МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
"_____"_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Онлайнова платформа пошуку ментора у навчанні

Виконавець: Коваленко Д.І.

Керівник: Артамонов Є.Б.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут неперервної освіти
Кафедра комп'ютеризованих систем управління

Освітньо-кваліфікаційний рівень бакалавр
Напрям (спеціальність) 6.050102 "Комп'ютерна інженерія"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.
«____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

Коваленка Дмитра Ігоровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема роботи: Онлайнова платформа пошуку ментора у навчанні

затверджена наказом ректора від " 04 " лютого 2021 року № 135 /ст.

2. Термін виконання роботи: з 17.05.2021 до 20.06.2021

3. Вихідні дані до проєкту (роботи): Використання мови *Java* для реалзації онлайнової платформи пошуку ментора у навчанні

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

Аналіз та вибір архітектури вебдодатку

Аналіз та вибір бази даних вебдодатку

Розробка вебдодатку

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Алгоритм метода *login* (схема алгоритму)

2) Алгоритм метода *checkPassword* (схема алгоритму)

3) *UML* схема бази даних

4) Сторінка відображення профіля ментора

6. Календарний план

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проектування	17.05.21 – 20.05.21	
2	Вивчити спеціальну літературу і технічну документацію	20.05.21– 25.05.21	
3	Проаналізувати архітектуру вебдодатку та бази даних	25.05.21 – 30.05.21	
4	Написати і налагодити програму	30.05.21 – 10.06.21	
5	Написати пояснювальну записку	10.06.21 – 15.06.21	
6	Підготовити графічний і демонстраційний матеріали	15.06.21 – 20.06.21	

7. Дата видачі завдання “ 17 ” травня 2021р.

Керівник дипломної роботи (проєкту) _____ Артамонов Є.Б.
(підпис керівника) (П. І. Б.)

Завдання прийняв до виконання _____ Коваленко Д.І.
(підпис студента) (П. І. Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Онлайнова платформа пошуку ментора у навчанні”: 60 с., 30 рис., 15 літературних джерел, 1 додаток.

ОНЛАЙНОВА ПЛАТФОРМА ПОШУКУ МЕНТОРА У НАВЧАННІ

Мета дослідження – розробити вебдодаток для пошуку ментора у навчанні.

Об’єкт дослідження – проєктування й створення вебдодатка на мові *Java*.

Предмет дослідження – вебдодаток на мові *Java*.

Метод дослідження – проєктування, структурна і програмна реалізація вебдодатка для пошуку ментора на мові *Java*.

Встановлено, що за допомогою обраних технологій та архітектури можливо створити онлайн платформу пошуку ментора у навчанні**.**

Результати дипломної роботи рекомендується використовувати при розробці онлайн платформи пошуку ментора у навчанні.

ЗМІСТ

ПЕРЕЛІКУМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ВНЗ – вищий навчальний заклад

API – програмний інтерфейс програми (*application programming interface*)

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. (*HyperText Transfer Protocol*)

SOLID – це абревіатура складена з перших літер п'яти базових принципів об'єктно-орієнтованого програмування

JDBC – підключення до бази даних *Java* (*Java Database Connectivity*)

СУБД – система управління базами даних

ВСТУП

У житті кожного було незліченна кількість моментів, коли потрібно було спиратися на когось, кому можна довіряти, за порадою чи допомогою. Незалежно від того, до кого звертається – професора, друга, члена сім'ї, атлетичного тренера чи будь-кого іншого, важливість наставників незаперечна. Ці люди можуть допомогти спрямувати та сформувати теперішню ситуацію та майбутні можливості на краще. Наставник – це той, з ким можна налагодити довгострокові стосунки, зосереджені на побудові росту та розвитку вихованця. Наставник не працює щодня, щоб допомагати вихованцеві приймати рішення, але він завжди присутній, щоб бути у пригоді та запропонувати підтримку, мудрість та навчання. Наставника не слід плутати з тренером. Тренер – це той, хто зосереджується на конкретних сильних і слабких сторонах. Що стосується освіти, студенти, які мають наставників, можуть отримати велику користь. Це пояснюється тим, що наставники можуть допомогти їм орієнтуватися в освітній подорожі студентів та аспірантів, передаючи мудрість та поради. Крім того, наставники можуть зіграти важливу роль у допомозі майбутнім або недавнім випускникам зайняти гарне становище на ринку праці. Вони можуть зв’язати вихованців з людьми у своїй професійній мережі, що може відкрити двері для нових можливостей. Наставник з часом знає сильні та слабкі сторони учня і може зіграти важливу роль, допомагаючи стати найкращою версією себе. Актуальність наставництва в Інтернеті полягає в тому, що учень і ментор мають можливість зв’язатися знаходячись у будь-якій точці світу. Це дає змогу зв’язати людей із різними поглядами, ідеями та думками. Подібним чином, завдяки онлайновим розробкам програмного забезпечення для наставництва, люди тепер можуть обмінюватися найкращими практиками та досвідом у різних країнах та культурах. Наставництво в Інтернеті дає можливість спілкуватися з людьми, до яких учні, певно, б ніколи не мали доступу в живому спілкуванні. Проте є й певні складнощі. Віртуальне спілкування обмежується такими комунікативними сигналами, як мова тіла та міміка. Особливо це стосується таких каналів, як електронна пошта

та телефонні дзвінки. Наставники та вихованці можуть подолати ці проблеми спілкування, використовуючи відеочат, коли це можливо. Це дозволить їм бачити вираз обличчя один одного та мову тіла. Це також формалізує програму, тому вони сприймають її серйозно. Часові зони. Коли учні та наставники не знаходяться в одному місці, цілком можливо, що часові пояси стануть проблемою. Щоб подолати проблеми часового поясу, адміністратори програми можуть допомогти учасникам піти на компроміс під час зустрічей. Ефективні часи можна вибрати, щоб максимізувати стосунки. Це справедливо навіть для найвіддаленіших між собою учасників. У ідеальному світі будь-яка форма технології працювала б постійно. Однак це не так. Технологічні збої можуть розчарувати обох учасників і, можливо, спричинити напружені стосунки. Проблеми з технологіями не завжди можна запобігти, але є можливість до них підготуватися. Наставники та підопічні можуть підготувати плани дій на випадок непередбачених ситуацій, коли виникають проблеми з технологіями. Це дозволить мінімізувати розчарування та дозволить продовжити навчання. Тут електронна пошта та телефонні дзвінки можуть бути корисними формами віртуального наставництва.

Перш ніж ваша працювати над реалізацією ідеї, важливо перевірити різні технології та інструменти. Розкрити глибокі потреби цільової аудиторії. Це допоможе зменшити багато ризиків, пов’язаних з веброзробкою, оскільки можна спроєктувати або випробувати різні сценарії та вибрати оптимальний. Виділення деякого часу для виконання цієї основи окупиться: робота з визначеним і перевіреним набором технологій буде більш продуктивною та ефективною. Дизайн продукту є вирішальним кроком у процесі веброзробки. Часто зустрічаються барвисті інтерфейси, які привертають увагу користувача, але найуспішніші конструкції мають дуже скромне начало – каркаси. Починаючи з низькоякісних каркасних програм чи макетів, це ефективний спосіб перевірити, чи резонують ідеї з кінцевими користувачами, а вебпрограма інтуїтивно зрозуміла для використання. Важливо не забувати про елементи успішного процесу веброзробки: забезпечення якості та підготовка до випуску. Забезпечення якості – для того, щоб продукт мав відмінну якість, слід його

протестувати. Ефективне забезпечення якості – це не етап тестування десь у кінці процесу розробки програмного забезпечення, а скоріше постійний процес, інтегрований із зусиллями з розробки. Підготовка до випуску – зазвичай, плавно запустити продукт і спланувати позитивні відгуки від користувачів – приплив трафіку. Перш ніж розгортати вебпрограму, потрібно переконатись, що продукт працює за планом.

Мета і завдання виконання дипломної роботи – дослідити й розробити онлайн платформу для пошуку ментора у навчанні. Об’єкт дослідження – проєктування й створення вебдодатка на мові *Java*. Предмет дослідження – вебдодаток на мові *Java*. Метод дослідження – проєктування, структурна і програмна реалізація вебдодатка для пошуку ментора на мові *Java*. Наукова новизна отриманих результатів полягає у розробці онлайн платформи для пошуку ментора у навчанні з архітектурою *MVC*, реалізації роботи з базою даних за допомогою *hibernate* та реалізацією користувацького інтерфейсу за допомогою *thymeleaf*. Розроблений додаток дипломної роботи рекомендується використовувати при розробці онлайн платформи пошуку ментора у навчанні, а також для розробки та проєктування вебдодатків на мові *Java*.

РОЗДІЛ 1
АКТУАЛЬНІСТЬ ОНЛАЙН НАВЧАННЯ З МЕНТОРОМ

Подібно до того, як технологія забезпечила більшу гнучкість на робочих місцях, у школах та в цілому в суспільстві – від платформ для відеоконференцій (наприклад, *Zoom, Google Hangouts*) до інструментів для онлайн-співпраці (наприклад, *Slack, Microsoft Teams*) – це дозволило збільшити гнучкість роботи наставників, а вихованці можуть отримувати та залишатися на зв'язку. Це навіть відкриває деякі нові можливості, які неможливо здійснити, якщо тримати речі строго віч-на-віч. Дозволило мати програми наставництва, орієнтовані на зв’язок людей у різних регіонах – на національному та міжнародному рівнях. Це стає можливим лише завдяки технологіям та програмі наставництва в Інтернеті. Навіть для наставників і вихованців, які вважають за краще зустрічатися особисто, бувають випадки хвороби, поїздки, насичений графік, коли гнучкість технологій забезпечує взаємозв'язок, навіть якщо приватна зустріч неможлива. Незважаючи на те, що наставництво в Інтернеті має багато переваг, важливо також усвідомлювати проблеми, які постає перед віртуальною програмою. Спілкування через обмеження читання мови тіла та міміки може призвести до відсутності високого рівня взаєморозуміння. На щастя, це можна подолати, надавши наставникам та учням доступ до різноманітних комунікативних інструментів та можливостей використовувати відео, коли це можливо. Також труднощі включають управління великими обсягами повідомлень, чітке висловлення ідей та почуттів, отримання доступу до технологій та постати перед технічними труднощами.

Хороший наставник може допомогти вихованцеві стати ефективнішим на роботі, засвоїти нові навички, набути більшої впевненості та прийняти кращі

рішення щодо загального кар’єрного зростання. Наставництво – це позитивні стосунки, що підтримують, заохочуючи молодих людей розвиватися на повну силу. Наставництво багатогранне; воно може бути формальним або неформальним і може змінюватися та розвиватися в міру того, як змінюються потреби вихованця. Наставник може бути зразком для наслідування, тренером, радником та надійним ресурсом. Наставники піклуються і запевняють своїх вихованців, що вони не самі вирішують щоденні виклики.

Проблема в тому, що знайти професійного наставника може бути важко. Зрештою, наставництво – це трудомісткий процес; отже, для людини, яка шукає наставника, грубо вважати, що хтось матиме час, терпіння та інтерес допомогти без принаймні якоїсь компенсації, проте існують онлайн платформи для пошуку ментора з різних напрямків та галузей. Однією з чудових речей наставництва є людський зв’язок, який створюється двома або більше людьми. Наставництво вимагає продуманих розмов, мозкового штурму та взагалі глибокого занурення з іншими людьми в спільне питання.

1.1. Навчання з ментором

Термін «менторство» означає «повчання». З англійської слово *mentor* можна перевести, як напутник, радник, керівник, вихователь. Менторство можна представити як спосіб передачі досвіду, в якому ментор є наставником, радником, який ділиться своїм досвідом для розвитку, зростання і підтримки менш досвідчених колег. Ментор є фахівцем, що досяг певних результатів у своїй професії, готовим навчати інших, тобто має вид співпраці молодшого і старшого колеги. Менторство поширений вид навчання у сфері освіти. Даний метод застосовується в роботі у вищих навчальних закладах, проте не часто й швидше неформально, з закріпленням ролі «ментора» за викладачем. Обираючи даний метод у ВНЗ, викладач повинен враховувати недоліки та переваги менторства. Відкрита й доброзичлива позиція, широке обговорення задач дозволяють зробити спілкування під час навчання більш щирим, студент буде відчувати себе впевненіше, пропонувати незвичайні способи розв’язання питання. Але не варто

забувати, що такий стиль передачі інформації може призвести до пониження дисципліни, відповідальності й своєчасним виконанням завдань.

Згідно з формою організації занять зі студентами виділяють три групи:

- Індивідуальні;
- Колективно-групові;
- Індивідуально-колективні;

Згідно з цією класифікацією менторство належать до групи індивідуальних занять [1]. Однак на практиці можливі різні форми реалізації цього методу: як з однією людиною, так і з групою осіб. Діяльність, здійснювану ментором в процесі навчання, можна схарактеризувати основними кроками:

- Аналіз і оцінка перспективи розвитку учня;
- Допомога в постановці цілей і розробці шляхів їх досягнення;
- Демонстрація свого прикладу в реалізації завдань;
- Психологічна підтримка;
- Оснащення підопічного корисними ресурсами;
- Видача йому практичних завдань, які допомагають освоїти професійну діяльність, контроль їх виконання, оцінка та рекомендації для подальшої діяльності;

В процесі навчання, наставником приділяється особлива увага індивідуальним завданням, які він видає студентам виходячи зі свого бачення шляху їх розвитку, а також подолання труднощів, які вимагають ретельного опрацювання. Так, якщо викладач стає ментором у кількох студентів, то для кожного він розробляє свій комплекс завдань, враховуючи індивідуальні знання, уміння і навички кожного зі студентів.

В список обов'язків ментора входить функція психологічної підтримки, що дозволяє, крім обговорення ходу виконання завдань, обговорювати успішність навчального процесу, що мотивувало б або які фактори заважають роботі, страхи й сумніви, що хвилюють учня. Таким чином, ментор встановлює індивідуальний зв’язок зі студентом, що робить процес навчання більш ефективним.

Метод менторства складається з реалізації чотирьох основних етапів:

- Підготовка;

- Обговорення;
- Уповноваження;
- Завершення;

Початкові етапи пов'язані з підготовкою та обговоренням процесу роботи, які необхідні для створення довірчої атмосфери, постановки завдань і цілей, злагоджених дій, обговорення розкладу зустрічей, утворення уявлень в обох сторін про взаємні очікування і досягнення процесу навчання. Після підготовки настає етап виконання роботи, в процесі якого ментор стимулює професійне зростання студента, допомагає йому, коментуючи дії студента, оцінює виконання наданих завдань та редагує їх виконання. На даному етапі особливо важливо, наскільки довірчі відносини були збудовані на першому етапі роботи та чи виходить у ментора створити щирий і відкритий діалог зі своїм студентом. Після виконання роботи, на етапі завершення, обидва учасники процесу надають один одному зворотний зв'язок про виконану роботу, оцінюють її в письмовій або усній формі.

Також менторство можна розділити на типи за формою отримання знань від ментора:

- Індивідуальне наставництво;
- Групове наставництво;
- Наставництво з боку однолітків;
- Дистанційне або електронне наставництво;
- Зворотне наставництво;
- Швидке наставництво;

Індивідуальне наставництво. Цей тип наставництва є найбільш традиційним серед усіх видів наставництва. У цьому виді наставництва беруть участь лише наставник і вихованець, і це, як правило, досвідченіша особа в парі з менш досвідченим або значно молодшим вихованцем. Групове наставництво: у цій моделі один або кілька наставників працюють із групою наставників. Школи та молодіжні програми часто застосовують цю модель, оскільки може не вистачити часу чи ресурсів, щоб мати одного наставника для кожного учасника. Наставництво з боку однолітків: учасники цієї моделі мають одну і ту ж роль або

відділ або мають спільний або подібний досвід у своєму професійному чи особистому житті. Ці однолітки поєднуються, щоб запропонувати підтримку один одному. Це можуть бути групові або індивідуальні наставницькі стосунки. Дистанційне або електронне наставництво: при наявності передових технологій відносини наставництва більше не повинні бути віч-на-віч. Використовуючи програмне забезпечення в Інтернеті або навіть електронну пошту, учасники цього типу наставництва можуть зв'язуватися практично, не втрачаючи особистого зв'язку. Зворотне наставництво: ці відносини наставництва відхиляються від традиційної моделі. Замість того, щоб старший фахівець навчав молодшого працівника, молодший працівник навчає професіонала. Швидке наставництво: цей тип наставництва – це зустрічі які зазвичай відбуваються в рамках корпоративного заходу чи конференції. Вихованець проводить серію індивідуальних розмов із набором різних наставників і зазвичай переходить від одного наставника до наступного після короткої зустрічі. Вихованець повинен бути з готовим списком запитань для поради професіоналів вищого рівня.

Менторство може бути корисним методом роботи як у процесі навчання, так і в період проходження навчальної та виробничої практики, а також під час написання випускної кваліфікаційної роботи.

1.2. Онлайн платформи для пошуку ментора

У менторстві однією з найбільших проблем студентів є пошук наставника, особливо коли професійні кола знайомств обмежені. Проте цю проблему вирішили за допомогою інформаційних технологій, розробивши онлайн платформи пошуку менторів.

Можна виділити основні типи онлайн-наставницьких платформ:

- Платформи наставництва для стартапів;
- Платформи бізнес-наставництва;
- Академічні платформи наставництва;
- Платформи для кар'єрного наставництва;
- Програмування наставницьких платформ;

– Дизайн наставницьких платформ;
– Платформи з питань мистецтва;
– Менторські програмні додатки;

Розглянемо деякі з цих платформ.

GrowthMentor

GrowthMentor – це перевірена платформа наставництва. Це означає, що наставники були ретельно відібрані – з понад 2000 претендентів, прийнято менше ніж 10% – для забезпечення найвищих стандартів надання послуг у менторстві. Приблизно 80% наставників пропонують свій час безплатно, а платні тарифи рекомендується залишати $20-50 на годину, для заохочення користувачів до співпраці. Інтерфейс платформи наведено на рис. 1.1.

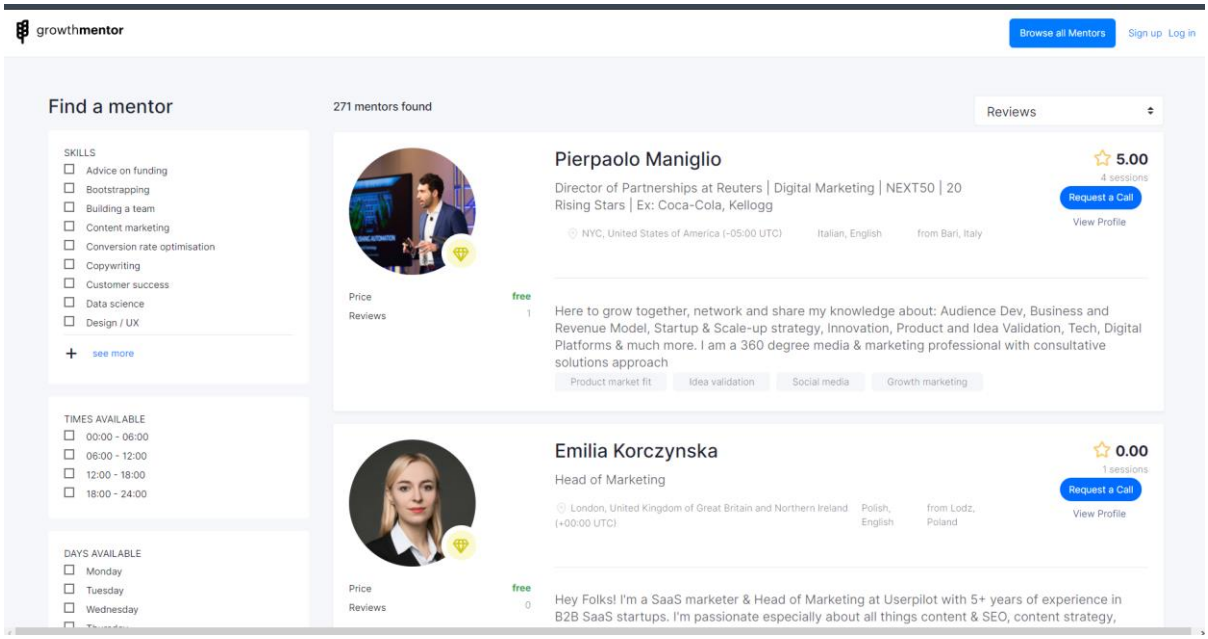

Рис. 1.1. Інтерфейс сторінки вибору менторів онлайн платформи *GrowthMentor*

Усі наставники повинні пропонувати свій час безплатно, поки не отримають три відгуки, а їх погодинна ставка не може перевищувати $99. Ця політика перетворюється на унікальну спільноту експертів, сегментовану за вмінням, галуззю, знаннями інструментів та мовою, і класифікується за оцінками оглядів, роблячи *GrowthMentor* доступним ресурсом для тих, хто намагається розвивати бізнес, підтверджувати ідею чи розвивати кар'єру.

MicroMentor

MicroMentor – це відкрита спільнота з понад 25 000 наставників та 70 000

учнів, яка поєднує учнів та наставників, дозволяючи їм визначити, чи добре вони підходять один одному для подальшої співпраці. Інтерфейс платформи наведено на рис. 1.2. Сильна сторона цієї моделі полягає в тому, що кожен може стати наставником чи вихованцем, роблячи цю мережу доступною для мікро-, малого та зростаючого бізнесу та для будь-якої людини, яка бажає стати наставником.

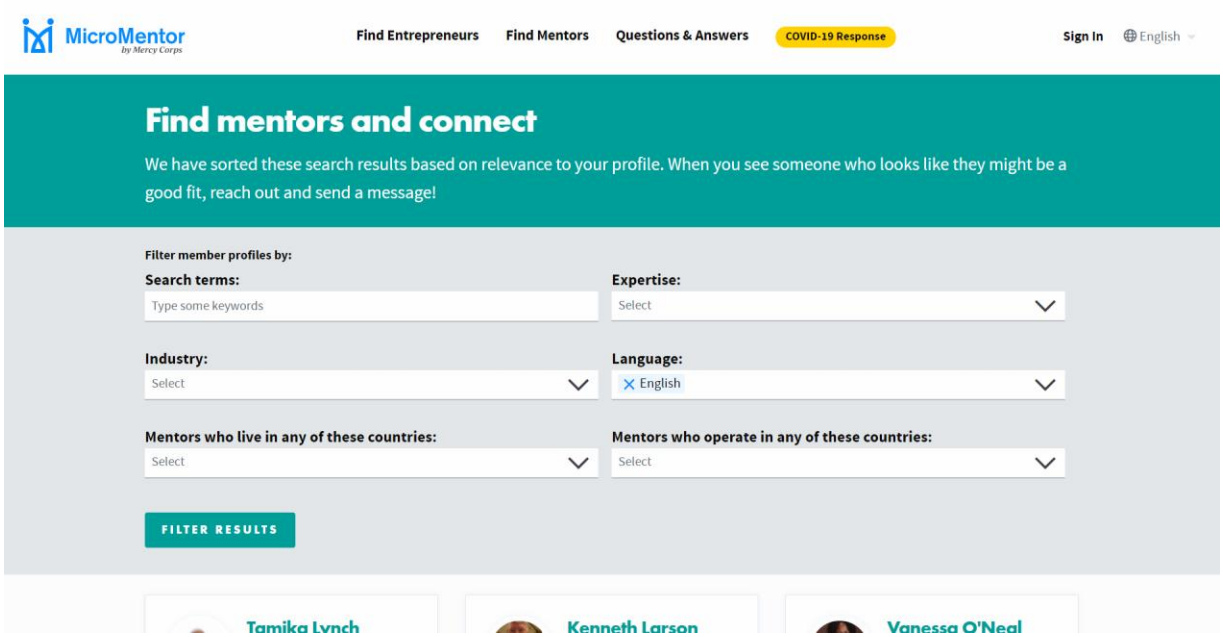

Рис. 1.2. Інтерфейс сторінки вибору менторів онлайн платформи *MicroMentor*

Наставників можна відфільтрувати за знаннями, галузями, роками бізнесу, мовою та регіоном. Однак не існує простого способу поставити їх у рейтинг чи перевірити, чи вони вже спілкувались з вихованцями, і якщо так, то наскільки корисними були їх поради.

Clarity

Clarity – це онлайн-платформа наставництва, яка дозволяє студентам замовляти дзвінки з наставниками на певний час. Після підтвердження сеансу надається номер телефону та код доступу для конференц-дзвінка, а студент попередньо приєднується на запитуваний час наставництва. Інтерфейс платформи наведено на рис. 1.3. Усі сеанси повинні відбуватися через телефон, оскільки кінцева ціна розраховується щохвилини.

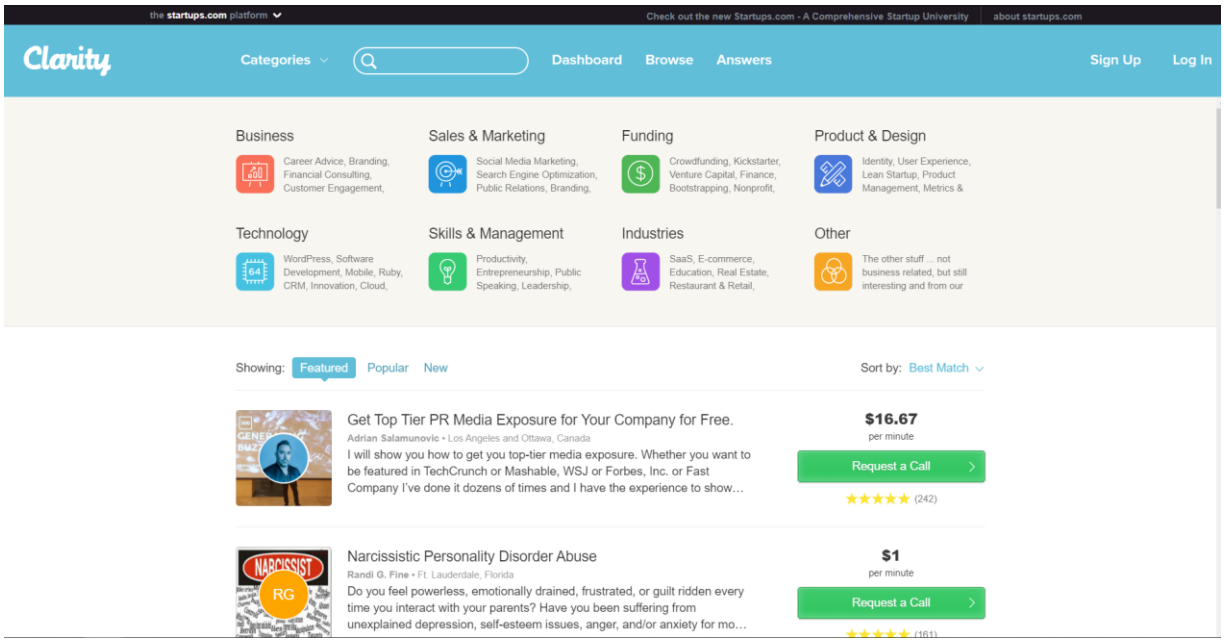

Рис. 1.3. Інтерфейс сторінки вибору менторів онлайн платформи *Clarity*

Мережа має велику кількість доступних наставників, близько 10 000. З іншого боку, головним недоліком є висока вартість більшості наставників. Однак, якщо бюджет дозволяє, *Clarity* може бути варіантом, який варто розглянути.

SCORE

SCORE – це мережа волонтерів, досвідчених наставників бізнесу, спрямована на те, щоб допомогти малому бізнесу знайти та розв'язати проблему, щоб рухатися далі, рости та досягти своїх цілей. Платформа надала освіту та наставництво понад 11 мільйонам підприємців. *SCORE* є некомерційною організацією та ресурсним партнером Адміністрації малого бізнесу США (*SBA*). Завдяки підтримці від *SBA* (*Small Business Administration*) та завдяки волонтерам, *SCORE* має можливість зробити більшість пропозицій безплатними. Інтерфейс платформи наведено на рис. 1.4.

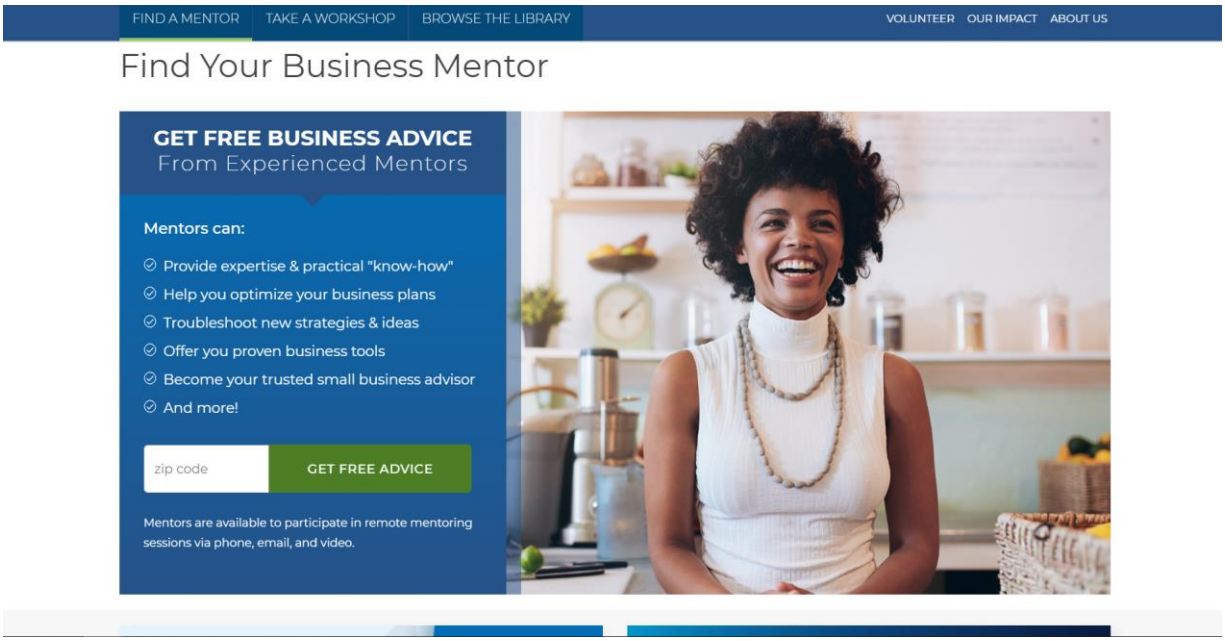

Рис. 1.4. Інтерфейс сторінки вибору менторів онлайн платформи *SCORE*

SCORE надає широкий спектр послуг досвідченим бізнесменам та початківцям. Підприємці можуть отримати доступ до безплатного конфіденційного наставництва у бізнесі особисто у більш ніж 250 місцевих розділах або віддалено через електронну пошту, телефон та відео. Наставники *SCORE*, усі експерти в галузі підприємництва та суміжних галузей, постійно зустрічаються зі своїми клієнтами малого бізнесу, щоб надавати постійні поради та підтримку.

SCORE регулярно пропонує безплатні онлайн-семінари на теми, починаючи від стратегій старту та закінчуючи маркетингом та фінансами. Учасники можуть переглядати вебінари в прямому ефірі або переглядати записи в Інтернеті у свій час. Платформа також пропонує інтерактивні курси на замовлення, щоб користувач міг пройти кожен модуль у своєму власному темпі. Користувачі також можуть скористатися великою колекцією шаблонів, контрольних списків, блогів, відео, інфографіки тощо. *SCORE* прагне надати найбільш актуальний та актуальний освітній контент, який допоможе власникам малого бізнесу та підприємцям досягти успіху.

Велика кількість платформ пошуку ментора та кількісні показники менторів та учнів на кожній з платформ свідчить про їх актуальність та попит серед людей різних спеціальностей та різного рівня професійних навичок.

1.3. Завдання на розробку

Онлайн-платформа використовується для забезпечення цілого ряду послуг, доступних в Інтернеті, включаючи торгові майданчики, пошукові системи, соціальні медіа, точки творчого контенту, магазини додатків, послуги зв'язку, платіжні системи, послуги, що включають так званий «спільний» або “концертна” економіка, і багато іншого. Онлайн-платформа визначається як цифрова послуга, яка полегшує взаємодію між двома або більше різними, але взаємозалежними групами користувачів (будь то фірми чи приватні особи), які взаємодіють через послугу через Інтернет. Інтернет-ринок, на якому одна сторона контактує з іншою, наприклад, покупці та продавці. Прикладами є *eBay, Craigslist, Amazon Marketplace, Airbnb* та *Uber*. Інтернет-система може бути повністю автономною, або вона може дозволяти стороннім програмам підключатися через інтерфейс програмування платформи (*API*).

Усі розглянуті онлайнові платформи пошуку ментора об’єднує певний функціонал. Тож онлайнова платформа пошуку ментора у навчанні повинна забезпечувати для користувача наступний набір функцій:

- Перегляд всіх менторів;
- Фільтрація менторів;
- Реєстрація профілю ментора;
- Перегляд профілю ментора;

Перегляд всіх доступних у базі даних менторів, дозволить користувачеві ознайомитись з усім списком менторів наявних на платформі. Фільтрація менторів за їх характеристиками, забезпечить зручний пошук наставника у вказаній галузі або декількох галузях, також фільтрація дозволить сортувати менторів за рейтингом для пошуку більш професійного й відповідального ментора, за думкою користувачів. Фільтрація за ціною, щоб користувач міг підібрати ментора за своїм бюджетом. Реєстрація профілю ментора дозволить, охочий стати наставником, додавати свої профілі. Перегляд профілю ментора дозволить більш детально переглянути інформацію про ментора, а саме його спеціальність, ціну, контактні дані та опис свого досвіду.

Для забезпечення користувачів необхідним функціоналом необхідно реалізувати вебдодаток та забезпечити *API*. Реалізувавши функції для користувачів онлайнової платформи пошуку ментора у навчанні, об’єднає споживачів тобто людей, що шукають наставника, та виробників послуг, тобто людей з професійними навичками, які можуть та мають бажання навчати та передати свій досвід іншим.

1.4. Висновки до розділу

Наставництво – це взаємні відносини за власним бажанням, які найчастіше виникають між старшим та молодшим працівником з метою зростання, навчання та розвитку кар’єри вихованця. Часто наставник та вихованець є колегами, і тут робиться акцент на організаційних цілях, культурі, цілях кар'єри, порадах щодо професійного розвитку та балансу роботи та життя. Ефективні наставники часто виступають в якості зразків для наслідування та надають вказівки, що допомагають їм досягти своїх цілей.

Існує багато платформ пошуку ментора, за допомогою яких людина може знайти собі наставника та працювати над підвищенням ефективності свого кар’єрного просування та швидшим підйомом по кар’єрних сходах. Існують платформи на яких можна обирати наставників з різних напрямків, а також існують платформи з наставниками з певної галузі знань.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-ДОДАТКУ

За своєю природою вебдодатки є розподіленими додатками, це означає, що вони працюють на двох і більше комп'ютерах і спілкуються на сервері або через мережу. Вебдодатки використовують за допомогою веббраузера й це робить їх популярними завдяки простоті користуванням браузером для користувача. До однієї онлайн платформи можуть отримати доступ і користуватися мільйони людей. Як і комп'ютерні програми, вебпрограми складаються з багатьох частин і часто містять мініпрограми, деякі з яких мають користувальницькі інтерфейси, а деякі з них взагалі не потребують графічного інтерфейсу для користувача. Крім того, онлайн платформи часто потребують додаткової верстки відображень та створення стилів або мови програмування для створення динамічних сторінок.

Вебсторінки написані з застосуванням *HTML* та перекладаються веббраузером. Вебсторінки можуть бути як динамічними, так і статичними. На статичних сторінках відображається кожен раз той самий вміст, коли їх переглядають. Динамічні сторінки мають вміст, який може змінюватися кожного разу, коли вони отримують доступ до них. Мови *ASP* або *JSP* зазвичай використовуються при написанні сценаріїв динамічних сторінок, а також для цього застосовуються мови *Perl* і *PHP*. Сценарії на сторінках запускають функції на сервері, які повертають такі дані, як дата та час, а також інформацію про базу даних. Вся інформація повертається як *HTML*-код, тому, коли браузер обробляє сторінку, все, що потрібно зробити браузеру – це перекласти *HTML*. Потрібно пам'ятати, що вебсторінка – це не те саме, що вебсайт. Вебсайтом називається сукупність сторінок. А окремий документ на *HTML* – це сукупність сторінок. Крім того, у багатьох додатків застосовують лише мову програмування *Java*.

Зазвичай вебпрограми написані на *Java* не працюють безпосередньо на сервері. Вони працюють у вебконтейнері на сервері. У контейнері розгортається необхідне середовище для виконання вебпрограм. Контейнер призначений для вебпрограм *Java*, а саме *JVM* (віртуальна машина *Java*) для локально запущених програм *Java* та працює у *JVM*.

Якщо онлайн платформа має дані, які час від часу змінюється – потрібна база даних. Бази даних, щоб зберігати зображення, статті, різні типи інформації та макети сторінок. Зберігаючи дані в базі даних, легко отримати до них доступ в Інтернеті за допомогою серверних мов, таких як *JSP* або *PHP*. Наприклад, сайт пов’язаний з продуктами, ціни на які постійно змінюються. У таких видах вебсайтів можна застосовувати БД, які заощаджують час, швидко виконуючи оновлення.

2.1. Аналіз архітектури вебдодатка

Архітектура вебдодатків – це схема взаємодії між компонентами вебпрограми. Спосіб планування цієї взаємодії визначає стійкість, продуктивність та безпеку майбутнього вебдодатку. За розподілом логіки додатку виділяють архітектуру: монолітна, мікросервісна.

Мікросервісна архітектура – це стиль архітектурної розробки, в якому додаток складається з менших служб, які взаємодіють між собою безпосередньо за допомогою невеликих протоколів, таких як *HTTP*.

Принцип мікропослуг – єдина відповідальність. У ньому зазначено, що окрема одиниця, клас, метод, або мікросервіс, повинна нести одну і тільки одну відповідальність. Кожна мікрослужба може використовувати різні технології, засновані на бізнес-вимогах. Мікросервіси повинні розроблятися з урахуванням випадків відмов. Мікросервіси повинні використовувати переваги цієї архітектури, і вихід з ладу одного мікросервісу не повинен впливати на всю систему. Ця архітектура має ряд переваг. Легко змінювати код – кожен сервіс порівняно невеликий, тому його легше зрозуміти та змінити. Кожен мікросервіс порівняно невеликий: додаток запускається швидше, що робить розробників

більш продуктивними та пришвидшує розгортання. Проте в мікросервісній архітектурі є й недоліки. Будучи розподіленою системою, вона набагато складніша, ніж монолітні програми. складність якої зростає зі збільшенням кількості мікропослуг. З архітектурою мікросервісів повинні працювати кваліфіковані розробники, які можуть ідентифікувати мікросервіси та управляти їх взаємозв'язками. Мікросервіси менш захищені у порівнянні з монолітними додатками через взаємозв'язок між службами мережею. Налагодження важке, оскільки елемент керування перетікає багато мікросервісів, і вказати, чому і де саме сталася помилка, є складним завданням.

Якщо всі функціональні можливості проєкту існують в одній кодовій базі, то ця архітектура відома як монолітна. Додаток розробляється на різних рівнях, таких як презентація, сервіс та контролер, а потім розгортаємо цю базу коду як один файл *jar* / *war*. Це не що інше, як монолітна програма, де «моно» представляє єдину базу коду, що містить усі необхідні функціональні можливості.

Реалізацією монолітної архітектури слугує шаблон *MVC Model-View-Controller*. У галузі веброзробки *Model-View-Controller* є однією з найбільш обговорюваних моделей дизайну у світі вебпрограмування. Модельні конструкції, засновані на архітектурі *MVC*, відповідають шаблону дизайну *MVC*, і вони відокремлюють логіку програми від інтерфейсу користувача при розробці програмного забезпечення. Як випливає з назви, шаблон *MVC* має три шари: модель, представлення, контролер.

Рівень модель відповідає за логіку, пов’язану з даними, з якою працює користувач. Тобто представляє як дані, що передаються між рівнями представлення та контролер, так і будь-які інші дані, пов’язані з бізнес-логікою. Рівень представлення реалізує всю логіку інтерфейсу програми. Контролери приймають запити від користувача та розподіляють логіку між рівнями модель і представлення для обробки всієї бізнес-логіки та вхідних запитів, маніпулювання даними за допомогою рівня модель та взаємодії з представленнями для надання кінцевого результату. Архітектура *MVC* надає переваги програмісту при розробці додатків. Кілька розробників можуть одночасно працювати з трьома шарами (*Model*, *View* та *Controller*). Пропонує покращену масштабованість, що доповнює

здатність програми рости. Оскільки компоненти мають низьку залежність один від одного, їх легко обслуговувати. Модель може бути повторно використана за допомогою декількох подань, що забезпечує повторне використання коду. Розширення та тестування програми досить просте.

2.2. Аналіз баз даних

База даних – це організована колекція структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді в комп’ютерній системі. База даних зазвичай контролюється системою управління базами даних (СУБД). Дані та СУБД разом із пов'язаними з ними програмами називають системою баз даних, часто скорочено до бази даних.

Оперативна база даних. Тип бази даних, яка створює та оновлює базу даних у режимі реального часу. В основному він призначений для виконання та обробки щоденних операцій з даними в декількох компаніях. Наприклад, організація використовує оперативні бази даних для управління щоденними операціями.

База персональних даних. Збір та зберігання даних у системі користувача визначає персональну базу даних. Ця база даних в основному розроблена для одного користувача. Переваги бази персональних даних: просто і легко працювати. Займає менше місця для зберігання, оскільки є невелика за розміром

Ієрархічні бази даних. Це тип бази даних, що зберігає дані у вигляді вузлів взаємовідносин батьків та дітей. Дані зберігаються у вигляді записів, які пов’язані за допомогою посилань. Кожен дочірній запис у дереві міститиме лише одного з батьків. З іншого боку, кожен батьківський запис може мати кілька дочірніх записів.

Мережеві бази даних. Це база даних, яка, як правило, слідує мережевій моделі даних. Тут представлення даних здійснюється у вигляді вузлів, з'єднаних за допомогою зв'язків між ними. На відміну від ієрархічної бази даних, вона дозволяє кожному запису мати кілька дочірніх та батьківських вузлів, щоб сформувати узагальнену структуру графіка.

Об'єктноорієнтовані бази даних. Тип бази даних, що використовує підхід

до об'єктної моделі даних для зберігання даних у системі баз даних. Дані представляються та зберігаються як об’єкти, подібні до об’єктів, що використовуються в об’єктноорієнтованій мові програмування. Переваги об’єктних баз даних:

– *ODBMS* забезпечують постійне зберігання об'єктів. Уявіть, що ви створюєте об’єкти у своїй програмі та зберігаєте їх, як у базі даних, і читаєте з бази даних;

– У типових СУБД існує шар об'єктно-реляційного відображення, який відображає схеми баз даних з об'єктами в коді. Читання та зіставлення даних бази даних об’єктів з об’єктами здійснюється безпосередньо без будь-якого інструмента *API*. Звідси швидший доступ до даних та краща продуктивність;

– Бази даних об’єктів повертають стійкі об’єкти. Об'єкти можна зберігати у постійному сховищі назавжди;

Недоліки баз даних об’єктів:

– Об’єктні бази даних не такі популярні, як СУБД;

– Не так багато мов програмування підтримують об'єктні бази даних;

– СУБД мають *SQL* як стандартну мову запитів;

– Об’єктні бази даних не мають стандарту;

– Об’єктні бази даних важко засвоїти для не програмістів;

Хмарна база даних. Тип бази даних, де дані зберігаються у віртуальному середовищі та виконуються на платформі хмарних обчислень. Хмарна база даних надає користувачам різні послуги хмарних обчислень (*SaaS*, *PaaS*, *IaaS* тощо) для доступу до бази даних.

Реляційна база даних. Ця база даних базується на реляційній моделі даних, яка зберігає дані у вигляді рядків (кортеж) і стовпців (атрибути) і разом утворює таблицю (відношення). Реляційна база даних використовує *SQL* для зберігання, маніпулювання, а також для обслуговування даних. Кожна таблиця бази даних містить ключ, який робить дані унікальними для інших.

Синтаксис *SQL* може дещо відрізнятися залежно від того, яка СУБД використовується. Було стисло розглянуто популярні СУБД.

MySQL – найпопулярніша база даних *SQL* з відкритим кодом. Зазвичай

використовується для розробки вебдодатків. Основними перевагами *MySQL* є те, що він простий у використанні та має велику спільноту розробників, які можуть допомогти відповісти на запитання.

PostgreSQL – це база даних *SQL* з відкритим кодом, яка не контролюється жодною корпорацією. Зазвичай він використовується для розробки вебдодатків. *PostgreSQL* поділяє багато однакових переваг *MySQL*. Він простий у використанні, недорогий, надійний і має велику спільноту розробників. Він також надає деякі додаткові функції, такі як підтримка зовнішнього ключа, не вимагаючи складної конфігурації. Основним недоліком *PostgreSQL* є те, що він може бути повільнішим за продуктивністю, ніж інші бази даних, такі як *MySQL*. Він також трохи менш популярний, ніж *MySQL*.

БД *Oracle*. Призначена для великих додатків, особливо в банківській галузі. *Oracle* пропонує потужне поєднання технологій та комплексних, попередньо інтегрованих бізнес-додатків, включаючи основні функціональні можливості, створені спеціально для банків. Основним недоліком використання *Oracle* є те, що він не безплатний у використанні, як його конкуренти з відкритим кодом.

SQL Server. *Microsoft* володіє *SQL Server*. Великі корпоративні програми здебільшого використовують *SQL Server*. Корпорація Майкрософт пропонує безплатну версію початкового рівня під назвою *Express*, але вона може стати дуже дорогою під час масштабування програми.

SQLite – це популярна база даних *SQL* з відкритим кодом. Може зберігати всю базу даних в одному файлі. Однією з найважливіших переваг цього є те, що всі дані можна зберігати локально, не підключаючи базу даних до сервера. *SQLite* – популярний вибір для баз даних у мобільних телефонах, *MP*3-плеєрах, приставках та інших електронних гаджетах.

2.3. Робота з базою даних у *Java*

JDBC (*Java Database Connectivity*) – це *Java API*, який управляє підключенням до бази даних, видачею запитів і команд та обробкою наборів результатів, отриманих з бази даних. Інтерфейс *JDBC* складається з двох шарів:

API JDBC підтримує зв'язок між додатком *Java* та менеджером *JDBC*. Драйвер *JDBC* підтримує зв'язок між менеджером *JDBC* і драйвером бази даних. *JDBC* – це загальний *API*, з яким взаємодіє код програми. Під цим знаходиться *JDBC*-сумісний драйвер для бази даних, яка використовується. Архітектурний огляд *JDBC* наведено на рис. 2.1.

Рис. 2.1. Спрощена архітектура *JDBC* у шарі стійкості *Java*

Етапи підключення до бази даних за допомогою *JDBC* такі:

- Встановити або знайти базу даних, до якої потрібно отримати доступ;
- Включити бібліотеку *JDBC*;
- Використати бібліотеку *JDBC* для підключення до бази даних;
- Використати з'єднання для виконання команд *SQL*;
- Завершити з’єднання;

JDBC Driver – це програмний компонент, який дозволяє додатку *Java* взаємодіяти з базою даних.

Робота з базою даних за допомогою *ORM Hibernate*. *Hibernate* – це фреймворк *Java*, який спрощує розробку програми *Java* для взаємодії з базою даних. Це інструмент з відкритим вихідним кодом, легкий, *ORM* (*Object Relational Mapping*). *Hibernate* реалізує специфікації *JPA* (*Java Persistence API*) для збереження даних. Засіб *ORM* спрощує створення даних, маніпулювання ними та доступ до даних. Це техніка програмування, яка відображає об’єкт із даними, що зберігаються в базі даних, див. рис. 2.2.

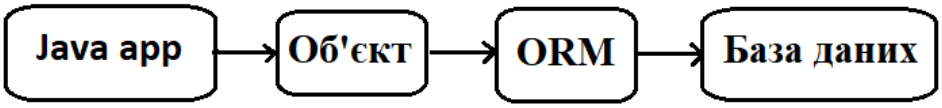

Рис. 2.2. Схема роботи додатку з *ORM*

Засіб *ORM* внутрішньо використовує *API JDBC* для взаємодії з базою даних. *API* стійкості *Java* (*JPA*) – це специфікація *Java*, яка забезпечує певну функціональність та стандарт для інструментів *ORM*. Пакет *javax.persistence* містить класи *JPA* та інтерфейси.

Переваги *hibernate*:

Легкий та відкритий код. *Hibernate framework* є відкритим кодом за ліцензією *LGPL* і легкий.

– Продуктивність *Hibernate framework* є швидкою, оскільки кеш-пам’ять використовується всередині фреймворку сплячого режиму. Існує два типи кеш-пам’яті в *Hibernate* кеш-пам’яті першого рівня та кешу другого рівня. Кеш-пам’ять першого рівня ввімкнено за замовчуванням;

– Незалежний запит до бази даних. *HQL* (*Hibernate Query Language*) – це об’єктноорієнтована версія *SQL*. Він генерує незалежні запити до бази даних. Тож вам не потрібно писати запити до конкретної бази даних. Якщо для проєкту буде змінено базу даних, нам також потрібно змінити запит *SQL*, що призведе до проблеми обслуговування;

– Автоматичне створення таблиці. *Hibernate* забезпечує можливість автоматичного створення таблиць бази даних. Тому нема потреби створювати таблиці в базі даних вручну;

– Спрощує складне приєднання. Отримати дані з декількох таблиць легко;

– Надає статистику запитів та статус бази даних. *Hibernate* підтримує кеш запитів та надає статистику щодо запитів та стану бази даних;

Недоліки *Hibernate*:

– *Hibernate* генерує багато операторів *SQL* під час виконання на основі опису об’єктів, тому він трохи повільніший, ніж *JDBC*;

– Не допускає декількох вставок. *Hibernate* не дозволяє деякі запити, які підтримуються *JDBC*. Наприклад, не дозволяє вставляти кілька об'єктів (постійні дані) в одну таблицю за допомогою одного запиту;

– Розробник повинен написати окремий запит, щоб вставити кожен об’єкт. Доцільно використовувати чистий *JDBC* для пакетної обробки, оскільки ефективність *Hibernate* не є ефективною в пакетній обробці;

2.4. Сервлети

Наразі існує необхідність створення динамічних вебсторінок, тобто тих, які мають можливість змінювати вміст сайту відповідно до часу або здатні генерувати вміст відповідно до запиту, отриманого клієнтом. За допомогою *Java* існує також спосіб генерувати динамічні вебсторінки, і це *Java Servlet*.

Сервлети – це програми *Java*, які працюють на вебсервері або сервері додатків із підтримкою *Java*. Вони використовуються для обробки запиту, отриманого від вебсервера, обробки запиту, отримання відповіді, а потім надсилання відповіді назад на вебсервер.

Властивості сервлетів:

– Сервлети працюють на стороні сервера;

– Сервлети здатні обробляти складні запити, отримані від вебсервера;

Виконання сервлетів передбачає шість основних етапів:

– Клієнти відправляють запит на вебсервер;

– Вебсервер отримує запит;

– Вебсервер передає запит відповідному сервлету;

– Сервлет обробляє запит і генерує відповідь у вигляді виводу;

– Сервлет надсилає відповідь назад вебсерверу;

– Вебсервер надсилає відповідь клієнту, і клієнтський браузер відображає її на екрані;

Розширення на стороні сервера – це не що інше, як технології, які використовуються для створення динамічних вебсторінок. Власне, для забезпечення можливості динамічних вебсторінок вебсторінкам потрібен контейнер або вебсервер. Щоб задовольнити цю вимогу, незалежні провайдери вебсерверів пропонують деякі власні рішення у формі *API* (інтерфейс програмування програм). Ці *API* дозволяють нам створювати програми, які

можуть працювати з вебсервером. У цьому випадку *Java Servlet* також є одним із компонентних *API Java Platform Enterprise Edition*, який встановлює стандарти для створення динамічних вебдодатків на *Java*.

Переваги *Java*-сервлету:

– Сервлет швидший за *CGI*, оскільки він не передбачає створення нового процесу для кожного нового отриманого запиту;

– Сервлети, написані на *Java*, не залежать від платформи;

– Видаляє накладні витрати на створення нового процесу для кожного запиту, оскільки сервлет не запускається в окремому процесі. Існує лише один екземпляр, який обробляє всі запити одночасно. Це також економить пам'ять і дозволяє сервлету легко керувати станом клієнта;

– Це компонент на стороні сервера, тому *Servlet* успадковує захист, який забезпечує вебсервер;

API, розроблений для *Java Servlet*, автоматично набуває переваг платформи *Java*, таких як незалежність від платформи та портативність. Крім того, він може використовувати широкий спектр *API*, створених на платформі *Java*, таких як *JDBC*, для доступу до бази даних.

2.5. Висновки до розділу

Java є загальновживаною мовою для веброзробки, особливо на стороні сервера. Існують різні способи створення динамічних вебсторінок у *Java*. Платформа *Java EE* (*Enterprise Edition*) надає розробникам різні технології *Java* для веброзробки. *Servlet API*, пакет *javax.servlet* постачається з багатьма інтерфейсами, такими як сервлет, фільтр, фільтр ланцюгів, *servletconfig* тощо. Сервлет збільшує можливості серверів, які використовуються для розміщення додатків. Вебпрограми, розроблені через сервлет на *Java*, відповідають моделі запит-відповідь. Сервлет має життєвий цикл, починаючи від ініціалізації та збирання сміття.

Підключення до бази даних *Java* (*JDBC*) містить методи та запити для доступу до бази даних. Клієнти можуть оновлювати будь-яку інформацію в базі

даних через вебпрограми, що містять драйвери *JDBC*. Чотири типи драйверів *JDBC* - це драйвер моста *JDBC-ODBC*, власний драйвер, драйвер мережевого протоколу та тонкий драйвер, які використовуються для підключення до бази даних. Клієнти можуть підключатися до бази даних через програми, створені за допомогою *JDBC API*, а також можуть оновлювати, видаляти, зберігати та отримувати доступ до даних. *JDBC* здатний читати будь-яку базу даних і автоматично створює *XML*-формат даних із бази даних.

Hibernate використовує об'єктно-реляційне відображення для підключення об'єктноорієнтованої моделі до бази даних. Реляційними даними в *Java*-програмах можна легко керувати за допомогою *Java Persistence*. Це допомагає постійно зберігати або отримувати велику кількість даних у/з бази даних. Для взаємодії з базою даних не потрібно використовувати багато коду, власні фреймворки тощо. *Hibernate* надасть прості засоби зв'язку з базою даних за допомогою об’єктно-реляційного підходу. *Hibernate* – це сукупність ефективних класів та методів, які можуть увімкнути додаток до бази даних.

База даних – це систематичний збір даних. Вони підтримують електронне зберігання та маніпулювання даними. Бази даних полегшують управління даними. Існує багато типів баз даних, найпопулярніші типи: централізована база даних, розподілена база даних, база даних підприємств (*Enterprise Database*), оперативна база даних, ієрархічні бази даних, об'єктноорієнтовані бази даних, хмарна база даних, *non-SQL*/*Not Only SQL*, реляційна база даних.

РОЗДІЛ 3
РОЗРОБКА ОНЛАЙНОВОЇ ПЛАТФОРМИ ПОШУКУ МЕНТОРА У НАВЧАННІ

Онлайн платформа, яку часто називають вебпрограмою, це – інтерактивна комп'ютерна програма, побудована з використанням вебтехнологій (*HTML, CSS, JS*), яка зберігає, тобто необхідно визначити, які дані потрібно зберігати у базі даних, а також типи даних.

Вебпрограми – це розподілені програми за своєю природою. Це означає, що будь-яка програма, яка працює на декількох комп'ютерах і взаємодіє за допомогою мережі та сервера. Доступ до вебдодатків здійснюється за допомогою веббраузера, тому вони дуже популярні завдяки простоті використання браузера як клієнтського користувача. Можливість оновлювати та підтримувати вебпрограми без встановлення будь-якого програмного забезпечення на тисячах клієнтських комп'ютерів стає ключовою причиною попиту. За допомогою багатьох компонентів створюються вебпрограми, деякі з яких мають користувальницький інтерфейс, а деякі з них не потребують графічного інтерфейсу користувача (*GUI*). Крім того, вебдодатки часто вимагають додаткової мови розмітки або сценарію, наприклад, мови програмування *HTML, CSS* або *JavaScript*. Багато додатків використовують лише мову програмування *Java*, що ідеально підходить завдяки своїй універсальності. Вебпрограма *Java* - це сукупність динамічних ресурсів (таких як сервлети, сторінки *JavaServer*, класи *Java*) та статичних ресурсів (*HTML*-сторінок та зображень). Вебпрограму *Java* можна розгорнути як файл *WAR* (*Web ARchive*).*u* Файл *WAR* – це *zip*-файл, який містить повний вміст відповідної вебпрограми.

Побудова бази даних. Вебдодаток обробляє дані (*CRUD*), і

використовується для виконання завдань через Інтернет. *CRUD* – це популярна абревіатура, яка лежить в основі розробки вебдодатків. Він розшифровується як створення, читання, оновлення та видалення. *API JDBC* дозволяє викликати команди *SQL* бази даних із методів мови програмування *Java*. Можна використовувати *API JDBC* у сервлеті, на сторінці технологій *JSP* або в корпоративному файлі, коли потрібен доступ до бази даних.

API JDBC складається з двох частин: інтерфейсу на рівні програми, який компоненти програми використовують для доступу до бази даних, та інтерфейсу постачальника послуг. Доступ до вебпрограм здійснюється через веббраузер, наприклад як *Google Chrome*. Також потрібно створити *backend* – одна з найскладніших частин процесу розробки вебдодатків. Основними функціями *backend* є надання кінцевих точок *HTTP* для інтерфейсу, автентифікація користувачів та обслуговування інтерфейсу.

3.1. Розробка бази даних

Перш ніж почати розробку схеми бази даних необхідно визначитись з якою базою даних буде працювати додаток. Проаналізувавши види бази даних та їх основні властивості для розробки онлайн платформи для пошуку ментора у навчанні було обрано *MySql* базу даних за наступними властивостями: *MySQL* – це проєкт з відкритим кодом із подвійним ліцензуванням. *MySQL* доступний безплатно для більшості власних потреб. Простота використання. *MySQL* – це високопродуктивна, але відносно проста система баз даних і набагато менш складна в налаштуванні та адмініструванні, ніж інші системи. Підтримка мови запитів. *MySQL* розуміє *SQL* (структуровану мову запитів), стандартну мову вибору для всіх сучасних систем баз даних. інтерфейси програмування доступні для багатьох мов, таких як *C*, *Perl*, *PHP*, *Python* та *Ruby*, включаючи *Java*. *MySQL* має помірний розмір розподілу, особливо в порівнянні з величезним дисковим простором певних систем комерційних баз даних. Відкритий дистрибутив та вихідний код. Обравши базу даних, необхідно сформувати таблиці для збереження даних. Було виокремлено наступні таблиці:

– *Users*;
– *Profiles*;
– *Reviews*;
– *Aspects*;
– *Profiles_aspects*;
– *Images*;

При розробці таблиць було дотримано правила нормалізації та всі таблиці нормалізовані до третьої нормальної форми.

Таблиця *users* зберігатиме дані користувачів. Діаграма наведена на рис. 3.1 [2]. Містить наступні поля: *id*, *full_name*, *bitrhday*, *email*, *phone*, *type*, *avatar_id*, *login*, *password*, *registration_date*, *last_visit_date*.

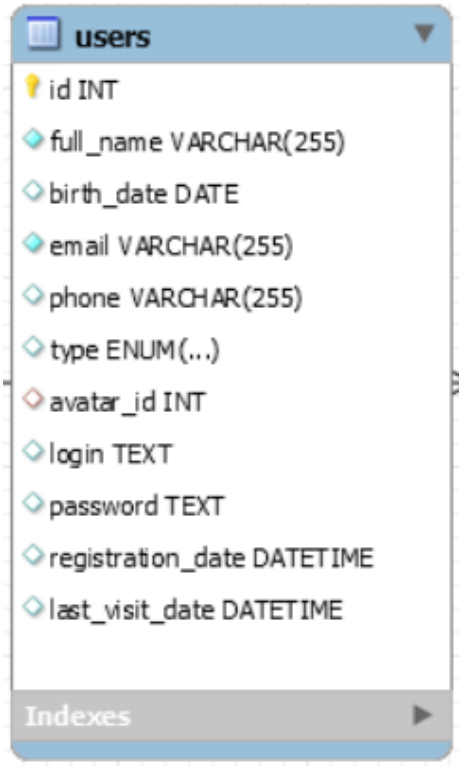

Рис. 3.1. Діаграма таблиці *users*

Поле *id* зберігає унікальне значення за яким можна знайти користувача та має *int*, який дозволяє зберігати цілі числа в рамках від -2147483648 до 2147483647, для зберігання унікального номера користувача системи цього буде достатньо. Поле *full_name* зберігає ім’я та прізвище користувача та має тип *varchar*(255), що дозволить зберігати строку зі зміною довжиною, тобто якщо зазначено *varchar*(255), а в це поле буде записана строка в 20 символів то поле буде займати 20 символів, а не 255, що дозволяє економити пам’ять, а 255

виділених символів повинно вистачити для зберігання ім’я та прізвища. Поле *birthday* має тип *date* та займає 3 *Б*, що дозволяє зберігання дати, а саме дату народження користувача. Полю *email* також присвоєно тип *varchar* та виділено 255 символів чого також буде достатньо для зберігання *email*. Поле *phone* має тип *varchar*. Для цього поля виділено 255 символів. Для зберігання номеру було обрано тип строки задля зручності зберігання номера у певному форматі, тобто номер може мати формат: +380998887744, +380 99 888 77 44, +380-99-888-77-44 і т.д., й у разі необхідності зміни формату збереження номера у базі даних не виникне проблем на відміну якщо, наприклад, номер зберігався б у числовому форматі. Поле *type* має тип *ENUM*('*MENTOR*', '*CLIENT*'). Цей перелік містить у собі типи користувачів: *MENTOR, CLIENT*. *ENUM* – це рядковий об'єкт зі значенням, вибраним зі списку дозволених значень, які явно перераховуються у специфікації стовпця під час створення таблиці. Поле *avatar_id* має цілочисельний тип та є зовнішнім ключем на таблицю *images*, яка буде описана згодом. Поле *login* має тип *text*, що дозволяє зберігати текст довжиною до 65 *Кб*, що дозволяє зберігати будь-який логін користувача в рамках розміру поля. Поле *password* має тип *text* для зберігання хешу пароля. Адже у випадку, коли паролі зберігаються у вигляді простого тексту зловмисник безпосередньо володіє паролями всіх користувачів. Зберігання паролів у простому тексті – не безпечне рішення. Ніхто, включаючи адміністраторів вебсайтів/баз даних, не повинен мати доступу до простого текстового пароля користувача. Поле *registration_date* має тип *datetime*, для збереження дати та часу реєстрації користувача, поле займає 8 *Б*. Поле *last_visit_date* має тип *datetime*, що зберігатиме дату та час останнього входу користувача для відображення активності та можливого подальшого впровадження системи сповіщення та нагадування користувачеві про вхід або бонуси.

Таблиця *profiles* зберігає дані профілів менторів та містить наступні поля: *id, description, rate, user_id, profesion, rating, views.*

Поле *id* має тип *int* та є первинним ключем таблиці *profiles*. Поле *description* має тип даних *text*, для збереження опису професії ментора. Поле *rate* має тип даних *decimal*(5,2) для збереження оплати ментора. Цей тип даних дозволяє

зберігати числа з фіксованою точністю. У цьому прикладі 5 – це точність, а 2 – шкала. Точність представляє кількість значущих цифр, які зберігаються для значень, а шкала - кількість цифр, які можна зберегти після десяткової коми. Поле *user_id* має тип *int* та є зовнішнім ключем на таблицю *users*, для вказання якому саме користувачеві належить профіль. Поле *profesion* має тип даних *text*, для збереження професії, яку має ментор. Поле *rating* має такий же тип як і *rate*, тобто *decimal*(5,2). У цьому полі зберігаються дані про рейтинг ментора. Поле *views* має тип *int* й зберігає дані про кількість переглядів профілю. Діаграма наведена на рис. 3.2.

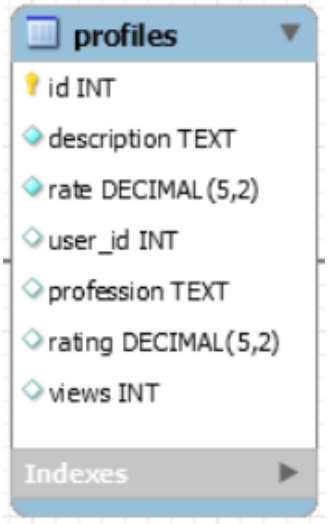

Рис. 3.2. Діаграма таблиці *profiles*

Таблиця *reviews* зберігає дані коментарів та має наступні поля: *id, author_id, profile_id, rating, comment, comment_date*.

Поле *id* має тип *int* та є первинним ключем таблиці *reviews*. Поле *author_id* має тип *int* та є зовнішнім ключем на таблицю *users* для збереження *id* користувача, який залишив коментар. Поле *profile_id* має тип *int* та є зовнішнім ключем на таблицю *profiles* для збереження *id* профілю користувача, що написав коментаря. Поле *rating* має тип *int* для збереження цілого числа, а саме оцінки від коментатора, яка зберігається в коментарі. Поле *comment* містить текст коментаря та має тип *text*. Поле *comment_date* зберігає дату та час надсилання коментарю та має тип даних *datetime*. Діаграма наведена на рис. 3.3.

Рис. 3.3. Діаграма таблиці *reviews*

Таблиця *aspects* зберігає назви професій менторів та має наступні поля: *id*, *name*.

Поле *id* має тип *int* та є первинним ключем таблиці *aspects*. Поле *name* зберігає назву професії ментора та має тип *varchar*(255) заданого розміру буде досить для зберігання назви професії.

Таблиця *profiles_aspects* зберігає зовнішні ключі на первинні ключі таблиць *aspects* й *profiles* для реалізації відношення багато до багатьох, що буде розглянуто далі. Таблиця містить поля: *profile_id*, *aspect_id*.

Поле *profile_id* має тип *int* та є зовнішнім ключем на таблицю *profiles*. Поле *aspect_id* має тип *int* та є зовнішнім ключем на таблицю *aspects*.

Таблиця *images* містить дані про зображення. Існує два способи збереження зображень. Найбільш поширеним способом є збереження імені файлу в таблиці *MySQL* та завантаження зображення в теку. Інший спосіб – збереження зображення безпосередньо в базі даних. Оскільки розробники зазвичай не використовують другий метод, вони можуть заплутатися. Тож було обрано перший спосіб збереження зображень у базу даних. Таблиця має наступні поля: *id*, *path*.

Поле *id* має тип *int* та є первинним ключем таблиці *images*. Поле *path* зберігає шлях до директорії де знаходиться картинка.

У базі даних є три різні типи відносин між таблицями:

- Один-до-одного;
- Один-до-багатьох;
- Багато-до-багатьох;

У взаємозв'язку один-до-багатьох один запис у таблиці може бути пов'язаний з одним або кількома записами в іншій таблиці. Таблиця *profiles* відноситься з таблицею *reviews* саме один до багатьох. Так як у таблиці *reviews* багато коментарів пишуться на один профіль. А зв’язок встановлюється за допомогою зовнішнього ключа *profile_id*. Діаграма наведена на рис. 3.4.

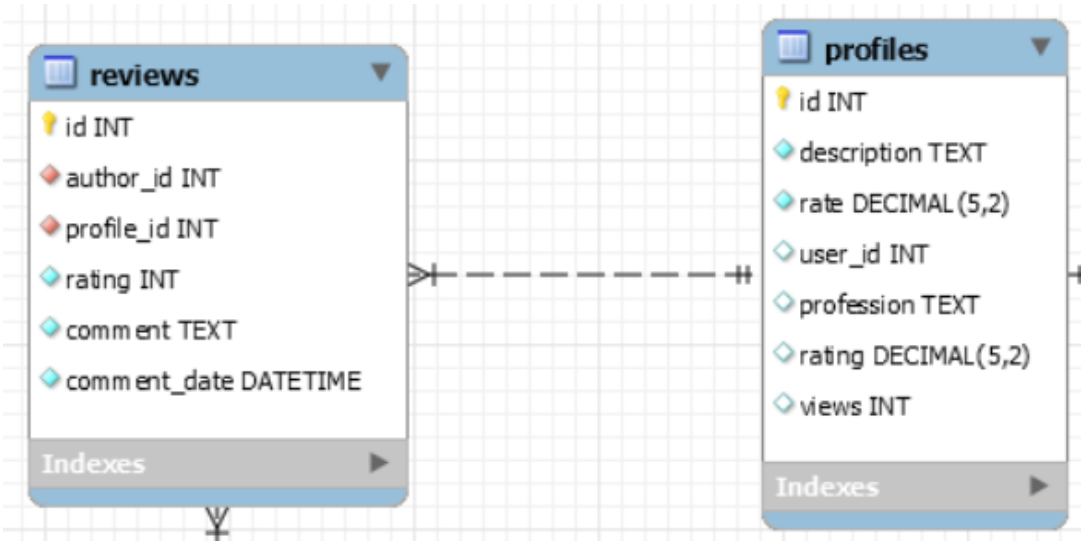

Рис. 3.4. Діаграма відношення один-до-багатьох таблиць *reviews* та *profiles*

Також зв’язком один-до-багатьох зв’язані таблиці *reviews* та *users*. Поле *author_id* вказує на первинний ключ таблиці *users*. Діаграма наведена на рис. 3.5.

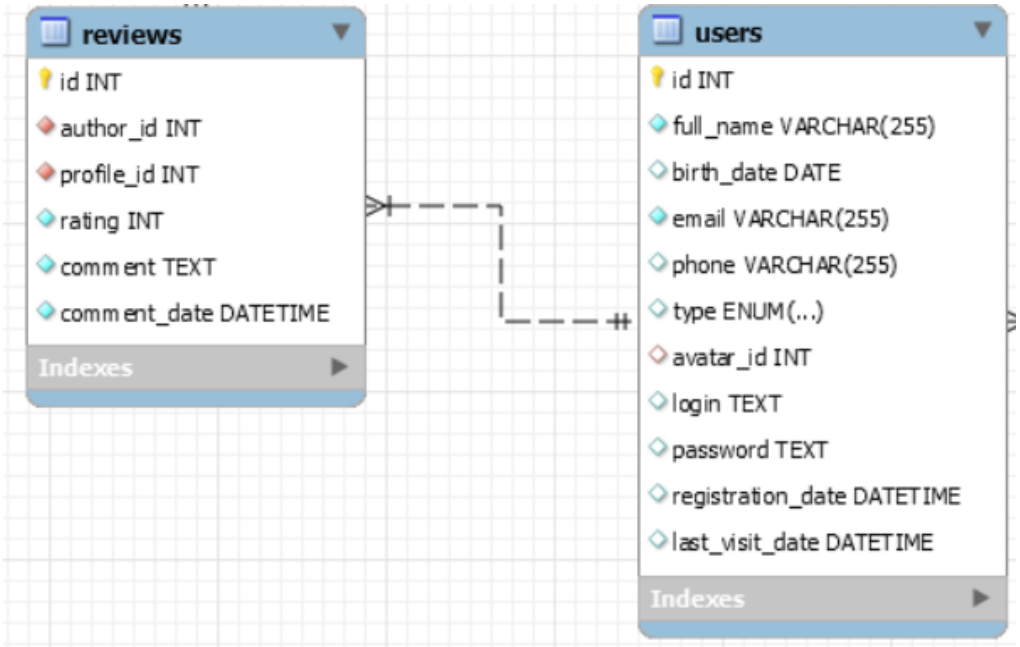

Рис. 3.5. Діаграма відношення один-до-багатьох таблиць *reviews* та *users*

У взаємозв’язку один-до-одного один запис у таблиці пов’язаний з одним і лише одним записом в іншій таблиці. У базі даних додатку таке відношення

мають таблиці *users* *images* адже в одного користувача може бути лише один аватар. Відношення між таблицями встановлюється за допомогою зовнішнього ключа *avatar_id* в таблиці *users*. Діаграма наведена на рис. 3.6.

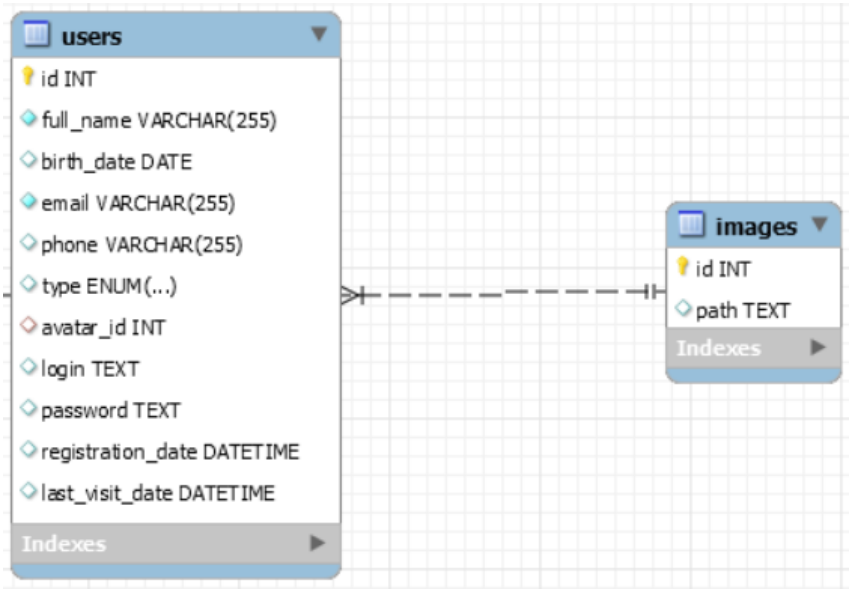

Рис. 3.6. Діаграма відношення один-до-одного таблиць *images* та *users*

Відношення багато-до-багатьох виникає, коли кілька записів у таблиці пов'язано з кількома записами в іншій таблиці У базі даних додатку відношення багато-до-багатьох мають таблиці *profiles* та *aspects*. Створено таблицю, що об’єднує *profiles_aspects*, яка має відношення один до багатьох з кожною із таблиць та дозволяє зберігати багато профілів, які мають багато професій. Діаграма наведена на рис. 3.7.

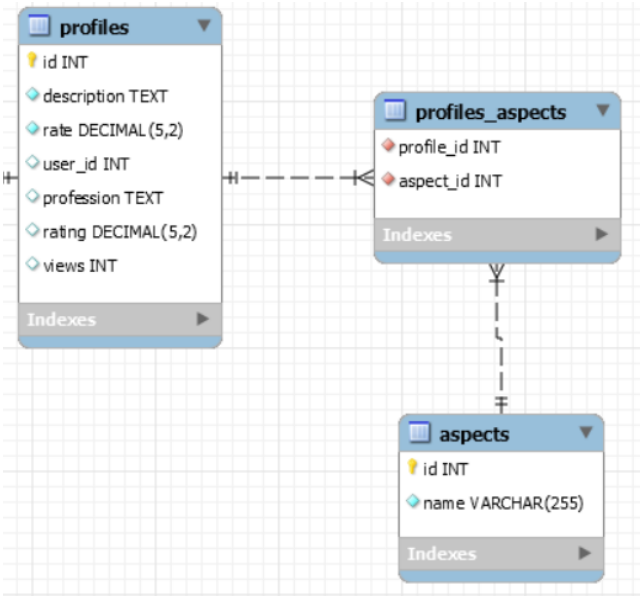

Рис. 3.7. Діаграма відношення багато-до-багатьох таблиць *profiles* та *aspects*

Таким чином було розроблено базу даних для додатку для пошуку ментора у навчанні. Діаграма всіх таблиць бази даних наведена на рис. 3.8, [3].

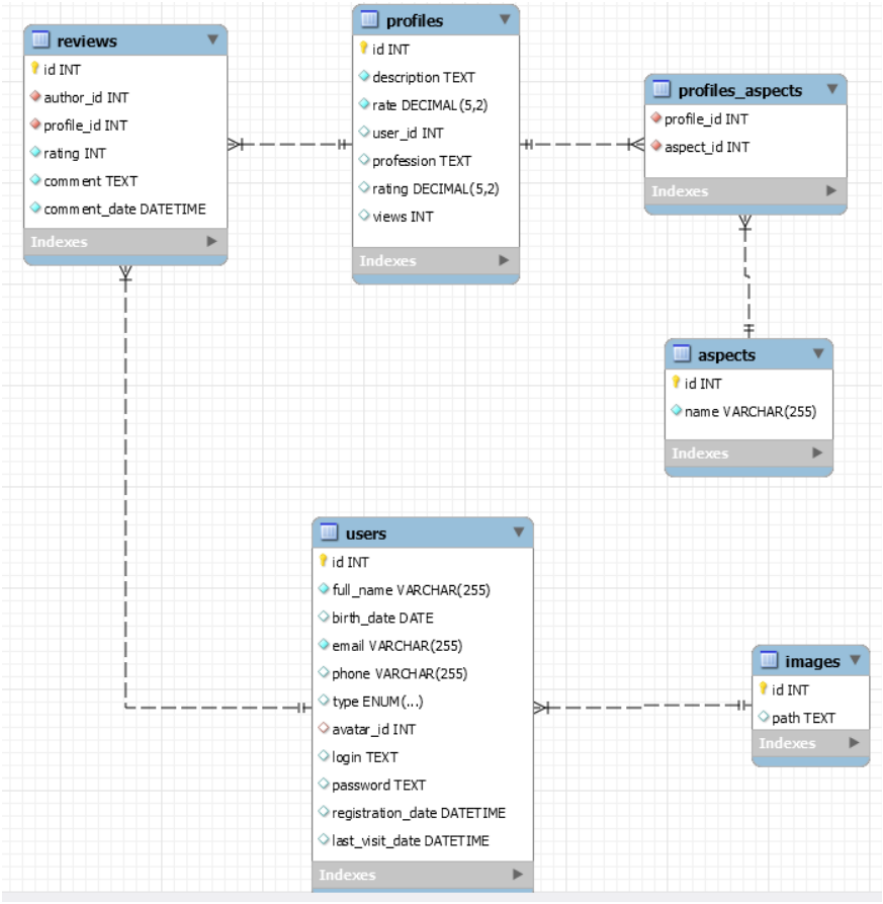

Рис. 3.8. Діаграма відношення багато-до-багатьох таблиць *profiles* та *aspects*

3.2. Застосування *Liquibase*

Liquibase – це інструмент з відкритим кодом, написаний на *Java*. Це полегшує визначення змін у базі даних у форматі, звичному та зручному для програміста. Потім він автоматично генерує *SQL* для бази даних. Зміни бази даних, кожна зміна називається набором змін (*changeset*), управляються в файлах, які називаються журналами змін (*changelogs*). *Liquibase* потребує двох таблиць у схемі баз даних, які створюються автоматично:

– *DATABASECHANGELOG* – таблиця, що зберігає інформацію про всі зміни, внесені в базу даних;

– *DATABASECHANGELOGLOCK* – використовується для того, щоб уникнути одночасного внесення змін користувачами, до бази даних;

Журнали змін (*changelogs*) розроблені з використанням мов, що відповідають домену. *Changelogs* складаються з серії наборів змін (*changesets*). *Changeset* містить реалізацію однієї зміни в базі даних. Деякі приклади включають створення таблиці, додавання стовпця до таблиці та додавання індексу. Також можна включити оператори *SQL* до журналу змін. Журнал змін слід тримати під контролем джерела. Коли вносяться зміни, додається новий набір змін до кінця журналу змін. Неможливо змінити набір змін, що існує, потрібно створювати новий набір змін.

У додатку *Liquibase* налаштований таким чином, що запускається головний *changelog* файл у форматі *xml* – *changelog-master.xml*. Файл містить у собі всі списки змін у базу даних, а саме включення інших *xml* файлів у яких описані певні зміни, які робились під час розробки додатку. Використовуючи тег <*include*>, у головному журналі змін зроблені посилання на інші журнали змін, а саме:

- 20210429-1851-*initial-setup.xml*;
- 20210429-1827-*initial-data.xml*;
- 20210504-1354-*profile-add-columns-update.xml*;
- 20210506-1342-*provide-image-save.xml*;
- 20210508-1557-*provide-user-authentication.xml*;
- 20210507-1308-*provide-review.xml*;

Файл змін 20210429-1827-*initial-data-setup.xml* містить набори зі створенням таблиць та описанням створення кожного з полів та ключів у базі даних. Тег *Changeset* – це одиниця змін, яку *Liquibase* виконує в базі даних і яка використовується для групування типів змін бази даних разом. Список змін, створених кількома наборами змін, відстежується в журналі змін. Набір змін однозначно позначається як атрибутами автора, так і ідентифікатора (*author*:*id*), а також шляхом до файлу журналу змін. Тег *id* використовується лише як ідентифікатор, він не спрямовує порядок запуску змін і не повинен бути цілим числом. Якщо фактичного автора зберігати не потрібно або його ім’я невизначено, використовується значення заповнювача, таке як *UNKNOWN*. Щоб виконати набір змін, потрібно вказати як автора, так і ідентифікатор. У *changelog* під назвою 20210429-1851-*initial- data.xml* описані додавання тестових даних у

таблиці бази даних, за допомогою створення листів змін, по одному на кожну таблицю. У кожному наборі змін цього файлу використовується тег <*insert*>. Файл 20210504-1354-*profile-add-columns-update.xml* у першому наборі змін додає до таблиці *profiles* поле *views* за допомогою тега <*addColumn*>, а в другому наборі змін запроваджує оновлення даних таблиці <*update*> [4]. Наступний файл змін 20210506-1342-*provide-image-save.xml*. Він у першому наборі змін створює таблицю *images* та описує її поля та їх обмеження за допомогою тега <*createTable tableName*=" *images*">. У другому наборі змін вставляються дані в таблицю *images* в поле *path* за допомогою тегу <*insert tableName*="*image*"> в якому зазначається ім’я таблиці, а в середині цього тегу використовується тег <*column name*=" " *value*=" "> в параметри яких вказується ім’я колонки в яку буде додано дані та значення даних. У третьому наборі змін до таблиці *user* додається поле *avatar_id* для. У третьому наборі у поле *avatar_id* вставляються значення. Файл 20210508-1557-*provide-user-authentication.xml* містить зміни для впровадження автентифікації користувача. Перший набір змін додає колонки *login*, *last_visit_date*, *registration_date*, *password* у таблицю *users*. Другий набір вставляє кожному користувачеві в базі у додані поля дані. Файл змін 20210507-1308-*provide-review.xml* містить набори змін для збереження відгуків на профілі. У першому наборі створюється таблиця *reviews*. У другому заповнюються поля створеної таблиці.

3.3. Розробка об’єктів предметної області

Представлені дані у БД потрібно організувати у програмному коді *Java* у вигляді класів. Це необхідно для зручної роботи з даними та розробки логіки роботи програми. В онлайн платформі для пошуку менторів для навчання виокремлено наступні класи для збереження даних з БД [5]:

- *Contact*;
- *Image*;
- *LifeAspect*;
- *Profile*;

– *Review*;

– *User*;

Усі зазначені класи знаходяться в каталозі *entity*. Клас *Contact* має поля *email*. *phone* текстового типу *String* та призначені для зберігання пошти та телефону відповідно. Поля помічені анотацією *@Column*. Анотація вказує відображений стовпець для властивості або поля. Якщо жодна анотація стовпця не вказана, тоді для відображення будуть використані назви полів. Над полем *email* стоїть анотація *@Column*(*name* = "*EMAIL*"), що вказує, що для даного поля в таблиці буде поле з ім’ям переданим в параметр *name*. Для поля *phone* в параметрі *name* передано *PHONE*. Поля мають приватний модифікатор доступу, а для роботи з ними створені методи, так звані гетери та сетери. Клас заміняє метод *toString* для виводу даних класу. Клас помічено анотацією *@Embeddable*. Тип анотації *Embeddable*. Визначає клас, екземпляри якого зберігаються як невід'ємна частина сутності-власника та мають ідентичність сутності. Кожне з постійних властивостей або полів вбудованого об'єкта відображається у таблиці бази даних для сутності. Тобто клас є частиною іншого класу та для даного класу не буде своєї таблиці, проте його поля будуть записані до таблиці класу в якому знаходиться *Embeddable* клас.

Клас *Image* зберігає дані про картинку та має приватні поля:

– *Id*;

– *Path*;

Поле *id* має тип *Long* та зберігає ідентифікатор картинки. Поле *path* має тип *String* та зберігає значення шляху до картинки. Клас помічено анотаціями *@Entity* й *@Table*(*name* = "*IMAGES*"). Анотація *@Entity* вказує, що клас є суттю і відображається у таблиці бази даних. Анотація *@Table* визначає ім'я таблиці бази даних, яка буде використовуватися для зіставлення. Анотація *@Id* визначає первинний ключ сутності, а *@GeneratedValue* передбачає специфікацію стратегій генерації значень первинних ключів. Поле *id* помічено анотацією *@Id* та *@GeneratedValue*(*strategy = GenerationType.IDENTITY*), що вказує на те, що постачальник стійкості повинен призначити первинні ключі для сутності за допомогою стовпця ідентифікації бази даних. У класі перевизначено методи

hashCode та *equals*. Методи *hashCode*() та *equals*() були визначені в класі *Object*, який є батьківським класом для об'єктів *Java*. З цієї причини всі об'єкти *Java* успадковують реалізацію цих методів за замовчуванням. Як правило, необхідно замінити метод *hashCode* (), коли метод *equals* () замінений, щоб підтримувати загальний контракт на метод *hashCode* (), який стверджує, що рівні об'єкти повинні мати рівні хеш-коди. Кожного разу, коли він викликається для одного і того ж об'єкта більше одного разу під час виконання програми *Java*, метод *hashCode* повинен послідовно повертати одне і те ж ціле число, за умови, що жодна інформація, яка використовується для порівняння рівних об'єктів, не змінюється. Це ціле число не повинно залишатися послідовним від одного виконання програми до іншого виконання тієї самої програми. Якщо два об'єкти рівні за методом *equals* (*Object*), тоді виклик методу *hashCode* для кожного з двох об'єктів повинен давати однаковий цілий результат. Не потрібно, щоб, якщо два об'єкти нерівні за методом *equals* (*java.lang.Object*), тоді виклик методу *hashCode* для кожного з двох об'єктів повинен давати різні цілі результати. Однак програміст повинен знати, що отримання чітких цілочисельних результатів для нерівних об'єктів може покращити продуктивність хеш-таблиць.

Клас *LifeAspect* зберігає усі подробиці про навички ментора та містить поля:

- *Id*;
- *Name*;
- *Profiles*;

Поле *id* має тип *Long* та зберігає ідентифікатор аспекту, також помічено анотацією *@Id* та *@GeneratedValue*(*strategy = GenerationType.IDENTITY*). Поле *name* має тип *String* та зберігає назву аспекту. Також клас реалізує методи: гетери, сетери, *hashCode*, *equals*, *toString*. Клас помічено анотаціями *@Entity* й *@Table*(*name = "ASPECTS*"). Поле *profiles* зберігає набір профілів та помічено анотацією *@ManyToMany*(*mappedBy = "lifeAspects*", *fetch = FetchType.LAZY*). Анотація *@ManyToMany* використовується для моделювання відносин багато-до-багатьох. Цей тип відносин може бути односпрямованим або двонапрямний: в односпрямованих відносинах лише одна сутність у відносинах вказує на іншу. У двонаправлених відносинах обидва суб’єкти вказують один на одного.

FetchType.LAZY – завантажує дані, коли вони будуть потрібні. Тип *FetchType.LAZY* говорить *Hibernate* отримувати пов'язані сутності з бази даних лише тоді, коли дані використовуються.

Клас *Profile* зберігає інформацію про профіль ментора. Містить наступні приватні поля:

- *Id*;
- *Profession*;
- *Rating*;
- *Description*;
- *Rate*;
- *ViewsCount*;
- *LifeAspects*;
- *User*;
- *Reviews*;

Поле *id* має тип *Long* та зберігає ідентифікатор профілю ментора, також помічено анотацією *@Id* та *@GeneratedValue*(*strategy = GenerationType.IDENTITY*). Поле *profession* має тип *String* для текстового збереження професії, помічено анотацією *@Column*(*name* = "*PROFESSION*"). Поле *rating* має тип *int* для цілочисельного збереження рейтингу, помічено анотацією *@Column*(*name* = "*rating*"). Поле *description* має тип *String* для текстового збереження опису профілю, помічено анотацією *@Column*(*name* = "*description*"). Поле *rate* має тип *Double* для збереження оплати у виді числа з комою, що плаває, помічено анотацією *@Column*(*name* = "*rate*"). Поле *viewsCount* має тип *int* для цілочисельного збереження оплати, помічено анотацією *@Column*(*name* = "*rate*"). Поле *lifeAspects* містить список аспектів профілю, помічено анотаціями *@ManyToMany*(*fetch = FetchType.LAZY*, *cascade* = {*CascadeType.PERSIST*, *CascadeType.MERGE*}), *@JoinTable*(*name* = "*PROFILES_ASPECTS*", *joinColumns* = *@JoinColumn*(*name* = "*ASPECT_ID*"), *inverseJoinColumns* = *@JoinColumn*(*name* = "*PROFILE_ID*")). Ці анотації вказують правила відносин між об’єктами та за якими полями в БД об’єднувати таблиці. Поле *user* має тип *User* та містить у собі поля з інформацією про користувача якому належить профіль. Поле помічено

анотаціями *@OneToOne*(*fetch = FetchType.LAZY, cascade = CascadeType.ALL*) *@JoinColumn*(*name* = "*USER_ID*"). Поле *reviews* має тип *Review* та зберігає список коментарів до профілю. Поле помічено анотацією *@OneToMany*(*mappedBy* = "*profileId*", *fetch = FetchType.LAZY, cascade = CascadeType.ALL*). Клас *Profile* має методи: гетери, *hashCode*, *equals*, *toString*.

Клас *Review* має методи: гетери, *hashCode*, *equals*, *toString*. Та приватні поля:

- *Id*;
- *ProfileId*;
- *Author*;
- *Rating*;
- *Comment*;
- *CommentDate*;

Поле *id* має тип *Long* та зберігає ідентифікатор коментаря, також помічено анотацією *@Id* та *@GeneratedValue*(*strategy = GenerationType*. *IDENTITY*). Поле *profileId* зберігає ідентифікатор профілю до якого залишено коментар, має анотацію *@Column*(*name* = "*PROFILE_ID*"). Поле *author* має тип *User* зберігає інформацію про автора коментаря, має анотацію *@ManyToOne*(*fetch = FetchType.LAZY*). Поле *rating* зберігає вподобайки до коментаря, має тип *int* для цілочисельного збереження рейтингу, помічено анотацією *@Column*(*name* = "*rating*"). Поле *comment* має тип *String* для збереження тексту коментаря, помічено анотацією *@Column*(*name* = "*comment*"). Поле *commentDate* має тип *LocalDateTime* для збереження дати та часу коментаря.

Клас *User* зберігає інформацію про користувача. Має методи: гетери, *hashCode*, *equals*, *toString*. Помічений анотаціями *@Entity*, *@Table*(*name* = "*USERS*"). Клас має приватні поля:

- *Id*;
- *FullName*;
- *Birthdate*;
- *Avatar*;
- *Password*;

– *Login*;
– *Usertype*;
– *RegistrationNate*;
– *LastVisitDate*;

Поле *id* має тип *Long* та зберігає ідентифікатор користувача, також помічено анотацією *@Id* та *@GeneratedValue*(*strategy = GenerationType. IDENTITY*). Поле *fullName* має тип *String* для текстового збереження ПІБ, помічено анотацією *@Column*(*name* = "*fullName*"). Поле *birthDate* має тип *LocalDate* для збереження дати народження користувача, помічено анотацією *@Column*(*name* = "*BIRTH_DATE*"). Поле *avatar* має тип *Image* що зберігає інформацію про аватар користувача, помічено анотаціями *@Column*(*name* = "*AVATAR_ID*"), *@OneToOne*(*fetch = FetchType.LAZY, cascade = CascadeType.ALL*). Поле *password* має тип *String* для текстового збереження пароля у зашифрованому вигляді, помічено анотацією *@Column*(*name* = " *password* "). Поле *login* має тип *String* для текстового збереження логіну, помічено анотацією *@Column*(*name* = "*login*"). Поле *registrationDate* має тип *LocalDateTime* для збереження дати та часу реєстрації користувача, помічено анотацією *@Column*(*name* = "*registrationDate*"). Поле *lastVisitDate* має тип *LocalDateTime* для збереження дати та часу останнього входу користувача, помічено анотацією *@Column*(*name* = "*lastVisitDate*"). Поле *userType* має тип *UserType* для визначення типу користувача: *CLIENT*/*MENTOR*.

3.4. Розробка сервісного рівня

Сервісний рівень містить у собі реалізації логіки роботи програми. Сервісний рівень онлайн платформи для пошуку ментора у навчанні можна умовно поділити на реалізацію роботи з БД та реалізацію фільтрування менторів. Частина роботи з БД складається з наступних класів:

– *AspectServiceImpl*;
– *UserServiceImpl*;
– *ReviewServiceImpl*;
– *ProfileServiceImpl*;

– *ImageServiceImpl*;

– *EncoderImpl*;

Сервіс *AspectServiceImpl* створений для роботи з аспектами менторів. Має метод *findAll* для знаходження всіх аспектів. Також використовує *AspectRepository*, який є реалізацією *JpaRepository* з *Spring Data*. Це зменшує кількість коду, необхідного для роботи з базами даних і сховищами даних. За допомогою *Spring Data* визначається інтерфейс сховища для кожної сутності домену в додатку. Репозиторій містить методи виконання *CRUD*-операцій, сортування та перенесення даних. *@Repository* – це анотація маркера, яка вказує на те, що базовий інтерфейс є сховищем. Сервіс *UserServiceImpl* використовує *UserRepository* та *Encoder*. *UserRepository* є інтерфейсом *JPA*. Сервіс реалізує логіку роботи над користувачами у методах:

– *Login*;

– *FindByIdAndUserType*;

– *FindByLogin*;

– *DeleteById*;

– *Update*;

– *OnApplicationEvent*;

– *FindById*;

– *FindAllByUserType*;

– *Save*;

Метод *login* забезпечує процес автентифікації користувача, див. рис. 3.9. Метод отримує два параметри: *login, password*. За допомогою методу *findByLogin* репозиторія *userRepository* за логіном знаходиться користувач, якщо користувача не знайдено, то автентифікація не успішна. Якщо ж користувача було знайдено – йде перевірка на пароль, якщо введений пароль й пароль з БД однакові, то автентифікація успішна. Перевірка паролів відбувається завдяки методу *decodePassword* класу *userRepository*.

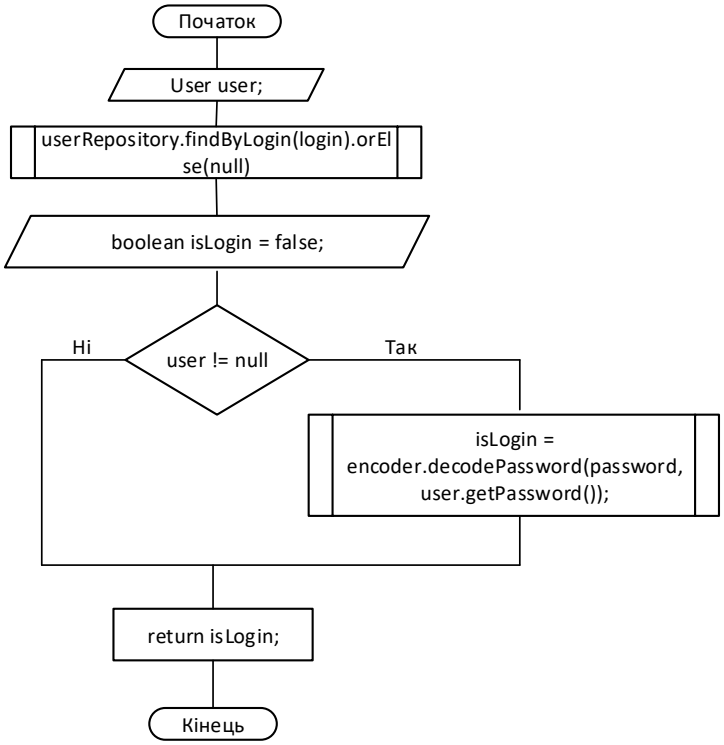

Рис. 3.9. Схема алгоритму метода *login*

Метод *findByIdAndUserType* повертає користувача за вказаним ідентифікатором та типом користувача. Якщо не було знайдено, то повертає помилку з повідомленням в якому зазначено, ідентифікатор користувача "*User with such id*: *%d not exist*". Метод *findByLogin* повертає, з БД, користувача за вказаним логіном користувача, якщо не знайдено – кидає помилку типу *EntityNotFoundException* з повідомленням: "*User with such login*: *%s not found*", де вказано логін за яким не було результатів пошуку. Метод *deleteById* видаляє з БД користувача за вказаним у параметрі ідентифікатором. Метод *update* оновлює інформацію в таблиці за вказаним ідентифікатором та повертає оновлений об'єкт. Якщо за переданим параметром не знайшлося інформації – кидається помилка *EntityNotFoundException* з повідомленням, що користувача не було знайдено за параметром. Метод *onApplicationEvent* під час кожного входу користувача оновлює дату входу в таблиці. Метод *findById* знаходить за *id* та повертає з таблиці об'єкт. Метод *findAllByUserType* знаходить за вказаним типом користувача та повертає список всіх знайдених об'єктів. Метод *save* зберігає користувача до БД. Методи *save*, *update*, *deleteById* помічені анотацією *@Transactional*. Якщо *Spring* виявляє анотацію *@Transactional* на компоненті, він створює динамічний проксі-сервер для цього компонента. Проксі має доступ до

менеджера транзакцій і попросить його відкрити та закрити транзакції. Транзакції в *Java*, як правило, стосуються ряду дій, які всі повинні бути успішно виконані. Отже, якщо одна або кілька дій не вдаються, всі інші дії повинні відступити, залишаючи стан програми незмінним. Це необхідно для того, щоб ніколи не порушувати цілісність стану програми. Крім того, ці транзакції можуть включати один або кілька ресурсів, таких як база даних, черга повідомлень, що породжує різні способи виконання дій під транзакцією. Сюди входить виконання локальних транзакцій ресурсів з окремими ресурсами. У глобальній транзакції можуть брати участь кілька ресурсів.

Сервіс *ReviewServiceImpl* обробляє збереження коментарів за допомогою метода *save*. Метод *save* помічено анотацією *@Transactional*. Метод використовує *ReviewRepository* для збереження коментарю в таблиці.

Сервіс *ProfileServiceImpl* реалізує логіку роботи з профілями завдяки наступним методам:

- *FindByUserIdAndUserType*;
- *Update*;
- *FindAll*;

Метод *update* оновлює профіль та є транзакційним, для оновлення даних використовує *EntityManager*. *JPA EntityManager* доступний у пакеті *javax.persistence*, який використовується для взаємодії з контекстом стійкості. Метод *findAll* знаходить та повертає список всіх профілів. Метод *findByUserIdAndUserType* знаходить та повертає профіль за вказаними у параметрах ідентифікатором та типом користувача.

Сервіс *ImageServiceImpl* має методи:

- *DeleteImageById* – є транзакційним, призначений для видалення зображення з таблиці та теки;
- *UpdateImage* – є транзакційним, призначений для оновлення інформації про зображення у таблиці та оновлення зображення у теці;
- *SaveImage* – є транзакційним, призначений для збереження зображення;
- *GetImageBytesById* – повертає зображення у вигляді байтів;

Сервіс *EncoderImpl* призначений для роботи з паролями. Має наступні

методи: *checkPassword, encodePassword*.

Метод *encodePassword* хешує пароль, використовуючи статичний метод *BCrypt.hashpw*, див. рис. 3.10. Метод *checkPassword* приймає строку пароль та хешоване значення. За допомогою статичного методу *BCrypt.checkpw* перевіряє чи вірний пароль.

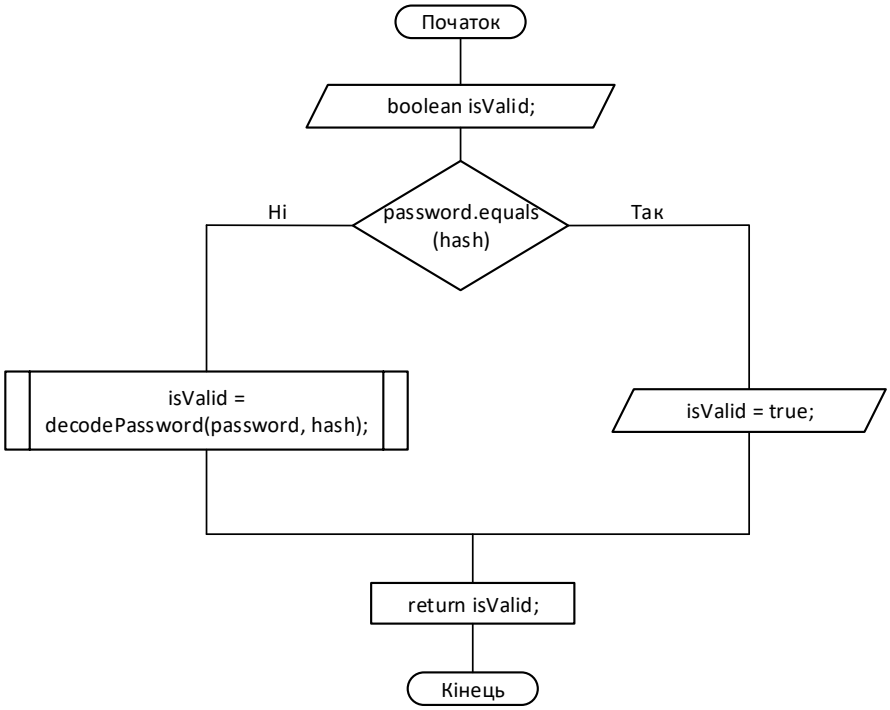

Рис. 3.10. Схема алгоритму метода *checkPassword*

3.5. Розробка рівня контролерів

Контролер відповідає за обробку вхідних запитів. Він викликає ділову логіку, оновлює модель і повертає подання, яке слід відтворити, див. рис. 3.11. Контролер *MVC* – це метод ресурсів *JAX-RS*, анотований *@Controller*. Якщо клас анотований *@Controller*, то всі методи ресурсів цього класу розглядаються як контролери [6].

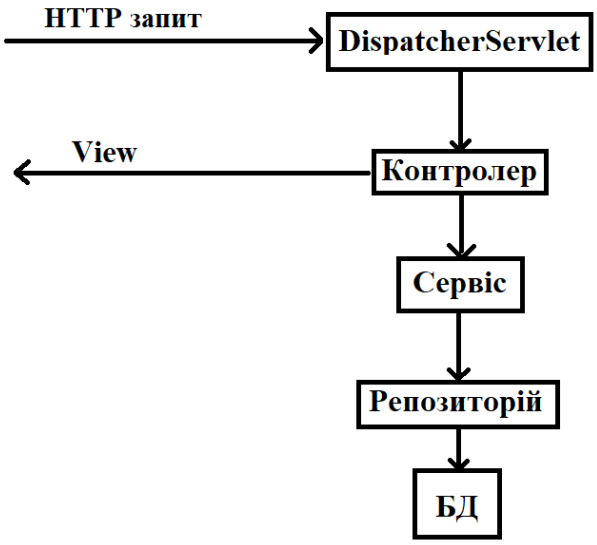

Рис. 3.11. Архітектура платформи для пошуку ментора онлайн

DispatcherServlet – це фронтальний контролер, оскільки він забезпечує єдину точку входу для запиту клієнта до вебпрограми *Spring MVC* та пересилає запит до контролерів *Spring MVC* для обробки.

Для реалізації *api* було створено контролери:

- *ViewsController*;
- *MentorsController* ;
- *AuthenticationController*;

Контролер *ViewsController* обробляє запити для повернення відображення головної сторінки та сторінки відображення для цього було реалізовано методи з *get* запитами *getHomeView getRegisterPage* відповідно.

Контролер *MentorsController* обробляє запити щодо менторів за допомогою методів: *saveMentor*, *getMentors*, *getMentorById*.

Метод *saveMentor* за *post* запитом отримує інформацію про ментора та зберігає її, використовуючи сервіс роботи з менторами [7]. Метод *getMentors* за *get* запитом повертає сторінку перегляду менторів. Метод *getMentorById* за *get* запитом повертає сторінку з усією інформацією про ментора.

Контролер *AuthenticationController* забезпечує автентифікацію ментора, див. рис. 3.12. Метод за *get* запитом *authentication* повертає відображення сторінки входу. Метод *login* за *post* запитом у разі успішного входу повертає відображення домашньої сторінки, у разі невдалого входу, коли користувач ввів

невірний пароль або логін, метод повертає сторінку входу.

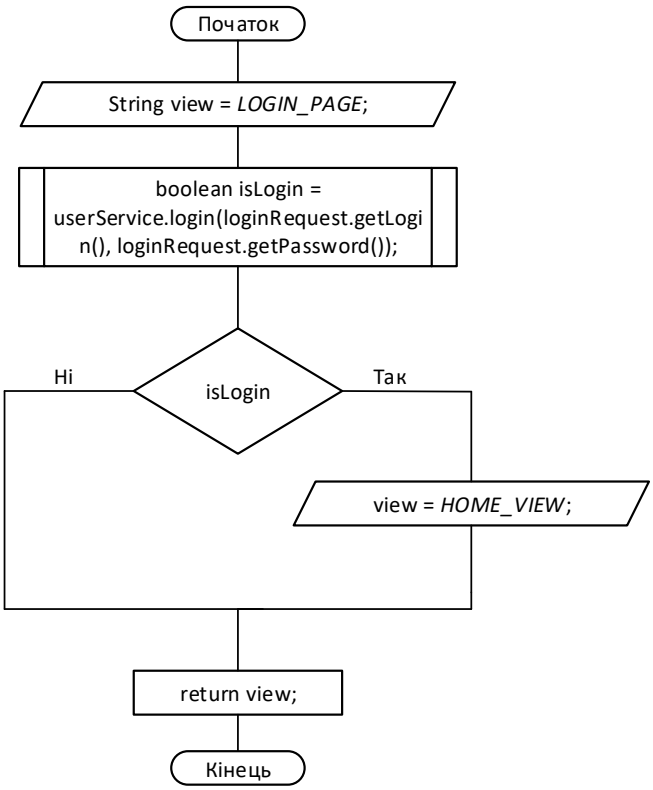

Рис. 3.12. Блок-схема алгоритму метода *login*

У кожному з контролерів використовується відповідний сервіс. Як було розглянуто вище сервіси можуть кидати помилки. Для обробки виключень було створено клас *RestResponseEntityExceptionHandler*. Клас помічено анотацією *@ControllerAdvice*. Це спеціалізація анотації *@Component*, яка дозволяє обробляти винятки в цілому додатку в одному глобальному компоненті обробки. Його можна розглядати як перехоплювач винятків, що створюються методами, анотованими *@RequestMapping*. Він оголошує, що методи *@ExceptionHandler*, *@InitBinder* або *@ModelAttribute* мають бути спільними для кількох класів *@Controller*. *ResponseEntityExceptionHandler* – це зручний базовий клас для класів *@ControllerAdvice*, які забезпечують централізовану обробку винятків для всіх методів *@RequestMapping* за допомогою методів *@ExceptionHandler*. Він надає методи обробки внутрішніх винятків *MVC Spring*. Він повертає *ResponseEntity* на відміну від *DefaultHandlerExceptionResolver*, який повертає *ModelAndView*. Так у класі *RestResponseEntityExceptionHandler* було реалізовано методи для обробки виключень: *handleRepositoryException* – виникають у репозиторіях. *handleEntityNotFoundException* – коли даних не було знайдено.

3.6. Розробка рівня відображення

У онлайн платформі пошуку ментора у навчанні за роботу користувацького інтерфейсу відповідає рівень відображення. При створенні рівня відображення, тобто вебсторінок застосовуються можливості *Thymeleaf*. Це інструмент шаблонів *Java* на стороні сервера для вебсередовищ на основі сервлетів. Він повністю забезпечує інтеграцію з *Spring Framework*. Також *Thymeleaf* застосовує набір перетворень до файлів шаблонів для відображення даних або тексту, створених додатком. *Thymeleaf* дозволяю забезпечити зручний та добре сформований спосіб створення шаблонів, за допомогою тегів та атрибутів *XML*. *Thymeleaf* може слугувати замінної *JSP*. *Thymeleaf* дозволяє швидко обробляти шаблони, що залежить від кешування проаналізованих файлів. Він використовує мінімально можливу кількість операцій вводу-виводу під час виконання.

Рівень відображення складається з вебсторінок написаних на *html*, а саме:

- *Navbar.html*;
- *HomeView.html*;
- *LoginView.html*;
- *MentorsView.html*;
- *ProfileView.html*;
- *RegisterView.html*;

Файл *navbar.html* знаходиться у каталозі *fragments* та реалізує навігаційне меню додатку, див. рис. 3.13, з функціональними кнопками: «Ментори онлайн», «Головна»

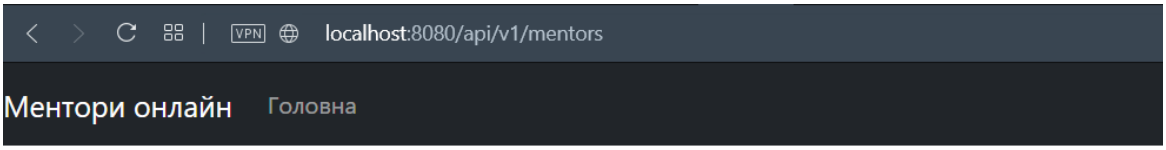

Рис. 3.13. Скріншот навігаційного меню

Кнопка «Ментори онлайн» є логотипом на який можна натиснути, натиснувши користувач залишиться на тій сторінці, де й був. Кнопка «Головна» відкриває головну сторінку «Головна». Це навігаційне меню на сторінці *navbar.html* реалізовано як фрагмент. Фрагмент у *Thymeleaf* – це невеликий

фрагмент коду, який можна включити в інші шаблони. Поширеною практикою веброзробки є створення багаторазових невеликих компонентів, таких як заголовок, нижній колонтитул, меню навігації та інші частини вебсайту, які повторюються на багатьох сторінках. Щоб визначити фрагмент *Thymeleaf*, потрібно використовувати атрибут *th*: *fragment*. Фрагменти можна визначити або в окремих файлах, або у загальному файлі. Навігаційне меню визначено в окремому файлі. Також хорошою практикою є розміщення всіх фрагментів у спеціальній теці, яка називається фрагментами, у каталозі шаблонів.

Файл *homeView.html* реалізує відображення сторінки «Головна», рис. 3.14.

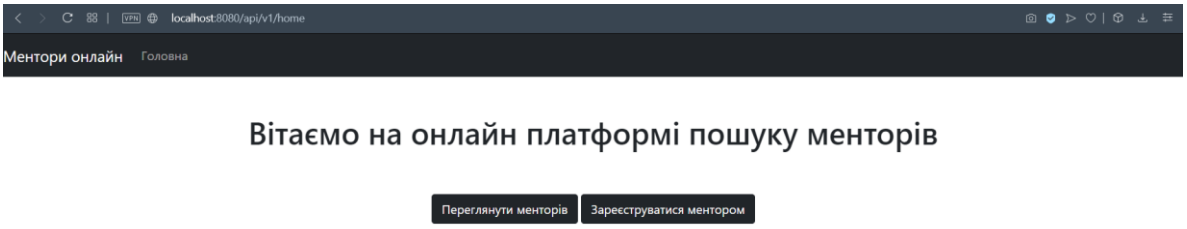

Рис. 3.14. Скріншот сторінки «Головна»

Для відображення навігаційного меню використовується, створений раніше фрагмент, за допомогою тегу *th*:*block* та тегу *th*:*include*. *th*:*block* – це контейнер атрибутів, який дозволяє розробникам шаблонів вказувати, які саме атрибути вони хочуть. На головній сторінці, під написом «Вітаємо на онлайн платформі пошуку менторів», знаходяться функціональні кнопки «Переглянути менторів» та «Зареєструватися ментором». Кнопка «Переглянути менторів» відкриває сторінку з усіма менторами – «Ментори», за допомогою посилання з тегом *th*:*href*="@{/*api*/*v*1/*mentors*}", який посилає *get* запит на *api* з таким мапінгом, а саме на клас *MentorsController* та його метод *getMentorById*. Кнопка «Зареєструватися ментором» відкриває сторінку з формою реєстрації ментора – «Реєстрація», за допомогою посилання з тегом *th*:*href*="@{/*api*/*v*1/ *sing-up*}", який посилає *get* запит на *api* за вказаним мапінгом, тобто на клас *AuthenticationController* та його метод *authentication*.

Файл *loginView.html* реалізує відображення сторінки «Логін», рис. 3.15. Сторінка забезпечує процедуру входу.

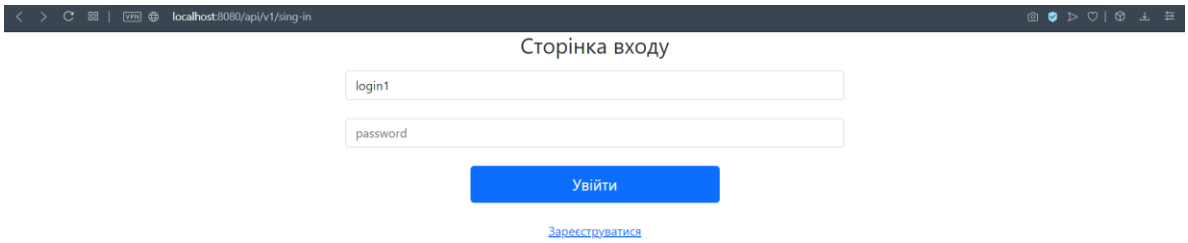

Рис. 3.15. Скріншот сторінки «Увійти»

Під написом «Сторінка входу» розташовано два поля для вводу логіну та паролю, а також кнопки «Увійти», якщо ментор не зареєстрований для нього є посилання «Зареєструватися» [8]. Поля реалізовані як <*input*> вебформи. Завдяки тегу *th:action="@{/api/v1/sing-in}* після натискання кнопки «Увійти» надсилається *post* запит на клас *AuthenticationController* та його метод *login*. А за допомогою тегу *th:object="${loginRequest}* запитом *post* передаються дані з полів «Логін» та «Пароль».

Посилання «Зареєструватися» за допомогою тегу *th:href="@{/api/v1/sing-up}"* відкриває сторінку «Реєстрація», рис. 3.16, яка реалізована у файлі *registerView.html*.

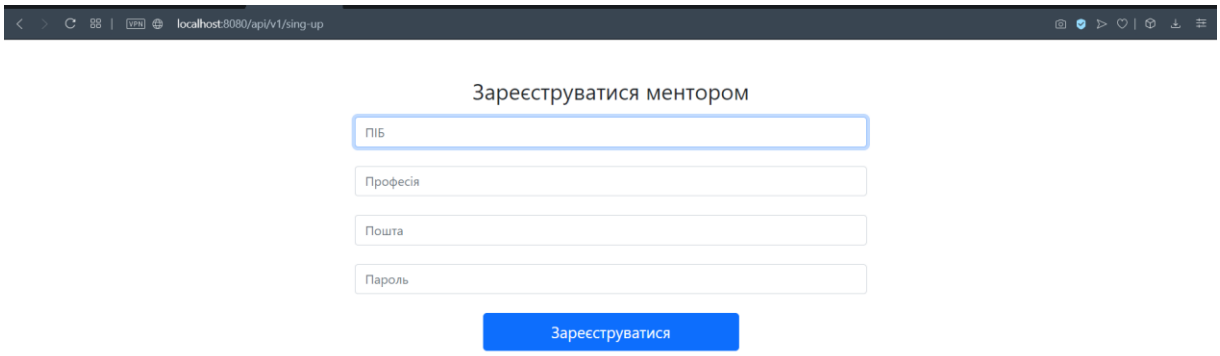

Рис. 3.16. Скріншот сторінки «Реєстрація»

Сторінка «Реєстрації» зустрічає користувача написом «Зареєструватися ментором» під яким розташована форма реєстрації. Форма складається з полів: «ПІБ», «Професія», «Пошта», «Пароль», Кнопка «Зареєструватися».

Вірно заповнивши поля, ментор буде зареєстрований та відкриється головна сторінка. Якщо поле буде не заповнено або заповнено невірно, то

користувача про це буде повідомлено рис. 3.18 та рис.3.19.

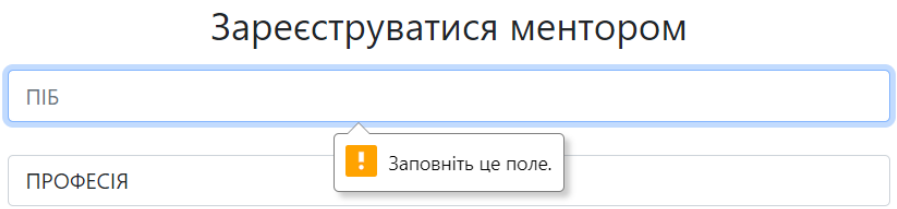

Рис. 3.18. Повідомлення користувача, що поле «ПІБ» не заповнене

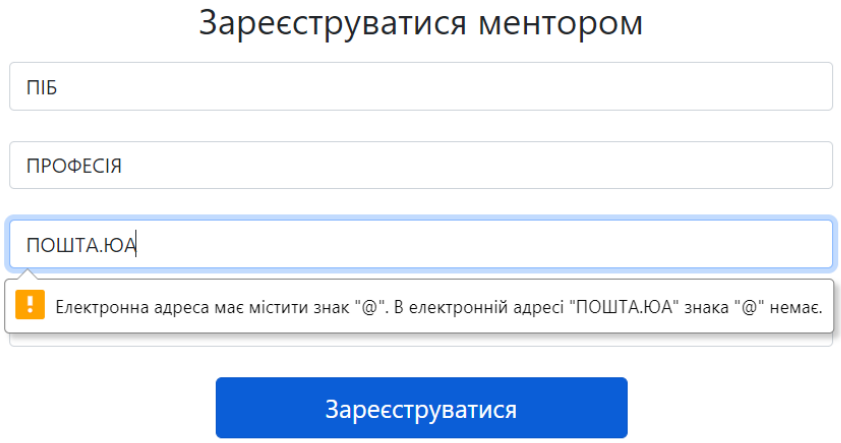

Рис. 3.19. Повідомлення користувача, що поле «Пошта» заповнене невірно

Файл *mentorsView.html* містить реалізацію сторінки «Ментори». Сторінка «Ментори» відображає менторів. На сторінці «Ментори» відображаються усі ментори, рис. 3.20.

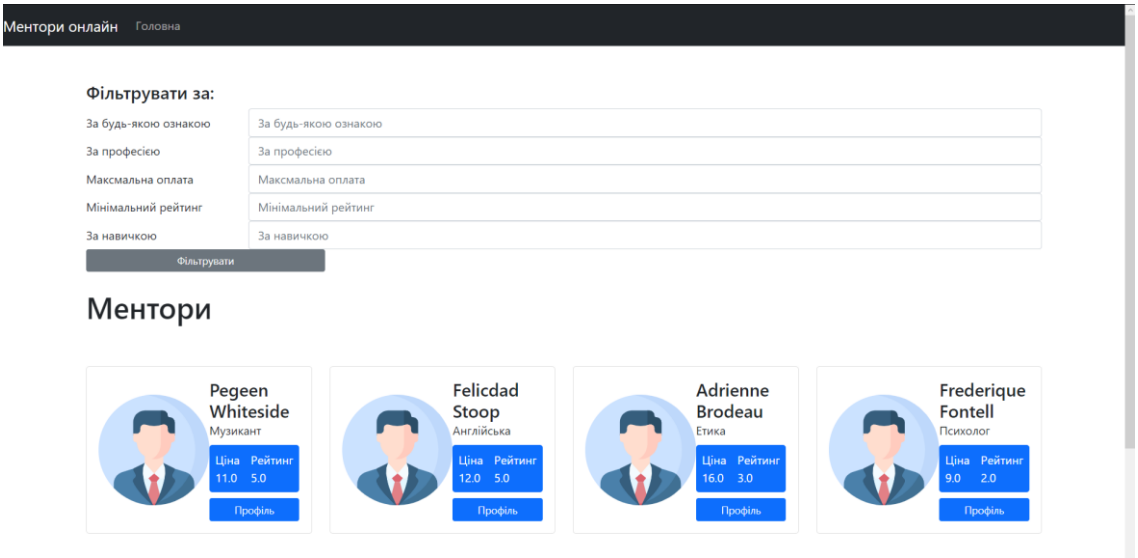

Рис. 3.20. Скріншот сторінки «Ментори»

Відображення ментора виглядає як картка з фотографією, прізвищем ім’ям та по батькові(якщо присутнє), рис. 3.21. Нижче ПІБ, описано професію ментора

за якою він може навчати. Нижче наведено ціну та рейтинг. А також кнопку «Профіль, яка відкриває профіль ментора.

Рис. 3.21. Відображення картки ментора

На сторінці «Ментори» вгорі знаходиться форма з фільтрами за якими можна відсортувати менторів, а саме за будь-якою ознакою: професією, максимальною оплатою, мінімальним рейтингом, навичкою.

Форма посилає *post* запит на метод *getMentors* контролера *MentorsController* та передає введені фільтри, рис. 3.24.

Рис. 3.22. Відображення форми фільтрування

Файл *profileView.html* містить реалізацію сторінки «Профіль». На сторінці «Профіль» можна переглянути детальну інформацію профілю ментора. Цю сторінку умовно можна поділити на інформаційну частину, рис. 3.23, та частину з коментарями, рис. 3.24.

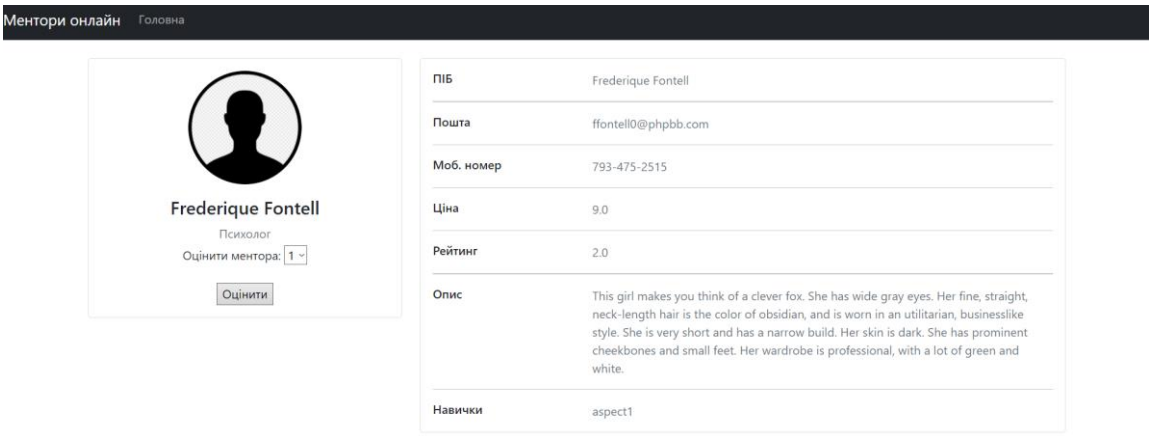

Рис. 3.23. Відображення інформаційної частини профілю

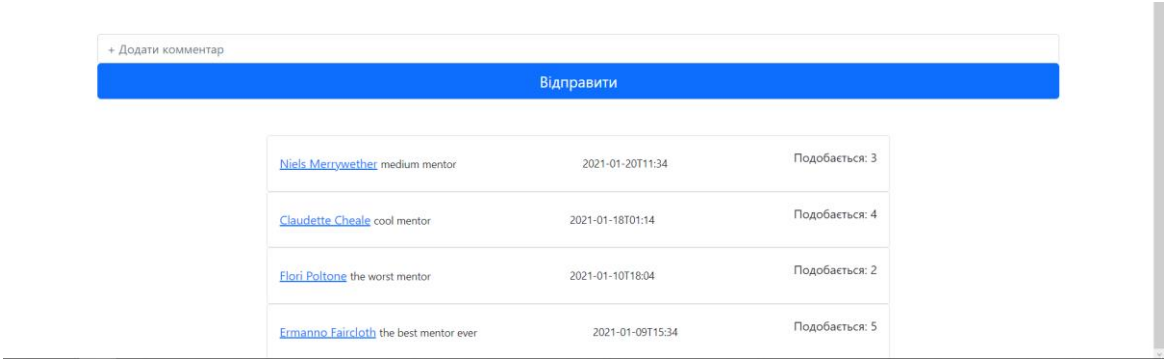

Рис. 3.24. Відображення коментарів профілю

В інформаційній частині наведено поля: «Фото», «ПІБ», «Професія», «Пошта», «Телефон», «Ціна», «Рейтинг», «Опис», «Навички ментора».

У частині з коментарями можна переглянути коментарі, відмітити як подобається або залишити коментар.

3.7. Висновки до розділу

Була реалізована онлайн платформа для пошуку менторів у навчанні. Визначено та розроблено схему бази даних *MySql*:

- *Users*;
- *Profiles*;
- *Reviews*;
- *Aspects*;
- *Profiles_aspects*;

– *Images*;

Розроблені класи для збереження даних у додатку, які знаходяться у пакеті *entity*:

– *Contact*;
– *Image*;
– *LifeAspect*;
– *Profile*;
– *Review*;
– *User*;

Застосовано трирівневу архітектуру *MVC* з використанням вебтехнологій *thymeleaf*. Сервісний рівень представлений класами:

– *AspectServiceImpl*;
– *UserServiceImpl*;
– *ReviewServiceImpl*;
– *ProfileServiceImpl*;
– *ImageServiceImpl*;
– *EncoderImpl*;

Розроблений рівень контролерів з класами:

– *ViewsController*;
– *MentorsController*;
– *AuthenticationController*;

А рівень відображень реалізований сторінками *html*:

– *Navbar*;
– *HomeView*;
– *LoginView*;
– *MentorsView*;
– *ProfileView*;
– *RegisterView*;

ВИСНОВКИ

Під час виконання дипломної роботи було проведено аналіз готових рішень онлайн платформ для пошуку менторів та встановлено, що платформи даного виду мають попит серед користувачів. Користувачі таких платформ бажають знайти собі ментора для набуття або підвищення практичних професійних навичок з різних сфер діяльності. Менторами можуть виступати люди з практичним досвідом у певному напрямку, з бажанням та вмінням ділитися своїм досвідом. Платформи для пошуку ментора у навчанні можуть містити менторів з конкретного напряму або платформи можуть містити менторів з будь-яких напрямів. Також під час аналізу було виокремлено функціонал, який повинна реалізовувати та надавати користувачам, онлайн платформа для пошуку ментора у навчанні:

- Перегляд всіх менторів;
- Фільтрація менторів;
- Реєстрація профілю ментора;
- Перегляд профілю ментора;

При розробці додатку було проаналізовано архітектури додатків серед:

- Мікросервісна;
- Монолітна;

Для додатка було обрано монолітну архітектуру за переваги:

- Простота розгортання та запуску монолітного додатка перед мікросервісним;
- Проблеми у програмі можна вирішити без серйозної зміни у коді;
- Простота реалізації;
- Ведення всієї кодової бази в одному місці та розгортання програми в одному місці;

Для збереження даних додатку було проаналізовано різні види баз даних та обрано реляційну. Реляційна база даних організовує дані в таблиці, які можна пов’язати на основі даних, загальних для кожної. Ця можливість дозволяє

отримати абсолютно нову таблицю з даних в одній або декількох таблицях за допомогою одного запиту. Після аналізу реляційних баз даних, обрано *MySql* базу даних через наступні переваги:

- Незалежність платформи;
- Можна розгорнути;
- Відкритий код та підтримка;
- Висока доступність;
- Масштабованість та гнучкість;

Під час розробки додатку було реалізовано рівні *MVC* архітектури, що розділили логіку програми за модулями:

- Контролер;
- Сервіс;
- Відображення;

Рівень контролерів приймає запити від користувача на отримання або додавання даних й повертає користувачеві відображення сторінок. Рівень сервісів реалізує логіку роботи з базою даних та забезпечення таких функцій як фільтрування менторів та перевірка паролів. Рівень відображення реалізує вебсторінки з інтерфейсом для користувача.

Результати роботи можна використовувати для проєктування вебдодатків на мові *Java*, для розробки онлайн платформи для пошуку ментора у навчанні, для навчання розробки вебдодатків на мові *Java*.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Starr*. *The Mentoring Manual*. – М.: *Ft Pr*, 2017. – 202 с.

2. Буч. Введение в *UML* от создателей языка / Гради Буч , Джеймс Рамбо, Ивар Якобсон. – М.: ДМК Пресс, 2015. – 496 *с*.

3. Лекция 11: Унифицированный язык визуального моделирования *Unified Modeling Language* (*UML*) // Национальный Открытый Университет "ИНТУИТ"|Бесплатное образование. Режим доступа: *http*://*www.intuit.ru*/*studies*/*courses*/2195/55/*lecture*/163, свободный (дата обращения: 12.03.2016 г.). – Заголовок с экрана.

4. Фиайли К. *SQL*. Руководство по изучению языка. – М., ДМК Пресс, 2014 г. – 454 с.

5. *Joshua*. *Effective Java*, 3*rd Edition*. – М.: *Addison Wesley*, 2017. – 412 *с*.

6. *Drupal*. Создание и управление сайтом; Символ-плюс – М., 2010. – 576 *с*.

7. Фрейен Б. *HTML5* и *CSS*3.Разработка сайтов для любых браузеров и устройств; Питер – Москва, 2014. - 304 *с*.

8. Гультяев А.К. Проектирование и дизайн пользовательского интерфейса / А.К. Гультяев, В.А. Машин. – М.: Корона-Принт, 2010. – 350 *с*

9. Тим К. *PHP* 5 и *MySQL*. Библия пользователя, *Wiley*, 2006. – 1216 с.

10. Брюс. Философия *java*, Питер, 2016. – 1168 *с*.

11. *Molinaro*. *SQL Cookbook, O'Reilly Media, Inc*., 2005. – 636 *с*.

12. *Sams*. *Teach Yourself SQL in* 10 *Minutes – Fifth Edition, Sams*, 2012. – 276 *с*.

13. Кристиан. *Java Persistence Api* и *Hibernate*, ДМК Пресс , 2017. – 1025 с.

14. ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – К.: Держстандарт України, 1995. – 37 с.

15. Слободян О. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2011. – 63 с.

ДОДАТОК А

ЛІСТИНГ КОДУ ОНЛАЙН ПЛАТФОРМИ ДЛЯ ПОШУКУ МЕНТОРА У НАВЧАННІ

```
@Entity @Table(name = "PROFILES")
public class Profile {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID") private Long id;
    @Column(name = "PROFESSION") private String profession;
    @Column(name = "RATING") private Double rating;
    @Column(name = "DESCRIPTION") private String description;
    @Column(name = "rate") private Double rate;
    @Column(name = "views") private Integer viewsCount;
    @ManyToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.MERGE}) @JoinTable(name = "PROFILES_ASPECTS", joinColumns = @JoinColumn(name = "ASPECT_ID"), inverseJoinColumns = @JoinColumn(name = "PROFILE_ID"))
    private Set<LifeAspect> lifeAspects;
    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "USER_ID") private User user;
    @OneToMany(mappedBy = "profileId", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<Review> reviews;
    public Profile(String profession, Double rating, String description, Double rate, Integer viewsCount,
                   Set<LifeAspect> lifeAspects, User user, List<Review> reviews) {
        this.profession = profession; this.rating = rating;
        this.description = description; this.rate = rate;
        this.viewsCount = viewsCount; this.lifeAspects = lifeAspects;
        this.user = user; this.reviews = reviews; }
```

```
    public Profile() {    }
    public void setViewsCount(Integer viewsCount) {
        this.viewsCount = viewsCount;    }
    public List<Review> getReviews() {
        return reviews;    }
    public void setReviews(List<Review> reviews) {
        this.reviews = reviews;    }
    public Long getId() {        return id;    }
    public void setId(Long id) {        this.id = id;    }
    public void setLifeAspects(Set<LifeAspect> lifeAspects) {
        this.lifeAspects = lifeAspects;    }
    public User getUser() {       return user;    }
    public void setUser(User user) {        this.user = user;    }
    public String getProfession() {        return profession;    }
    public void setProfession(String profession){this.profession = profession;}
    public Double getRating()
{ return rating;    }
    public void setRating(Double rating) {this.rating = rating;    }
    @Override
    public boolean equals(Object o) {
        if (this == o) {           return true;        }
        if (o == null || getClass() != o.getClass()) { false;       }
        Profile profile = (Profile) o;
        return rate.equals(profile.rate) && Objects.equals(id, profile.id) &&
Objects.equals(profession, profile.profession) && Objects.equals(rating, profile.rating)
&&
Objects.equals(viewsCount, profile.viewsCount) && Objects.equals(description,
profile.description) && Objects.equals(lifeAspects, profile.lifeAspects) &&
Objects.equals(user, profile.user);    }
    @Override
```

```
    public int hashCode() {return Objects.hash(getId(), getDescription(), getRating(), getRate(), getViewsCount(), getUser());    }
    @Override
    public String toString() {        return "Profile{" +", profession='" + profession + '\'' +", rating=" + rating +", description='" + description + '\'' +", rate=" + rate +", viewsCount=" + viewsCount +", lifeAspects=" + lifeAspects +", user=" + user + '}';  }}
@Entity
@Table(name = "USERS")
public class User implements Serializable {
    @Id    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "FULL_NAME")        private String fullName;
    @Column(name = "BIRTH_DATE")        private LocalDate birthDate;
                @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "AVATAR_ID")    private Image avatar;
    @Column(name = "TYPE")          @Enumerated(EnumType.STRING)
    private UserType userType;
    @Column(name = "PASSWORD")          private String password;
    @Column(name = "LOGIN")        private String login;
    @Column(name = "REGISTRATION_DATE")        private                LocalDateTime registrationDate;
    @Column(name = "LAST_VISIT_DATE")    private LocalDateTime lastVisitDate;
    public User(String fullName, UserType userType, Contact contact, Image avatar, String password, String login,
                LocalDateTime registrationDate, LocalDateTime lastVisitDate) {
        this.fullName = fullName;        this.userType = userType;
        this.contact = contact;          this.avatar = avatar;
        this.password = password;        this.login = login;
        this.registrationDate = registrationDate;        this.lastVisitDate = lastVisitDate;    }
    public User() {    }
```

```
    public User(String fullName) {    this.fullName = fullName;    }
    @Embedded     private Contact contact;
    public String getLogin() {        return login;    }
    public void setLogin(String login) {this.login = login;    }
    public LocalDateTime getRegistrationDate() {return registrationDate;    }
    public void setRegistrationDate(LocalDateTime registrationDate) {
        this.registrationDate = registrationDate;    }
    public LocalDateTime getLastVisitDate() {return lastVisitDate;    }
    public void setLastVisitDate(LocalDateTime lastVisitDate) {        this.lastVisitDate =
lastVisitDate;   }
    public String getPassword() {     return password;    }
    public void setPassword(String password) { this.password = password;    }
    public Long getId() {       return id;    }
    public void setId(Long id) {        this.id = id;    }
    public String getFullName() {  return fullName;    }
    public void setFullName(String fullName) {  this.fullName = fullName;}
    public LocalDate getBirthDate() {        return birthDate;    }
    public void setBirthDate(LocalDate birthDate) {this.birthDate = birthDate;    }
    public UserType getUserType() {        return userType;    }
    public void setUserType(UserType userType) {this.userType = userType;   }
    public Contact getContact() {        return contact;    }
    public void setContact(Contact contact) {    this.contact = contact;    }
    public Image getAvatar() {        return avatar;    }
    public void setAvatar(Image avatar) {       this.avatar = avatar;    }
    @Override
    public boolean equals(Object o) {
        if (this == o) {           return true;       }
        if (o == null || getClass() != o.getClass()) {return false;        }
        User user = (User) o;
        return Objects.equals(id, user.id) &&
```

```
Objects.equals(fullName, user.fullName) && Objects.equals(birthDate, user.birthDate) &&
Objects.equals(avatar, user.avatar) && userType == user.userType &&
Objects.equals(password, user.password) && Objects.equals(login, user.login) &&
Objects.equals(registrationDate, user.registrationDate) &&
Objects.equals(lastVisitDate, user.lastVisitDate) && Objects.equals(contact, user.contact);  }
  @Override  public int hashCode() {
    return Objects.hash(id, fullName, birthDate, avatar, userType, password, login, registrationDate,
        lastVisitDate, contact);  }
  @Override  public String toString() {
    return "User{" +id=" + id +", fullName='" + fullName + '\'' +  ", birthDate=" + birthDate +", avatar=" + avatar +", userType=" + userType +", contact=" + contact +'}';  }}
@Service
public class UserServiceImpl implements UserService {
  private final UserRepository userRepository;
  private final Encoder encoder;
  public UserServiceImpl(UserRepository userRepository, EncoderImpl encoder) {
this.userRepository = userRepository;
    this.encoder = encoder;  }
  @Override  @Transactional
  public void onApplicationEvent(AuthenticationSuccessEvent event) {
    String login = ((UserDetails) event.getAuthentication()
        .getPrincipal()).getUsername();
    User user = findByLogin(login);
user.setLastVisitDate(LocalDateTime.now());  update(user);  }
  @Override
  public Optional<User> findById(Long userId) {  return
userRepository.findById(userId);  }
```

```
    @Override
    public List<User> findAllByUserType(UserType userType) {       return
userRepository.findAllByUserType(userType);    }
    @Override    @Transactional
    public User save(User user) {     return userRepository.save(user);    }
    @Override       @Transactional
    public User update(User user) {userRepository.findById(user.getId())
            .orElseThrow(() -> new EntityNotFoundException(
String.format("User with such id:%d not found", user.getId())));
        return userRepository.save(user);    }
    @Override    @Transactional
    public void deleteById(Long userId) { userRepository.deleteById(userId);    }
    @Override    public User findByLogin(String login) {
        return userRepository.findByLogin(login)
            .orElseThrow(() -> new EntityNotFoundException(
                String.format("User with such login: %s not found", login)));    }
    @Override
    public User findByIdAndUserType(Long userId, UserType userType){
        return userRepository.findByIdAndUserType(userId, userType)
            .orElseThrow(() -> new EntityNotFoundException(
                String.format("User with such id: %d not exist", userId)));    }
    @Override
    public boolean login(String login, String password) {
        User user = userRepository.findByLogin(login).orElse(null);
        boolean isLogin = false;
        if (user != null) {
            isLogin = encoder.decodePassword(password, user.getPassword());       }
        return isLogin; }}
@Service
public class ProfileServiceImpl implements ProfileService {
    private final ProfileRepository profileRepository;
```

```
    private final EntityManager entityManager;
  @Autowired public ProfileServiceImpl(ProfileRepository profileRepository,
EntityManager entityManager) {
    this.profileRepository = profileRepository;
    this.entityManager = entityManager;  }
  @Override  public Profile findByUserIdAndUserType(Long id, UserType userType)
{
    return profileRepository.findByUserIdAndUserUserType(id, userType)
        .orElseThrow(() -> new EntityNotFoundException(
            String.format("Profile with such id: %d not exist", id)));  }
  @Override  public List<Profile> findAll(ProfileSpecification specification) {
    return profileRepository.findAll(specification);  }
  @Override  @Transactional
  public Profile update(Profile profile, Long profileId) {
    return entityManager.merge(profile);  }}
@Controller  @RequestMapping(path = "api/v1/sing-in")
public class AuthenticationController {
  private static final String LOGIN_PAGE = "loginView";
  private static final String HOME_VIEW = "homeView";
  private final UserService userService;
  public AuthenticationController(UserServiceImpl userService) {
    this.userService = userService;  }
  @GetMapping()  public String authentication() {  return LOGIN_PAGE;  }
  @PostMapping()  public String login(@ModelAttribute(name = "loginDto")
LoginRequest loginRequest) {  String view = LOGIN_PAGE;
    boolean isLogin = userService.login(loginRequest.getLogin(),
loginRequest.getPassword());
    if (isLogin) {  view = HOME_VIEW;  }
    return view;  }}
@Controller  @RequestMapping("api/v1/mentors")
@Api(value = "Mentors management system")
```

```
public class MentorsController {
   private static final String MENTORS_VIEW = "mentorsView";
   private static final String PROFILE_VIEW = "profileView";
   private static final String HOME_VIEW = "homeView";
      private final UserService userService;
   private final UserMapper userMapper;
   private final ProfileService profileService;
   private final ProfileMapper profileMapper;
   public MentorsController(UserMapper userMapper, UserService userService,
ProfileServiceImpl profileService,
                        ProfileMapper profileMapper) {
      this.userMapper = userMapper;            this.userService = userService;
     this.profileService = profileService; this.profileMapper = profileMapper;   }
   @GetMapping("/profile/{id}")
   public String getMentorById(@PathVariable Long id, Model model) {
model.addAttribute("mentor",              profileService.findByUserIdAndUserType(id,
UserType.MENTOR));       return PROFILE_VIEW;    }
   @GetMapping public String getMentors(Model model, @ModelAttribute(name =
"filter") MentorFilter filter,
 @RequestParam(defaultValue = "1") int page) {
 model.addAttribute("mentors",profileService.findAll(new
ProfileSpecification(filter)));
      model.addAttribute("currentPage", page);
      model.addAttribute("userType", "students");
     return MENTORS_VIEW;    }
   @PostMapping       public String saveMentor(@RequestBody @Valid UserDto
newMentor) {       User user = userMapper.map(newMentor);
      user.setUserType(UserType.MENTOR);
      return HOME_VIEW;    }
   @PutMapping("/{id}")
```

```
    public UserDto updateMentor(@RequestBody UserDto userDto, @RequestParam
Long id) {        userDto.setId(id);
        User user = userService.update(userMapper.map(userDto));
        return userMapper.map(user);    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteMentor(@PathVariable Long id) {
        userService.deleteById(id);
        return ResponseEntity.noContent().build();    }}
@Controller     @RequestMapping("api/v1")
public class ViewsController {
    private static final String HOME_VIEW = "homeView";
    private static final String REGISTER_PAGE = "registerView";
    @GetMapping("/home")      public String getHomeView() {
        return HOME_VIEW;    }
    @GetMapping("/sing-up")           public String getRegisterPage() {
        return REGISTER_PAGE;    }}
<!DOCTYPE html>    <htmllang="en"
xmlns:th="http://www.w3.org/1999/xhtml"><head>    <meta charset="UTF-8">
</head><body><div th:fragment="navbar">
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <a class="navbar-brand" href="#">Ментори онлайн</a>
        <button  class="navbar-toggler"  type="button"  data-toggle="collapse"  data-
target="#navbarNav"              aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">          <span class="navbar-toggler-icon"></span>
        </button>       <div class="collapse navbar-collapse" id="navbarNav">
          <ul class="navbar-nav mr-auto mt-2 mt-lg-0">              <li class="nav-item
active">    <a class="nav-link" th:href="@{/api/v1/home}"> <span
class="sr-only"></span>Головна</a>          </li></ul><span  class="navbar-text"
align="right"></span></div>  </nav></div></body></html>
<!DOCTYPE  html  SYSTEM  "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-
spring4-4.dtd"><html        lang="en"        xmlns="http://www.w3.org/1999/xhtml"
```

xmlns:th="http://www.w3.org/1999/xhtml"><head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width, initial-scale=1">

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1" crossorigin="anonymous"> <title>Головна</title></head>

<body><th:block th:include="fragments/navbar :: navbar"></th:block>

*

 <h1>Вітаємо на онлайн платформі пошуку менторів</h1>

 Переглянути менторів Зареєструватися ментором </body></html>*

<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-spring4-4.dtd"><html lang="en" xmlns="http://www.w3.org/1999/xhtml" mlns:th="http://www.thymeleaf.org"><head> <meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1"><link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/sign-in/"><link href="../assets/dist/css/bootstrap.min.css" rel="stylesheet">

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1" crossorigin="anonymous"><link href="css/signin.css" rel="stylesheet"> <title>Логін</title></head><body class="d-flex flex-column h-100"> <main class="form-signin"><form th:action="@{/api/v1/sing-in}" th:object="${loginRequest}" method="post"><h1 class="h3 mb-3 fw-normal">Сторінка входу</h1>

* <input type="login" id="inputEmail" class="form-control" name="login" placeholder="login" required autofocus>
<input type="password" id="inputPassword" class="form-control" name="password" placeholder="password"required>
 <button class="w-100 btn btn-primary btn-lg"*

```
type="submit">Увійти</button><br><br><a th:href="@{/api/v1/sing-up}">Зареєструватися</a></div></form></main></div></body></html>
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-spring4-4.dtd">
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.w3.org/1999/xhtml"><head>
<meta charset="utf-8"> <meta name="viewport" content="width=device-width, initial-scale=1">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-iJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1" crossorigin="anonymous"> <title>Ментори</title> </head><body><th:block th:include="fragments/navbar:: navbar"></th:block>
<br><br><div class="container">
<form th:action="@{api/v1/mentors}" th:object="${filter}" method="get"> <h4>Фільтрувати за: </h4> <div class="form-group row"> <label for="search" class="col-sm-2 col-form-label">За будь-якою ознакою</label><div class="col-sm-10"><input type="text" name="search" id="search" placeholder="За будь-якою ознакою" class="form-control"></div>
</div><div class="form-group row"><label for="profession" class="col-sm-2 col-form-label">За професією</label><div class="col-sm-10"> <input type="text" name="profession" id="profession" placeholder="За професією" class="form-control"></div></div><div class="form-group row"><label for="maxRate" class="col-sm-2 col-form-label">Максмальна оплата</label><div class="col-sm-10"><input type="text" name="maxRate" id="maxRate" placeholder="Максмальна оплата" class="form-control"></div></div><div class="form-group row">
<label for="minRating" class="col-sm-2 col-form-label">Мінімальний рейтинг</label><div class="col-sm-10"><input type="text" name="minRating" id="minRating" placeholder="Мінімальний рейтинг" class="form-control"></div></div>
```

* <label for="aspectsNames" class="col-sm-2 col-form-label">За навичкою</label>*

* <input type="text" name="aspectsNames" id="aspectsNames" placeholder="За навичкою" class="form-control"> *

*<button class="w-25 btn btn-secondary btn-sm float-right" type="submit">Фільтрувати</button> </form>
*

* <h1>Ментори</h1> *

* <img alt="avatar img"src="https://www.flaticon.com/svg/vstatic/svg/3135/3135715.svg?token=exp=1620559631~hmac=100105d67d9a391628471caa8c4e1a83"class="rounded" width="150"> *

<h4 class="mb-0 mt-0" th:text="${mentor.user.fullName}"></h4> Ціна </div><div class="d-flex flex-column">Рейтинг <span class="number3" h:text="${mentor.rating}"> <a class="btn btn-sm btn-primary w-100 ml-2"th:href="@{/api/v1/mentors/profile/{id}(id=${mentor.id})}">