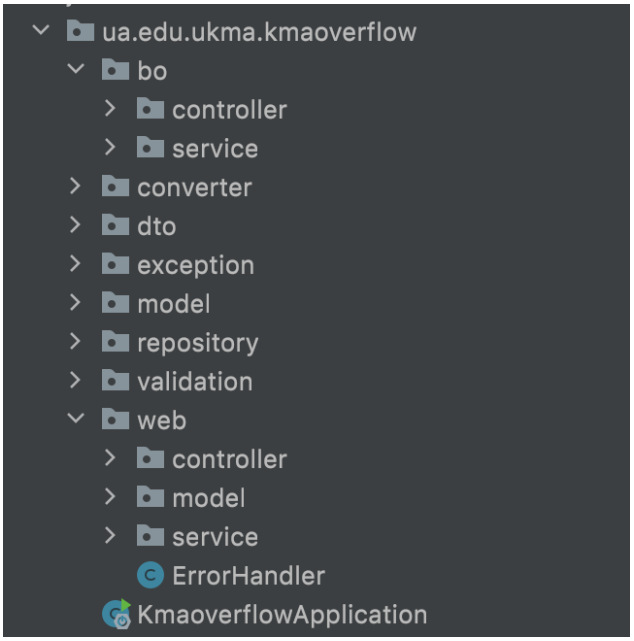
Розраховується він за наступною формулою:
Рейтинг = (n * 0,1) - (k * 0,2) + (q * 0,4) + (c * 0,3)

n - кількість лайків на питаннях
k - кількість дизлайків на питаннях
q - кількість лайків на коментарях
c - кількість дизлайків на коментарях

Такий вибір коефіцієнтів зумовлений тим, що коментарі цінуються більше, ніж питання. Бо, якщо ти зможеш допомогти людині, то це цінніше, ніж те, що ти зміг задати питання.

3.3 Опис реалізації

Сервіс має наступну архітектуру пакетів:



Є окремий пакет “bo”, де знаходяться всі класи, які мають відношення до логіки бекофісної частини застосунку. Там знаходиться пакет “controller”, з усіма класами-контролерами, до яких надходять запити на взаємодію із усіма наявними моделями у сервісу.

Пакет “converter” зберігає усі класи-конвертори, які відповідають за перехід від DTO до внутрішньої моделі, з якої в подальшому будуть відбуватись зміни, відповідно до логіки.
DTO (data transfer object) - шаблон проектування, який використовується для передачі даних між сервісами. Він не має ніякої логіки, тому щоб над ним можна було би проводити якісь маніпуляції, його потрібно буде конвертувати у бізнесову модель.

Кожен клас конвертера імплементує інтерфейс ConverterInterface<D, M> , який містить два метода.

```java
public interface ConverterInterface<D, M> {

    M fromDto(D dto);

    D toDto(M model);

}
```

Приклад імплементації конвертора:

```java
@Component
public class DisciplineConverter implements ConverterInterface<DisciplineDto, Discipline>{

    @Override
    public Discipline fromDto(DisciplineDto dto) {
        return Discipline.builder()
                .id(dto.getId())
                .code(dto.getCode())
                .name(dto.getName())
                .teacher(dto.getTeacher())
                .description(dto.getDescription())
                .build();
    }

    @Override
    public DisciplineDto toDto(Discipline model) {
        return DisciplineDto.builder()
                .id(model.getId())
                .code(model.getCode())
                .name(model.getName())
                .teacher(model.getTeacher())
                .description(model.getDescription())
                .build();
    }
}
```

Далі є пакет “dto”, який зберігає всі dto моделі, які використовуються у програмі, а саме у бекофісі, такі як UserDto, QuestionDto, AnswerDto, DisciplineDto.

Приклад однієї dto моделі:

```
@Value
@Builder
public class AnswerDto {
  @NotBlank(groups = OnUpdate.class)
  String id;

  @NotBlank(groups = OnCreate.class)
  String body;

  Integer likes;

  Integer dislikes;

  @NotBlank(groups = OnCreate.class)
  String createdAt;

  @NotBlank(groups = OnCreate.class)
  Question question;

  @NotBlank(groups = OnCreate.class)
  User author;
}
```

Так як ці моделі ми очікуємо у вигляді JSON-об’єктів, які будуть надходити до серверу у тілі запитів, то на них обов’язково потрібно зробити валідацію.

Окрім звичайної валідації я ще зробила власні групи валідацій (OnCreate, OnUpdate), які будуть валідувати поля тільки за певних умов. Як у даному прикладі, на створення та на оновлення моделі.

Потім у проекті присутній пакет “model”, який зберігає усі бізнес моделі застосунку, які відповідають dto. Також вони ще є структурою документів для бази даних MongoDB.

Приклад моделі:

```java
@Data
@Document
@Builder
public class Answer {
  @Id
  private String id;

  private String body;

  private Integer likes;

  private Integer dislikes;

  private String createdAt;

  @ManyToOne
  @JoinColumn(name="question_id")
  private Question question;

  @ManyToOne
  @JoinColumn(name="user_id")
  private User author;
}
```

Як можна побачити, то тут ще прописані зв’язки між документами у базі даних та типи цих зв’язків.

Пакет “repository” містить у собі усі класи-репозиторії, які є інтерфейсами, та взаємодіють із базою даних. Містять усі потрібні запити, які потім будуть використовуватися у класах-сервісах.

Пакет “exception” має усі потрібні власні помилки, які буде кидати застосунок. Щоб вони повертались користувачу у нормальному вигляді вони конвертуються у модель Error Response.

```
@Value
public class ErrorResponse {
  String message;
}
```

Ця конвертація знаходиться у класі ErrorHandler. Він відповідає за те, щоб відловлювати усі помилки, які трапляються у програмі та конвертування їх у модель, зрозумілу для користувача.

Пакет “web” відповідає за всю головну логіку застосунку.

Пакет “web.model” зберігає всі моделі, які будуть надходити у тілі запитів. Створення додаткових моделей для запитів зумовлено тим, щоб не було одразу багато запитів на одну дію та щоб не передавати непотрібні дані. Наприклад, для того, щоб поставили лайк на коментар у першому варіанті потрібно б було відправити запити на оновлення коментаря та користувача, а тут це буде зроблено одним запитом.

Приклад моделі запиту:

```
@Value
public class AnswerRequest {

  @NotBlank
  String body;

  @NotBlank
  String authorNickname;

  @NotBlank
  String questionId;
}
```

Пакет “web.controller” містить усі класи-контролери, які приймають запити на основну бізнес логіку застосунку.

Пакет “web.service” у собі має усі класи-сервіси, які використовують ті ж самі класи-репозиторії із пакету “repository”, що й сервіси бекофісу.

Також для розподілення ролей користувачів на звичайного користувача та адміністратора, та для можливості логіну до застосунку було додане Spring Security.

Spring Securiry - це фреймворк, який дозволяє додавати аутентифікацію та авторизацію до застосунку.

Для реалізації цієї логіки був створений клас-конфігурації WebSecurityConfig,де прописані доступні енпоінти для кожного користувача.

Опис контрактів сервісу:

Бекофіс:

Endpoint: /bo/users
HTTP method: GET
Призначення: отримання інформації про всіх користувачів

Endpoint: /bo/users/{id}
HTTP method: GET
Призначення: отримання інформації користувача за його ідентифікаційним номером у системі
Endpoint: /bo/users
HTTP method: POST
Request body:
{
“nickname” : “<nickname>”,
“password”: “<password>”,
“rating”: <rating>,
“questions” : [<questionId, questionId>],
“answers”: [<answerId>, <answerId>]
}
Призначення: створення користувача

Endpoint: /bo/users
HTTP method: PATCH
Request body:
{
“nickname” : “<nickname>”,
“password”: “<password>”,
“rating”: <rating>,

“questions” : [<questionId, questionId>],
“answers”: [<answerId>, <answerId>]
}
Призначення: оновлення користувача

Endpoint: /bo/users/{id}
HTTP method: DELETE
Призначення: видалення користувача за його ідентифікаційним номером у системі

Endpoint: /bo/disciplines
HTTP method: GET
Призначення: отримання інформації про всі дисципліни

Endpoint: /bo/disciplines/{id}
HTTP method: GET
Призначення: отримання інформації про дисципліну за її ідентифікаційним номером у системі

Endpoint: /bo/disciplines
HTTP method: POST
Request body:
{
“code” : “<code>”,
“name”: “<name>”,
“teacher”: “<teacher>”,
“description”: “<description>”
}
Призначення: створення дисципліни

Endpoint: /bo/disciplines
HTTP method: PATCH
Request body:
{
“code” : “<code>”,
“name”: “<name>”,
“teacher”: “<teacher>”,
“description”: “<description>”
}
Призначення: оновлення дисципліни

Endpoint: /bo/disciplines/{id}
HTTP method: DELETE
Призначення: видалення дисципліни за її ідентифікаційним номером у системі

Endpoint: /bo/questions
HTTP method: GET
Призначення: отримання інформації про питання

Endpoint: /bo/questions/{id}
HTTP method: GET
Призначення: отримання інформації про питання за його ідентифікаційним номером у системі

Endpoint: /bo/questions
HTTP method: POST
Request body:

{
“title” : “<title>”,
“body”: “<body>”,
“likes”: <likes>,
“dislikes”: <dislikes>,
“createdAt”: “<createdAt>”,
“disciplines”: [<disciplineId>,<disciplineId>],
“answers”: [<answerId>, <answerId>],
“author”: “<authorId>”,}
Призначення: створення питання
Endpoint: /bo/questions
HTTP method: PATCH
Request body:
{
“title” : “<title>”,
“body”: “<body>”,
“likes”: <likes>,
“dislikes”: <dislikes>,
“createdAt”: “<createdAt>”,
“disciplines”: [<disciplineId>,<disciplineId>],
“answers”: [<answerId>, <answerId>],
“author”: “<authorId>”
}
Призначення: оновлення питання

Endpoint: /bo/questions/{id}
HTTP method: DELETE
Призначення: видалення питання за його ідентифікаційним номером у системі

Endpoint: /bo/answers
HTTP method: GET
Призначення: отримання інформації про коментарі

Endpoint: /bo/answers/{id}
HTTP method: GET
Призначення: отримання інформації про коментар за його ідентифікаційним номером у системі

Endpoint: /bo/answers
HTTP method: POST
Request body:
{
“body”: “<body>”,
“likes”: <likes>,
“dislikes”: <dislikes>,
“createdAt”: “<createdAt>”,
“question”: “<questionId>”
“author”: “<authorId>”
}
Призначення: створення коментаря

Endpoint: /bo/answers
HTTP method: PATCH
Request body:
{
“body”: “<body>”,
“likes”: <likes>,

“dislikes”: <dislikes>,
“createdAt”: “<createdAt>”,
“question”: “<questionId>”
“author”: “<authorId>”
}
Призначення: оновлення коментаря

Endpoint: /bo/answers/{id}
HTTP method: DELETE
Призначення: видалення коментаря за його ідентифікаційним номером у системі

Головний контракт сервісу:

Endpoint: /blog/createQuestion
HTTP method: POST
Request body:
{
“title”: “<title>”,
“body”: <body>,
“disciplinesCodes”: [<code>,<code>],
“authorNickname”: “<authorNickname>”
}
Призначення: створення питання

Endpoint: /blog/createAnswer
HTTP method: POST
Request body:
{

“questionId”: “<questionId>”,

“body”: <body>,

“authorNickname”: “<authorNickname>”

}

Призначення: створення коментаря

Endpoint: /blog/like

HTTP method: POST

Request body:

{

“postId”: “<postId>”,

“isQuestion”: <isQuestion>

}

Призначення: лайк питання чи коментаря

Endpoint: /blog/dislike

HTTP method: POST

Request body:

{

“postId”: “<postId>”,

“isQuestion”: <isQuestion>

}

Призначення: дизлайк питання чи коментаря

Endpoint: /blog/question/{id}

HTTP method: DELETE

Призначення: видалення питання

Endpoint: /blog/questions/{id}

HTTP method: GET

Призначення: отримання питання за його ідентифікаційним номером у системі

Endpoint: /blog/questions

HTTP method: GET

Призначення: отримання усіх питань сортованих за рейтингом користувача та датою створення

Висновки

В ході виконання даної роботи були грунтовно проаналізовані та розглянути усі теоретичні відомості, технології розробки, види вже існуючих сервісів-форумів, різні види архітектури та підходи до написання проектів.

Були реалізовані поставлені задачі у технічному завданні, такі як додавання, редагування, видалення всіх сутностей за допомогою сервісу бекофісу, розподілення користувачів на ролі (звичайні користувачі та адміністратори), вирахування та нарахування рейтингу користувачу, можливість лайкати та дизлайкати питання та коментарі, прив’язка питань до наявних дисциплін.

Усі технології, які були обрані мною, ідеально підійшли для реалізації подібного застосунку та задовільнили усі умови.

Та як результат був розроблений сервіс, який виконує усі поставлені задачі та зручний у користуванні.

Додаток А

Discipline:

Поле	Тип	Опис
id	String	Унікальний ідентифікатор дисципліни
code	String	Унікальний код дисципліни в системі університету
name	String	Назва дисципліни
teacher	String	ПІБ викладача
description	String	Опис дисципліни
questions	List<Question>	Список усіх питань, які мають відношення до дисципліни

Додаток Б

Question:

Поле	Тип	Опис
id	String	Унікальний ідентифікатор питання
title	String	Назва питання
body	String	Текст питання
likes	Integer	Кількість лайків питання
dislikes	Integer	Кількість дизлайків питання
createdAt	String	Дата та час, коли питання було створене
disciplines	List<Discipline>	Список усіх дисциплін, до яких питання прив’язане
answers	List<Answer>	Список усіх коментарів питання
author	User	Автор питання

Додаток В

Answer:

Поле	Тип	Опис
id	String	Унікальний ідентифікатор коментаря
body	String	Текст коментаря
likes	Integer	Кількість лайків коментаря
dislikes	Integer	Кількість дизлайків коментаря
createdAt	String	Дата та час, коли був створений коментар
question	Question	Питання, до якого відноситься коментар
author	User	Користувач, який написав коментар

Додаток Г

User:

Поле	Тип	Опис
id	String	Унікальний ідентифікатор користувача
nickname	String	Нікнейм користувача
password	String	Пароль користувача
rating	Double	Власний рейтинг користувача
questions	List<Question>	Усі питання користувача
answers	List<Answer>	Усі коментарі користувача

Список використаної літератури

[1] Java Documentation - https://docs.oracle.com/en/java/

[2] SpringBoot Documentation - https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[3] Spring Security - https://spring.io/projects/spring-security

[4] Comparing databases - https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

[5] Top NoSql databases - https://www.trustradius.com/nosql-databases

[6] MongoDB Documentation - https://www.mongodb.com/docs/

[7] TypeScript Documentation - https://www.typescriptlang.org/docs/

[8] React Documentation - https://www.typescriptlang.org/docs/handbook/react.html