НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»

УДК _____ ___

«До захисту допущено»

Завідувач кафедри
_____ *Стіренко С.Г.*
(підпис) (ініціали, прізвище)

“ ” _____ 20***20*** р.

Магістерська дисертація

зі спеціальності: *121. Інженерія програмного забезпечення*_____
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *121. Інженерія програмного забезпечення комп'ютерних систем*

на тему: Програмна платформа підтримки процесів дистанційного навчання

Виконав: студент 6 курсу, групи ІП-94мп
(шифр групи)

Сулима Олександр Сергійович
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник проф., д.т.н., проф.Писарчук О. О.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

ЗАВДАННЯ
на магістерську дисертацію студента
Сулими Олександра

Науковий керівник.
доктор технічних наук, професор, лауреат державної премії України в галузі науки і техніки, професор кафедри обчислювальної техніки факультету інформатики та обчислювальної техніки НТУУ «КПІ» імені Ігоря Сікорського Писарчук Олексій Олександрович.

Тема.
Програмна платформа підтримки процесів дистанційного навчання.

Затверджені наказом по університету від «26» жовтня 2020 №3133-с

Мета досліджень.
Підвищення ефективності дистанційної форми навчання за рахунок створення і впровадження спеціалізованої програмної платформи.

Предмет: методи і моделі побудови баз даних та сховищ даних.
Об'єкт: процеси розподіленого обміну інформацією.

Перелік завдань, які необхідно розробити.
1. Аналіз відомих методологічних основ щодо реалізації процесів дистанційного навчання з використанням технологій розподіленого збереження даних.
2. Аналіз відомих технологічних рішень щодо програмної підтримки процесів дистанційного навчання.
3. Розробка методики та технології побудови спеціалізованих програмних платформ підтримки процесів дистанційного навчання.
5. Розробка програмної платформи підтримки процесів дистанційного навчання.
6. Оформлення результатів розробки програмної платформи підтримки процесів дистанційного навчання у формі стартап-проекту.

Технічні умови.
Результатом досліджень має стати програмна платформа підтримки процесів дистанційного навчання.
Програмний комплекс має реалізувати наступний перелік вимог.
1. Програмний комплекс, побудований на мікро серверній архітектурі.
2. Власний захищений сервіс для зберігання та поширення даних користувача технологією OAuth2.
3. Систематизоване зберігання та доступ до навчально-методичних матеріалів навчальних дисциплін. Під матеріалами слід вважати поняття "курс". Кожен курс, в свою чергу, поділений на лекції.

4. Автоматизований облік звітності за навчальними заняттями та надання форми для опитування студентів.

5. Реалізація розширеного пошукового механізму для курсів та лекцій.

6. Впровадження аспектно-орієнтованих компонентів, таких як: відправлення email повідомлення про початок або завершення курсу.

7. Інтерактивне створення коментарів студентами до кожної лекцій.

8. Систематизовані дії користувача в залежності від його ролі: викладач (адміністратор), студент (авторизований користувач), анонім (не авторизований користувач).

9. Асинхронна взаємодія між сервісами для поновлення інформації користувача.

10. Інтеграція з Ansible та Docker технологіями задля полегшення розгортання усієї платформи як кінцевого продукту.

11. Інтеграція з Jenkins платформою задля автоматизації розгортання платформи на віддаленому сервері.

Перелік графічного матеріалу.

Структурна схема програмного продукту.
Блок-схема алгоритму функціонування програмного продукту.
Юзкейс-діаграма, діаграма класів.
Програмна документація.
Презентаційні матеріали.

Орієнтований перелік публікацій.

1. IT lessons [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://ruseller.com/lessons.php?id=666
2. Web-architecture [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://tproger.ru/translations/web-architecture-101/
3. Spring [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/333756/
4. Gradle Build Tool [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://gradle.org/
5. Portal IT [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://dou.ua/lenta/articles/portrait-2016/
6. PostgreSQL [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://www.postgresql.org/docs/9.5/functions-json.html
7. Security Filters [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/346628/
8. Boot Spring [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/352954/
9. Microservices [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/company/otus/blog/413567/

План досліджень
(зміст роботи та вихідні дані до календарного плану).

ВСТУП.
1. РОЗГЛЯД НАВЧАЛЬНИХ СИСТЕМ.
1.1. Аналіз загальні відомостей про навчальні системи.
1.2. Аналіз та розгляд навчальних систем, побудованих на ІТ технологіях.
1.3. Аналіз існуючих механізмів дистанційного ІТ навчання.
1.4. Формалізація задачі та постановка часткових задач досліджень.
ВИСНОВОК ДО РОЗДІЛУ 1.

2. АРХІТЕКТУРА ПРОГРАМНОГО КОМПЛЕКСУ ДИСТАНЦІЙНОГО НАВЧАННЯ.
2.1. Опис технологічної платформи.
2.1.1. Архітектура MVC шаблону.
2.1.2. Компоненти OAuth2 сервісу.
2.1.3. Взаємодія мікро сервісів та інженерія вимог.
2.1.4. Spring Framework та його компоненти.
2.2. Опис збереження даних.
2.3. Опис компонентів побудови та розгортання кінцевої платформи.
2.3.1. Технологія Docker
2.3.2. Технологія Ansible
ВИСНОВОК ДО РОЗДІЛУ 2.

3. РОЗРОБКА ТА ПОКРАЩЕННЯ ПРОГРАМНОГО ДИСТАНЦІЙНОГО НАВЧАННЯ.
3.1. Покращення захисту даних користувача.
3.2. Аспектно-орієнтована складова.
3.3. Побудова та розгортання платформи на віддаленому сервері.
3.4. Пошуковий механізм.
3.5. Аналітика та збір відгуків користувачів.
ВИСНОВОК ДО РОЗДІЛУ 3.

4. ОПИС РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.
4.1. Демонстрація практичних можливостей розробленого програмного комплексу.
4.2. Програмна документація.
4.3. Стартап: «Програмна платформа підтримки процесів дистанційного навчання».
ВИСНОВОК ДО РОЗДІЛУ 4.
ВИСНОВКИ.
СПИСОК ЛІТЕРАТУРИ.

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Програмна платформа підтримки процесів дистанційного навчання

студентом: Сулимою Олександром Сергійовичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 85 аркушів основного тексту, 31 ілюстрації та 11 таблиць. При підготовці використовувалася література з 18 різних джерел.

Актуальність. Актуальність дослідження магістерської дипломної роботи полягає в тому, що з розвитком ІТ структури в країні, так званій “діджиталізації”, все більше набуває попиту принципово нова галузь, яка зветься програмісти, або ж розробники ПЗ.

За статистикою, все більше людей прагнуть змінити свій вид діяльності та остаточно перейти до програмування. Кожен третій не знає, які саме потрібні технології вивчати задля підкорення мети стати “айтішником”. А й подекуди взагалі вивчається не те, що дійсно стане йому в нагоді.

Тому існує необхідність заміни застарілої та мало ефективної системи дистанційного навчання, яке може прискорити вивчення матеріалів, використовуючи знання вже досвідчених осіб.

Мета і завдання дослідження. Необхідно виділити мету магістерської дипломної роботи, а саме продовження розробки сучасної платформи для впровадження онлайн-навчання в обраних користувачем областях ІТ.

Об’єкт дослідження – процеси розподіленого обміну інформацією.

Предмет дослідження – методи і моделі побудови баз даних та сховищ даних.

Методи досліджень. Для досягнення поставлених в магістерській роботі задач використано методи побудови програмного додатку на основі мікросервісної архітектури.

Наукова новизна одержаних результатів роботи полягає у наступному:

- Побудована інфраструктура та механізм взаємодії мікросервісів, із застосуванням необхідних покращених шаблонів.
- розроблено програмний продукт платформу яка надає рішення для трейдингу транспортними засобами з використанням механізму, описаного вище.

Проведене дослідження дає можливість динамічної конфігурації платформи і надання різних рішень, в залежності від бізнес вимог.

Особистий внесок здобувача. Дослідження магістра - це самостійно виконана робота, яка відображає особистий авторський підхід та особисто отримані теоретичні та прикладні результати, пов’язані з конструкторською та інженерною архітектурою коду програми на основі архітектури мікросервісу, що працює в хмарному середовищі.

Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність. Отримані результати можуть використовуватися у майбутніх дослідженнях за напрямками:

- оптимізація роботи окремих мікросервісів за рахунок горизонтального масштабування.
- Можливість швидкого розгортання інфраструктури мікросервісів на різних типах хмарних середовищ.

Конференції. Інтернет-конференція konfirencia-online. Технічний дизайт, розробка та тестування додатків на основі мікросервісної архітектури, з використанням фреймворку Spring Cloud: матеріали онлайн конференції. (10 грудня 2020 р.), 2020.

Ключові слова.

Дистанційне навчання, мікросервісна архітектура, цифрова платформа, підтримка процесів навчання

Пояснювальна записка
до дипломного проекту

на тему: «Програмна платформа підтримки процесів дистанційного навчання»

Київ – 2020

ЗМІСТ

ВСТУП…. 3
РОЗДІЛ 1 РОЗГЛЯД НАВЧАЛЬНИХ СИСТЕМ 4
1.1. Аналіз загальних відомостей про навчальні системи 4
1.2. Аналіз та розгляд навчальних систем, побудованих на ІТ технологіях 6
1.3. Аналіз існуючих механізмів дистанційного ІТ навчання 10
1.4. Формалізація задачі та постановка часткових задач досліджень 14
ВИСНОВОК ДО РОЗДІЛУ 1 16
РОЗДІЛ 2 АРХІТЕКТУРА ПРОГРАМНОГО КОМПЛЕКСУ ДИСТАНЦІЙНОГО НАВЧАННЯ 17
2.1. Опис технологічної платформи 17
2.1.1. Архітектура MVC шаблону 17
2.1.2. Компоненти OAuth2 сервісу 20
2.1.3. Взаємодія мікро сервісів та інженерія вимог 24
2.1.4. Spring Framework та його компоненти 27
2.2. Опис збереження даних 34
2.3. Опис компонентів побудови та розгортання кінцевої платформи 37
2.3.1. Технологія Docker 40
2.3.2. Технологія Ansible 43
ВИСНОВОК ДО РОЗДІЛУ 2 45
РОЗДІЛ 3 РОЗРОБКА ТА ПОКРАЩЕННЯ ПРОГРАМНОГО ДИСТАНЦІЙНОГО НАВЧАННЯ 46
3.1. Покращення захисту даних користувача 46
3.2. Аспектно-орієнтована складова 49

3.3. Побудова та розгортання платформи на віддаленому сервері 51
3.4. Пошуковий механізм 55
3.5. Аналітика та збір відгуків користувачів 57
ВИСНОВОК ДО РОЗДІЛУ 3 58
РОЗДІЛ 4 ОПИС РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……...59
4.1. Демонстрація практичних можливостей розробленого програмного комплексу. 59
4.2. Програмна документація. 68
4.3. Проектування інтерфейсу. 69
4.4. Розроблення стартап-проекту. 69
4.4.1. Опис ідеї проекту 71
4.4.2. Технологічний аудит ідеї проекту 74
4.4.3. Аналіз ринкових можливостей запуску стартап-проекту 75
4.4.4. Розроблення маркетингової програми стартап-проекту 80
ВИСНОВОК ДО РОЗДІЛУ 4 82
ВИСНОВКИ 83
СПИСОК ЛІТЕРАТУРИ 84

ВСТУП

Задля поглиблення успіхів у дистанційному навчанні, а також розширення аудиторії відносно вдосконалення процесу навчання, на сьогоднішній день приділяється велика увага. Не мало важлива, грошова компенсація, тобто заробітна плата, є головною рушійною силою більшості з тих, хто готовий відмовитися від своєї старої професії, від того, на що навчався і перейти до вивчення нових IT технологій XXI століття.

У цьому сенсі питання поліпшення процесу навчання в області IT стоїть на першому місці в багатьох компаніях. Актуальність та продовження вивчення теми полягає в тому, що з придбанням все більшого числа людей знань про IT буде забезпечений процес цифрового зростання країни.

Слід зазначити, що більшість з тих, хто вже працює в сфері IT, не знають, скільки коштує їхня робота на українському ринку IT. По цій темі необхідно замінити застарілу систему навчання людей в області IT і адаптувати їх мислення до нової грошово-кредитної політики.

Виходячи з попередніх пояснень, необхідно виділити мету магістерської дипломної роботи: аналіз, дослідження і продовження розробки сучасної платформи для впровадження онлайн-навчання в обраних користувачем областях IT. Задля досяглення цілі магістерської дипломної роботи, були поставлені наступні цілі:

- аналіз та покращення поточної системи;
- дослідження та забезпечення захисту даних користувача;
- покращення розгортання системи в один клік;
- аспектне программування як частина сервісу;
- дослідження засобів, які стануть у нагоді при реалізації платформи;
- реалізація платформи.

РОЗДІЛ 1
РОЗГЛЯД НАВЧАЛЬНИХ СИСТЕМ

1.1. Аналіз загальних відомостей про навчальні системи

Інформаційні технології - це вираз, який всім давно знайомий, технології вже точно характеризують наше життя і потреби сучасного людства. Однак питання про те, що таке інформаційні технології, може поставити багатьох в глухий кут.

Тому інформаційні технології (IT) - це набір методів та інструментів, які використовуються для збору, зберігання, передачі та обробки інформації через веб-портал, званий Інтернет. Сьогодні така діяльність людини сильно залежить від цих технологій і тому вимагає постійного розвитку.

Багато фахівців, звані IT-фахівцями, працюють в комп'ютерній сфері, а також «програмісти» або «IT-фахівці», їх робота так чи інакше пов'язана з комп'ютерами, програмами, виконуваним кодом. Спробуємо виділити кілька груп з них:

1. Фахівці серверної частини, або ж “системні адміністратори”.
2. Фахівці внутрішньої розробки, або ж “Software engineer”, що в перекладі означає той, хто створює програмне забезпечення.
3. Фахівці, що працюють з готовими інформаційними продуктами. Спеціалісти цього рівня переважно називаються “Тестувальники ПЗ”.

Так чи інакше, у кожної з цих галузей є свої розподілення на підгрупи, але це вже інша історія, для якої немає місця тут. В руках перших двох категорій знаходиться майбутнє комп'ютерних технологій, від них залежить, як людство буде передавати і отримувати інформацію.

Достатньо подивитися на будь-який офіс: жоден з них не працює без ПК, ноутбука або іншого. Більшість користується ПК задля будь-яких цілей, які сприяють поширенню бізнесу. Багато компаній, які навіть не займаються

інформаційними технологіями, мають в своєму штаті співробітників, які знаються на обчислювальних пристроях. Це вказує на велику потребу в IT-фахівцях. У той же час можна обговорити розшарування IT-фахівців на тих, хто дуже популярний і менш популярний серед роботодавців. Це пов'язано з тим, що деякі області в області інформаційних технологій мають особливий пріоритет і тому концентрують щодо більше ресурсів для розвитку.

Тому, згідно зі статистикою, експерти вказують, що в найближчі роки буде великий попит на фахівців з розробки веб-порталів. Слід також зазначити, що інформаційні технології - це швидко зміцнюване середовище, яке з часом показує, що кількість технологічних новинок тільки збільшується. Різноманітні проекти і розробки з'являються тут майже кожен день.

Здається, ніхто не сумнівається, що рано чи пізно програмісти витіснять добру половину сьогоднішніх професій. Але те, що буде з самими програмістами, теж незрозуміло.

По-перше, програмування низького рівня не менш підлягає автоматизації, ніж звичайна офісна робота. По-друге, IT-сектор не є гумовим, і рано чи пізно ринок буде насичений фахівцями, і зважаючи на постійний приплив нових працівників, конкуренція серед них буде посилюватися.

Насправді цей процес вже триває: незважаючи на відновлення економічного зростання, кількість вакансій у російських IT-компаніях у 2017 році навряд чи зросла.

Таким чином, необхідно виділити мету магістерської дипломної роботи, а саме: продовження розробки сучасної платформи для впровадження онлайн-навчання в обраних користувачем областях IT.

1.2. Аналіз та розгляд навчальних систем, побудованих на IT технологіях

Протягом розвитку IT-технологій, поступово в країну надходили новітні сфери праці, які вимагали знань молодих і талановитих програмістів. Дохід засновників IT-компаній спочатку був необмеженим: низька заробітна плата в Україні поєднувалася з поганими умовами праці, але для роботодавців це була «золота жила».

Наприклад, в 2004 році програміст отримав 2 долари за годину роботи і продав цю годину за 16. Тому програмісти стали приймати всіх, хто може написати програму «Hello World», і всіх оцінювачів. хто, принаймні, знає, як включити комп'ютер. Цей період тривав приблизно до 2008 року. Було мало людей, готових увійти в IT з інших спеціальностей, тому що доходи розробника росли, але дуже повільно, наприклад, для майстра кондиціонерів, на жаль, немає він прибув.

У 2008 році була криза, все було звільнено, включаючи програмістів, але, на довершення всього, показник виріс, і, нарешті, програмісти почали заробляти набагато більше. Отже, потік заявників і, як наслідок, вимоги до них також почали зростати. Таке зростання триває і донині.

Якщо раніше для успішного працевлаштування були потрібні теоретичні знання, то тепер новачкові необхідно мати хоча б деякий досвід. Тобто, щоб отримати роботу програміста, потрібно мати досвід написання реальних, хоча б невеликих, некомерційних програм. Тестер: підтверджений досвід тестування додатків.

IT-системи щільно увійшли в наше життя. Потужні і складні програмні продукти використовуються в самих різних сферах. При цьому багато хто забуває, що з'явилися IT-системи не просто так, як програмні продукти, які потрібно продавати і впроваджувати, а як інструменти організації та автоматизації праці.

І дуже важливо пам'ятати при виборі та впровадженні IT-систем, що первинний важіль тут - праця, а не програмне рішення. Я не один раз стикався з тим, що люди вибирали програму просто тому, що: "вона сподобалася". В результаті з'являються спроби "натягнути" процессное виробництво, наприклад, роботу молокозаводу, на ERP-систему, призначену для дискретного виробництва (збірка виробів).

Існуючі підходи та IT системи:

1) Застосовується найнижчий рівень. Це програмні рішення, які кожен працівник використовує індивідуально на своєму робочому місці для виконання деяких місцевих завдань. Це може бути AutoCAD, Mathcad, Photoshop, Corel Draw тощо. Важливо розуміти, що прикладні системи не найпростіші або «найгірші», оскільки вони знаходяться на нижчому рівні. Ці програмні рішення можуть бути дуже складними і необхідними в роботі фахівців. Вони мають нижчий рівень лише з точки зору організації праці в цілому, тобто призначені для індивідуального використання. Навіть якщо ці продукти дозволяють певну форму співпраці, вони все-таки в першу чергу призначені для роботи фахівця, а деякі варіанти спільного використання є допоміжними. тому такі системи також застосовуються.

2) Системи спільного вирішення певних завдань. Ці системи призначені для вирішення певних проблем, пов’язаних із певною сферою діяльності. Це можуть бути CRM, кадрові системи, MRP, SCM системи тощо. Кожна з цих систем має свою вузьку функціональність, вона призначена для вирішення певного кола спеціалізованих завдань. Програмні продукти цієї категорії можуть бути інтегровані один з одним або з програмними рішеннями нижчого рівня. Наприклад, системи людських ресурсів часто інтегруються з MS Word. І CRM інтегрується з індивідуальними програмами відстеження та запису дзвінків.

3) ERP-системи. Програмне забезпечення, розроблене для вирішення широкого спектру завдань та поєднання роботи різних відділів компанії в

єдиній інформаційній системі. ERP-системи можуть включати спеціалізовані рішення другого рівня (CRM, MRP, SCM тощо) або можуть бути інтегровані з спеціалізованими програмними продуктами, розробленими для певної сфери діяльності.

4) Системи BPMS. Що це, я детально написав у статті "Що таке БПМС". Коротко пам’ятайте: це безпосередньо система реєстрації бізнес-процесів. Я ставлю їх на найвищий рівень ієрархії, тому що вважаю БПМС системою організації праці в "найчистішій" формі. У той же час БПМ може бути частиною будь-якої системи управління роботою, як другого рівня (CRM, HR, MRP, SCM), так і третього рівня (ERP). Анотації BPMS інтегровані в програмні продукти, що дозволяє нам стандартизувати та спростити організацію підходу до організації роботи на основі процесу.

5) Безкласові системи. Крім того, всі існуючі системи поділяються на дві категорії: з розробленою методологією і без неї. Що стосується розробленої методології, з назви системи вже зрозуміло, для чого її можна використовувати. Прикладами є CRM, MRP, HR, ERP.

Що стосується нерозвиненої методології, назва також не дуже інформативна і має загальний характер ("електронне управління документами", "управління товарами", "корпоративний портал"). Основна особливість таких програмних рішень полягає в тому, що вони не мають чіткої спеціалізації чи обмежень. Вони намагаються охопити якомога більше сфер діяльності, але в кожній окремій області вони відчувають себе некомфортно або недорозвинені. Використання таких рішень виправдано в певних випадках, але щоб зрозуміти, підходить це рішення чи ні, слід уважно вивчити обрану систему глибоко. Виходячи з назви та опису, неможливо точно визначити сферу застосування такого товару.

Швидше за все, люди цінують свободу, яку надає галузь: тут можна вибрати близьку по духу компанію, будь то міжнародна корпорація або невеликий стартап, і не хвилюватися щодо свого професійного зростання.

Коли людина тривалий час робить те саме, вона починає нудьгувати. Це стосується не тільки розробників, а й роботи взагалі. Тому важливо, щоб працівники професійно розвивалися, це додасть їм додаткової мотивації та натхнення для отримання нових знань.

Тепер можна побачити багато статей про "баланс між робочим життям" з порадами, як організувати своє життя так, щоб було достатньо часу для роботи та життя. Але насправді ми не завжди можемо цього досягти. Є обов'язки, строки та понаднормовий час, і все це стало звичним явищем у компаніях.

Для деяких інженерів саме «баланс робочого життя» підтримує їх у компанії. Це особливо важливо для тих, хто має сім'ю та дітей та хоче провести з ними більше часу.

Отже, з усього вищезазначеного, необхідно виділити об’єкт аналізу та досліджень магістерської дисертації, а саме: процеси розподіленого обміну інформацією. Предметом цього об’єкту являються методи і моделі побудови баз даних та сховищ даних.

1.3. Аналіз існуючих механізмів дистанційного IT навчання

В області IT ефективне самонавчання є одним з найбільш важливих навичок, але одним з найменш розвинених. З постійними змінами і розвитком технологій ефективне самоосвіта допомагає тримати багато двері відчиненими, регулярно оновлюючи свої знання.

Але що саме означає вчити щось ефективно? На мій погляд, ефективне навчання - це односторонній процес навчання і розуміння важливої інформації для застосування цих знань у подальшому житті. Це сфокусований процес, в ідеалі націлений на розуміння і освоєння нових знань раз і назавжди. Проблема з підходом, який знає більшість програмістів, яких я знаю (включно зі мною в минулому), полягає в тому, що вони вчаться досить для вирішення проблеми. Ця стратегія працює добре, але її недостатньо для майбутніх ситуацій, і якщо нові знання не використовуються якийсь час, вони забуваються і час вважається витраченим даремно. Це поверхове знання буде важко застосувати до інших подібних завдань в майбутньому.

Читання книг від початку до кінця, перегляд звіту або відео дає помилкове відчуття розуміння і запам'ятовування, створює в пам'яті складні острівці поверхневих знань.

Якщо десять років тому розповідь про журналіста, який опанував основи програмування та створив власні ресурси власними силами (наприклад, це зробив у свій час Костянтин Бочарський, який 10 років працював над «Секретом компанії ", а потім створили сервіс самостійно) і здивували Сьогодні багато людей опановують навички програмування, від дизайнерів та журналістів до економістів та засновників власних компаній.

Взяти хоча б американського бездомного Лео Гранда, який навчився кодить за чотири місяці і запустив мобільний додаток. Багато більш традиційних з бізнесу і в Росії: Костянтин Шадрін, співзасновник IT-агентства, а в минулому керівник групи в консалтинговій компанії вирішив, що без знання предмета неможливо запустити власний IT-стартап, і пройшов базовий

курс HTML і CSS. Таке ж думки розділяє і Олег Юсупов, засновник digital-агентства, який пройшов курси на шляху створення власної компанії, щоб краще розбиратися в предметі, а не «просто наймати людей з боку».

Шадрін і Юсупов вважали за краще вчитися програмуванню в Moscow Coding School (компанія позиціонує себе як «школа нового зразка, де круті девелопери з прогресивних інтернет-компаній і стартапів долучають до програмування креативних новачків»), однак багато, в тому числі згодом професійні програмісти, починають шлях самостійно - з книг і безкоштовних онлайн-уроків.

Наприклад, до такого висновку прийшли дослідники HackerRank, онлайн-платформи тестових завдань з програмування, яка в кінці січня 2018 року представила річний про навичках і кваліфікації розробників. В основу дослідження лягли результати опитування 39,4 тис. Розробників з 17 країн світу, зареєстрованих на платформі.

У житті та творчості будь-якої людини є проблеми різного характеру. Коли компанія, в особі керівництва чи колег, допомагає працівникам, вони бачать і цінують це. Коли працівник може розраховувати на допомогу, це, безумовно, підвищує його лояльність і відштовхує його від розгляду інших пропозицій.

Останнім часом все важче стає займатися IT. З іншого боку, як людина, що бере активну участь в розробці, я не можу не зауважити, що в галузі існує постійна нестача персоналу. Подивимося, чи варто починати вивчати інформаційні технології на курсах і що від них очікувати.

Прямо зараз, величезний приплив "айтішників". З одного боку, економічна криза, а в IT зарплата в доларах, більше нормальних офісів, добре обладнане робоче місце, чіткий графік роботи. Мотивація людей, що прагнуть до IT, проста і зрозуміла. Ще раз багато курсів з рекламою в метро, в яких йдеться

про те, що IT-шники в метро не ходять. Середня людина народжується з ідеєю: «Ну і що? Чим я гірше? Отже, є проста поведінка, щоб піти на підготовчі курси IT. Давайте почнемо з великомасштабних проблем, які стають ключовими в країні:

1. Дефіцит

На ринку праці бракує програмістів. Крім того, що для України добре, це дефіцит не на внутрішньому ринку, а на глобальному. Тобто, якщо кількість програмістів в Україні раптом подвоїться, це не сильно вплине на український ринок, і зовсім не вплине на світовий ринок. Але в той же час це матиме дуже позитивний вплив на середню заробітну плату в Україні та економіку загалом.

2. Вчити програмістів дорого.

Хороший учитель - це штучний і дефіцитний товар. Тільки любов до краси може змусити програміста йти викладати. Навіть якщо панує любов до краси, тобто обмеження кількості учнів. В результаті ми отримуємо високу вартість навчання для одного фахівця.

3. Мовний бар'єр.

Зараз для проходження 99% онлайн курсів потрібні знання англійської.

Існує три типи студентів, які:

1) вони вчаться не знаючи цілі, аби пройти

2) вони вчаться лише в тому випадку, якщо їх мотивують «морква» ззаду або спереду, тобто лише для конкретної цілі

3) вони не навчатимуться, як би їх не мотивували

Аргумент "вони заплатили свої гроші, тож навчатимуться" працює лише з першого разу. Хто б не навчався на курсах англійської мови, тепер киває, пам’ятаючи, яка частина студентів заплатила за заняття, а потім пропустила чи вивчила «тяп-блап».

Я чув про подібне оцінювання працівників на роботі, єдиними крайніми варіантами було "буде / не буде працювати". Середня категорія, як правило, становить переважну більшість, близько 80%. Я думаю, що таке ж співвідношення буде і для будь-якої випадкової групи.

У більшості курсів третя група, як правило, ігнорується, оскільки денюжки капає з них і не викликає занепокоєння. Вони якось працюють з другим, якщо втомленому вчителю після основної роботи часто вдається розважити аудиторію. Перший ... і що є першим ... благодать, просто кидає знання дров у піч розуму.

Ми поставили собі за мету ефективність курсу. Щоб сертифікат на курси був знаком якості як для випускника, так і для роботодавця. Але перший час швидше для випускника. Як генеральний директор я знаю ціну більшості сертифікатів та навіть дипломів про вищу освіту

Студенти думають: "Я сумував за половиною пар, але вони навчать мене, чому я заплатив гроші" або "Я не хочу робити домашнє завдання, але я заплатив гроші в цьому офісі, вони навчать мене" або "Я просто буду слухати, я нічого не буду робити, але, думаю, вони мене навчать, я заплатив ". Це відома ситуація, швидше навіть проблема. Такі студенти вважають, що хтось їм щось зобов’язаний. Але, насправді, єдине, що потрібно: читати звичайний курс і більше нічого!

Учнями не завжди звертається увага на рівень обраного курсу. Цілком можливо, що людина пішла на курс “для продвинутих”, коли для неї потрібно брати “старт”. Іноді буває, на курсах з якихось бібліотек на Java є люди, які навіть не дуже розуміють, що таке взагалі Java, і мають невиразні уявлення про програмування в цілому. Знову ж таки, може, не варто відразу лізти в нетрі. Варто сказати собі правду і почати з азів. Можливо, програмування - це не ваше?

Іноді так трапляється. Потрібно бути чесними з самими собою. Це найголовніше.

1.4. Формалізація задачі та постановка часткових задач досліджень

Рішення для багатьох. Створення, підтримка і розвиток онлайн-курсів, розрахованих на маси. На даний момент таких курсів багато, але у більшості є три основних недоліки.

1. Мова. У кращому випадку під лекціями будуть субтитри. Все інше на англійській мові, або створювати і переписувати нові матеріали для читання.

2. Унікальність. Там дійсно мало якісних курсів. Чомусь основні платформи орієнтовані на професійних викладачів і викладачів. Це досить складно зробити повний курс.

3. Проблема просування. Хоча Інтернет є глобальним, курси слід просувати в вашому місті, районі, тобто на місцевому рівні. Ходіть в школи, університети, де завгодно і знаходите зацікавлену аудиторію, залучайте офлайн-курси, просувайте свою платформу.

Виходячи з вищезазначеного, формалізуємо основну задачу: розробити програмну платформу підтримки процесів дистанційного навчання.

Але з цим виникає потреба постанови таких часткових задач як:

• Яку роль має виконувати вчитель? Відповідь – система має перейняти 80% рутинної роботи вчителя на себе. Кращий підхід: один вчитель може створювати безліч курсів і під’єднувати інших викладачів. Автоматизація процесів обліку документів та записів має стати систематичною.

• Як повинен здійснюватися навчальний процес?

Знову повернемося до самоосвіти. Нерозумно читати лекцію, під час якої студент спить, а потім навряд чи що-небудь згадає. Тому прочитка лекцій лягає на самого студента. Студент зобов’язаний самостійно проходити лекції, відповідати на запитання і навчатися. Врешті решт, це потрібно студенту, а не викладачу. Задача викладача – підтримувати, мотивувати і допомогати студенту.

• Як повинна здійснюватися перевірка знань студента?

Виключно через систематичні тести, результати яких будуть відправлятися для аналізу викладачеві. Студент матиме можливість спілкуватися з викладачем безпосередньо на будь-яку тему і отримувати якісну допомогу та поради.

Результатом досліджень має стати програмна платформа підтримки процесів дистанційного навчання.

Програмний комплекс має реалізувати наступний перелік вимог.

1. Програмний комплекс, побудований на мікро серверній архітектурі.

2. Власний захищений сервіс для зберігання та поширення даних користувача технологією OAuth2.

3. Систематизоване зберігання та доступ до навчально-методичних матеріалів навчальних дисциплін. Під матеріалами слід вважати поняття “курс”. Кожен курс, в свою чергу, поділений на лекції.

4. Автоматизований облік звітності за навчальними заняттями та надання форми для опитування студентів.

5. Реалізація розширеного пошукового механізму для курсів та лекцій.

6. Впровадження аспектно-орієнтованих компонентів, таких як: відправлення email повідомлення про початок або завершення курсу.

7. Інтерактивне створення коментарів студентами до кожної лекцій.

8. Систематизовані дії користувача в залежності від його ролі: викладач (адміністратор), студент (авторизований користувач), анонім (не авторизований користувач).

9. Асинхронна взаємодія між сервісами для поновлення інформації користувача.

10. Інтеграція з Ansible та Docker технологіями задля полегшення розгортання усієї платформи як кінцевого продукту.

11. Інтеграція з Jenkins платформою задля автоматизації розгортання платформи на віддаленому сервері.

ВИСНОВОК ДО РОЗДІЛУ 1

Перевчитися і змінити професію можна завжди. Але чи дійсно потрібно? Варто кожен раз повторювати це собі і кожного разу пам'ятати, що це дуже довгий і складний процес. І чим пізніше за це взятися, тим складніше буде. Простих перетворень не буває, тому на нову сферу можуть піти місяці, а то і роки.

Щоб стати початківцем в сфері IT, потрібно налаштувати себе на довгий процес вивчення азів і безперервне самонавчання протягом всієї кар'єри. Запастися терпінням потрібно буде ще краще, адже між початком навчання і першою роботою може пройти безліч часу.

Працюючи в IT, працюючи в офісі чи на фрілансері, можна значно покращити матеріальне благополуччя та забезпечити стабільний дохід. Але не слід забувати, що для досягнення цих цілей необхідно не тільки мати спеціальні знання та розуміти всю природу процесів, що відбуваються, але й організовуватись та постійно вчитися, вдосконалюючи свої вміння та застосовуючи нові знання на практиці.

Приватні освітні центри, кількість яких зростає, як дріжджі, намагаються навчити людей програмувати. Багато з них готові перекваліфікувати неспеціалізованих спеціалістів з нуля. Ми також спробуємо долучитися до кількості таких центрів та продовжимо створювати та покращувати власні курси для всіх.

РОЗДІЛ 2
АРХІТЕКТУРА ПРОГРАМНОГО КОМПЛЕКСУ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1. Опис технологічної платформи

Сучасний розвиток веб-додатків та технології «клієнт-сервер» призвів до багаторівневої архітектури, в якій між модулем Web-сервера та базою даних (сервер бази даних) вводиться проміжний сервер додатків (відправка повідомлень, обробка проміжних даних, збір статистика).

Веб-сервер забезпечує організацію взаємодії клієнтів і сервера баз даних. Веб-додатки передбачають інтенсивну роботу з формами, тобто із введенням та виведенням даних.

Основна ідея концепції веб-архітектури, що відрізняє її від більш традиційних архітектур сервера - створення сервера у вигляді набору функціонуючих REST-сервісів, що не економлять державу.

Для магістерського дипломного проекту, мова *Java* обрана в якості виконавчої мови програмування. Парадигма програмування: *Об'єктно-орієнтоване програмування*.

2.1.1. Архітектура MVC шаблону

MVC не потрібно називати шаблоном всього проекту, це в основному структурний шаблон, який описує, як побудувати структуру програми, зону відповідальності і взаємодію кожної з частин в цій структурі. Дика популярність цієї структури в веб-додатках розвивається завдяки простоті використання.

Архітектура MVC дозволяє розділити програмний код на 3 частини: модель, уявлення(далі – View) і контролер. Поділ на частини дозволяє спростити велику кількість коду. Якщо код написаний одним довгим скриптом, його стає важко зрозуміти і важко вносити зміни без помилок.

MVC не прив'язаний до якогось конкретного мови програмування і парадигм програмування, здебільшого це універсальний підхід. Поділ на частини тут не означає, що в коді повинно бути рівно 3 файлу (або 3 папки з файлами, або 3 класу) з назвами моделі, виду і контролера. MVC нічого не говорить нам про те, як організувати файли коду.

На практиці модель часто займає основний обсяг програми та представлена у вигляді великої кількості різних класів - сутностей, сервісів, класів роботи з базою даних, і для кожного типу класів роблять окремі папки.

Модель містить всю логіку додатка, вона зберігає і обробляє дані без безпосередньої взаємодії з користувачем (доступ до моделі можливі лише з коду, що викликає його функції). Наприклад, збереження інформації в базі даних, перевірка правильності даних, введених в форму - це завдання моделі, але отримання цих даних від користувача відбувається через рівень контролера.

Модель жодним чином не повинна залежати і не повинна нічого знати про контролери і view. Модель не є окремим класом або набором класів одного типу. Це основна частина програми, яка може містити безліч різних класів: сервіси, класи для взаємодії з базою даних, сутності, валідатори.

View відображає дані, передані йому. У веб-додатку воно зазвичай складається з шаблонів HTML-сторінок. Подання надає різні способи представлення даних, отриманих з моделі. Це може бути шаблон, який заповнюється даними. Може бути кілька різних видів, і контролер вибирає, який з них найкраще підходить для поточної ситуації.

Контролер відповідає за виконання запитів, отриманих від користувача. У веб-додатку контролер зазвичай аналізує параметри HTTP-запиту (або HTTPS) в основному з методів POST / GET, звертається до моделі для отримання або зміни деяких даних і, нарешті, викликає уявлення для відображення результату запиту. Кількість контролерів визначається кількістю розділів або сторінок сайту.

Один контролер може працювати з декількома моделями і навпаки, одна модель може використовуватися в декількох контролерах. У веб-додатку контролери зазвичай представляють собою набір класів одного типу, кожен розділ сайту має свій власний клас і створює методи для окремих сторінок (наприклад: для розділу новин - клас NewsController, це методи getLatestNews - витягти сторінку останніх новин, getNewById - одна сторінка перегляду новостей). Весь функціонал додатка міститься в моделі.

Контролер і уявлення дозволяють тільки користувачеві взаємодіяти з моделлю і відображати дані з неї. Наприклад, якщо ми створимо рекламний сайт з такими функціями, як «додати рекламу», «видалити рекламу», «знайти рекламу за критеріями», то для кожної дії десь в моделі повинна бути функція, яку можна викликати.

Які переваги? Основна перевага, яку ми отримуємо від використання концепції MVC - це чіткий поділ логіки подання (призначений для користувача інтерфейс) і логіки програми. Завдяки цій концепції стає простіше реалізувати підтримку для різних типів користувачів, які використовують різні типи пристроїв.

Реалізація повинна відрізнятися, якщо запит надходить з персонального комп'ютера або мобільного телефону. Модель повертає ті ж дані, єдина відмінність полягає в тому, що контролер вибирає різні типи для виведення даних. На додаток до ізоляції видів від логіки додатки, концепція MVC значно зменшує складність великих додатків. Код набагато більш структурований, що полегшує підтримку, тестування і повторне використання рішень.

Оскільки MVC не має суворої реалізації, він може бути реалізований різними способами. Не існує загальноприйнятого визначення того, де повинна розташовуватися бізнес-логіка, але логіка часто розташовується на рівні моделі. Ця модель буде містити всі бізнес-об'єкти з усіма даними і функціями. Деякі рамки жорстко визначають, де повинна розташовуватися бізнес-логіка, інші не мають таких правил.

Припустимо, користувач заходить на сторінку форуму. Його браузер відправляє HTTP запит на отримання сторінки зі списком повідомлень. При цьому запускається контролер, який аналізує запит користувача і запитує у моделі список повідомлень. Отримавши його, він викликає представлення і передає йому список, який в свою чергу відображає його у вигляді веб-сторінки. Після цього скрипт завершується. Якщо користувач захоче додати повідомлення, він заповнить форму, відправить її, ініціалізується контролер, який відповідає за обробку даних цієї форми, прийме дані, попросить модель перевірити і вставити в базу даних нове повідомлення, і потім віддасть HTTP відповідь з перенаправленням на сторінку перегляду повідомлень. Графічно це можна зобразити наступним чином:

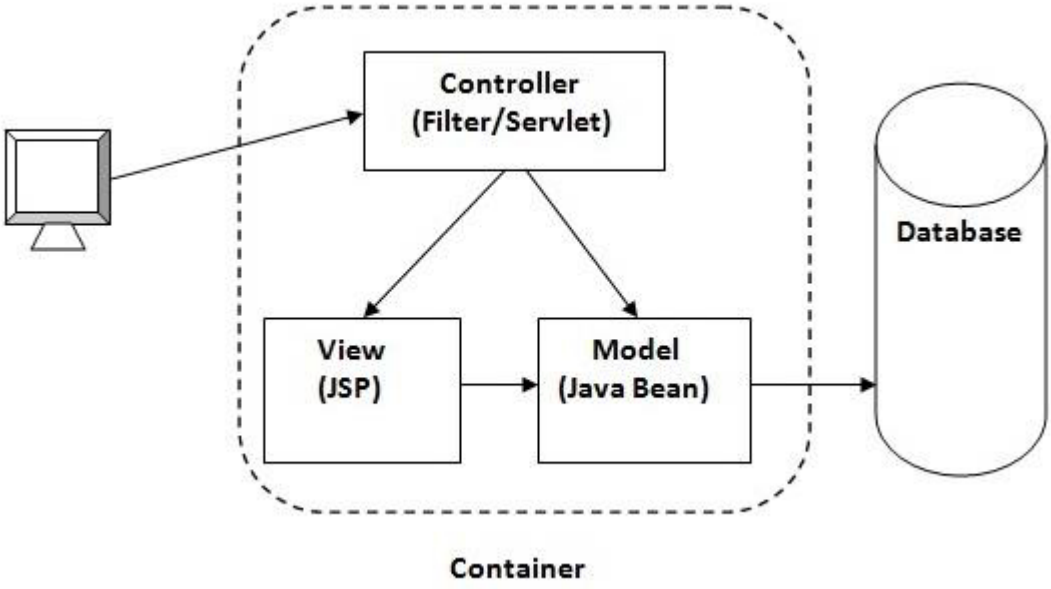

Рис. 2.1.1. Опис MVC шаблону.

2.1.2. Компоненти OAuth2 сервісу

Для початку, розберемося що собою являє протокол OAuth2. OAuth 2.0 - протокол авторизації, що дозволяє видати одному сервісу (з додатком) права на доступ до ресурсів користувача на іншому сервісі. Протокол позбавляє від необхідності довіряти додатком логін і пароль, а також дозволяє видавати обмежений набір прав, а не все відразу.

Як і перша версія, OAuth 2.0 заснований на використанні базових веб-технологій: HTTP-запити, редірект і т. П. Тому використання OAuth можливо на будь-якій платформі з доступом до інтернету і браузеру: на сайтах, в мобільних і desktop-додатках, плагинах для браузерів.

Ключова відмінність від OAuth 1.0 - простота. У новій версії немає громіздких схем підписи, скорочено кількість запитів, необхідних для авторизації.

Загальна схема роботи програми, що використовує OAuth, така:

- отримання авторизації
- звернення до захищених ресурсів

Результатом авторизації є access token - певний ключ (зазвичай просто набір символів), пред'явлення якого є пропуском до захищених ресурсів. Звернення до них в найпростішому випадку відбувається по HTTPS із зазначенням в заголовках або в якості одного з параметрів отриманого access token'a.

У протоколі описано кілька варіантів авторизації, що підходять для різних ситуацій:

- авторизація для додатків, що мають серверну частину (найчастіше, це сайти і веб-додатки);
- авторизація для повністю клієнтських додатків (мобільні і desktop-додатки);
- авторизація за логіном і паролем;
- відновлення попередньої авторизації.

Візьмемо звичайну ситуацію: при відкритті будь-якого сайту – потрібно обрати, який провайдер даних і який конкретно аккаунт використовувати, даєте дозвіл на використання даних (якщо необхідно) і працюєте далі від імені

власника цього облікового запису. Не згадуйте пароль, не витрачаєте час на тяжку реєстрацію - все це зводиться до кількох кліками.

Звісно, для цього у користувача повинен бути обліковий запис у обраного провайдера.

Термін «провайдер» має на увазі сервіс, який володіє даними користувача і, з дозволу користувача, надає стороннім сервісам (клієнтам) безпечний доступ до цих даних. Додаток, яка бажає отримати дані користувача - це клієнт.

Для прикладу взята форма логіна / реєстрації на Aliexpress. В даному випадку на формі присутній вибір з кількох провайдерів - Google, Facebook і Twitter.

Якщо натиснути на «увійти через Facebook» або «увійти через Google», сайт перенаправить на сторінку вибору аккаунта, де потрібно обрати свій обліковий запис і все – логін пройшов успішно.

Специфікація OAuth 2.0 визначає протокол делегування, який надає клієнтам безпечний доступ до ресурсів користувача на сервісі-провайдера. Такий підхід позбавляє користувача від необхідності вводити пароль за межами сервісу-провайдера: весь процес зводиться до натискання кнопки «Згоден надати доступ до ...». Ідея в тому, що маючи один добре захищений аккаунт, користувач може використовувати його для аутентифікації на інших сервісах, не розкриваючи при цьому свого пароля.

На відміну від OpenID, OAuth 2.0 може використовуватися і для авторизації. Тобто, дозволяє видати права на дії, які сам сервіс-клієнт зможе виробляти від імені власника аккаунта. При цьому сам власник після авторизації може взагалі не брати участь в процесі виконання дій. Нижче представлена стандартна схема роботи протоколу.

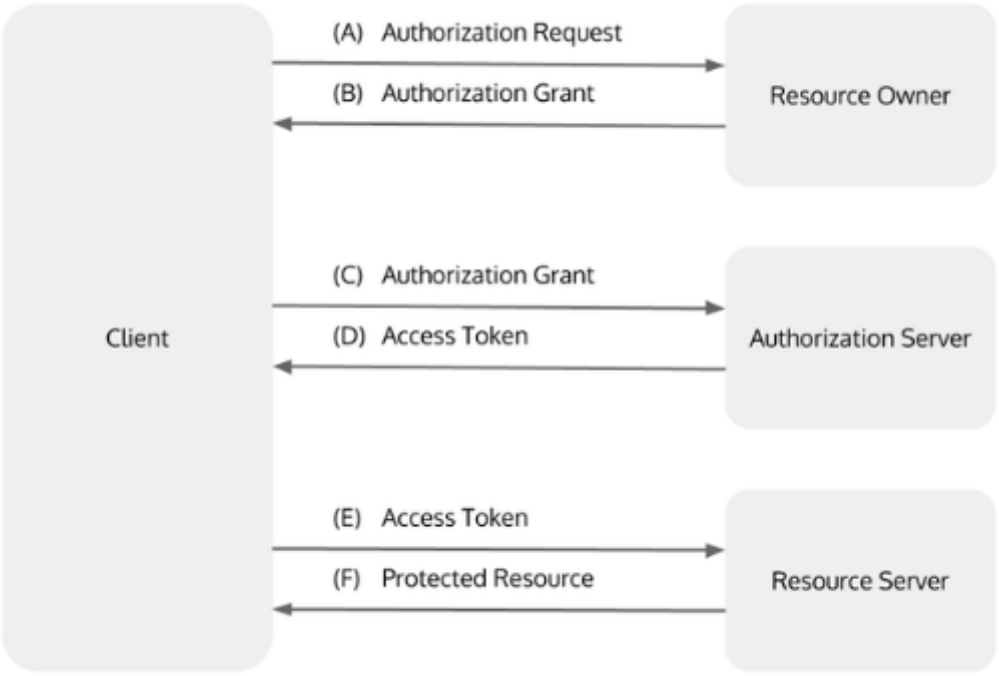

Рис. 2.1.2. Схема OAuth2 протоколу.

Що собою представляє refresh токен? Уявімо ситуацію, коли хтось заволодів Вашим access токеном і отримав доступ до захищених даних. Саме тому у токенов є термін придатності. У access токена він зазвичай маленький - від декількох секунд, до декількох днів, у refresh токена - побільше. Доступ до даних у зловмисника буде до тих пір, поки токен доступу немає протухне.

Припустимо, цей хтось заволодів ще й refresh токеном і скористався ним раніше ніж сервіс, тим самим отримавши новий access токен. В такому випадку сервіс не може отримати дані з провайдера, але для вирішення цієї проблеми буде досить вийти зі свого профіля і залогінитися заново - і все! Всі старі токени стануть недійсними або віддалятися (залежить від реалізації). Після цієї процедури сервіс отримуєте нові access / refresh токени і можете спокійно продовжити роботу, а поганий хлопець, зі старими токенами, залишився з носом.

Плюси та мінуси підходу, використовуючи OAuth2:

+ Звернення до ресурсів відбувається по HTTP / HTTPS;
+ Можливість авторизації користувача;
+ Популярність - більшість компаній використовують його в своїх API;
+ Простота реалізації і велика кількість "літератури";
+ Наявність готових рішень, які можна змінювати під свої потреби.
- Немає єдиного встановленого формату, внаслідок чого на кожен сервіс потрібно мати окрему реалізацію;
- При аутентифікації іноді доводиться робити додаткові запити для отримання навіть мінімальної інформації про користувача. Вирішується використанням jwt токена, але далеко не всі сервіси його підтримують;
- При крадіжці токена у зловмисника на якийсь час з'являється доступ до захищених даних. Для мінімізації даного варіанту можна використовують токен з підписом.

Деталі реалізації компонентів з урахуванням OAuth2 протоколу. За сервіс надання токену буде відповідати окремий новий мікросервіс на Java мові з урахуванням Spring Boot архітектури. Мікросервіс буде налагоджений на генерацію спеціального токену. Також буде реалізована доповнення стандартного виду токену з урахуванням додаткової інформації користувача. Задля безпеки даних користувача, усі контактні дані, паролі та ролі будуть зберігатися в базі, яка розташована як сховище для OAuth2 мікросервісу.

2.1.3. Взаємодія мікро сервісів та інженерія вимог

Мікросервісні архітектури розвивалися як рішення проблем масштабованості та інновацій з монолітними архітектурою. Є ряд визначень, пропонованих для мікросервісов.

- *Невеликі автономні сервіси, які працюють спільно* - Сем Ньюман (Sam Newman)
- *Розробка окремого додатка у вигляді набору невеликих сервісів, кожен з яких працює в своєму власному процесі і взаємодіє з полегшеними*

> *механізмами, часто API-інтерфейсом HTTP-ресурсів. Ці сервіси побудовані на бізнес-можливості і можуть бути розгорнуті незалежно за допомогою повністю автоматизованого механізму розгортання. Існує мінімальний рівень централізованого управління цими службами, які можуть бути написані на різних мовах програмування і використовувати різні технології зберігання даних* - Джеймс Льюїс (James Lewis) і Мартін Фаулер (Martin Fowler)

Для мікросервісов немає єдиного прийнятого визначення, є кілька важливих характеристик:

- REST - побудований навколо RESTful ресурсів. Зв'язок між сервісами може бути на основі подій або протоколу HTTP;
- Невеликі добре обрані розгортаються блоки - обмежений контекст;
- Хмарні можливості - динамічне масштабування.

Хоча розробка кількох невеликих компонентів може здатися легкою, існує ряд властивих їй складнощів, пов'язаних з архітектурою мікросервісов.

Найчастіші проблеми:

1) Потрібно швидке налаштування.

Не потрібно витрачати місяць на налаштування кожного мікросервіса. Потрібно бути в змозі швидко створювати мікросервіси.

2) Автоматизація.

Оскільки замість моноліту є ряд дрібніших компонентів, необхідно автоматизувати все - збірки, розгортання, моніторинг і т. Д

3) Видимість.

Тепер є кілька невеликих компонентів для розгортання та обслуговування. Може бути, 100 або 1000 компонентів. Повинна бути змога відслідковувати і

визначати проблеми автоматично. Також потрібна відмінна видимість навколо всіх компонентів.

4) Обмежений контекст.

Визначення меж мікросервіса - непросте завдання. Обмежений контекст з доменного дизайну - хороша відправна точка. Чітке розуміння домену розвивається протягом певного періоду часу. Потрібно переконатися, що кордони мікросервіса розвиваються.

5) Управління конфігураціями.

Необхідно підтримувати конфігурації для сотень компонентів в різних середовищах. Потрібно рішення для управління конфігурацією

6) Динамічне збільшення і зменшення.

Переваги мікросервісов будуть реалізовані тільки в тому випадку, якщо програми можуть легко масштабироваться в хмарі.

7) Колода карт.

Якщо мікросервіс в нижній частині ланцюжка викликів виходить з ладу, він може вплинути на всі інші мікросервіси. Мікросервіси повинні бути відмовостійкими.

8) Налагодження.

Коли виникає проблема, що вимагає вивчення, може знадобитися вивчити кілька служб в різних компонентах. Централізоване ведення журналу та інструментальні панелі необхідні для полегшення налагодження проблем.

9) Узгодженість.

Не має бути широкого спектра інструментів, які вирішують одну і ту ж проблему. Хоча важливо стимулювати інновації, важливо також мати децентралізоване управління мовами, платформами, технологіями та інструментами, використовуваними для реалізації / розгортання / моніторингу мікросервісів.

Схема стандартної взаємодії мікросервісів:

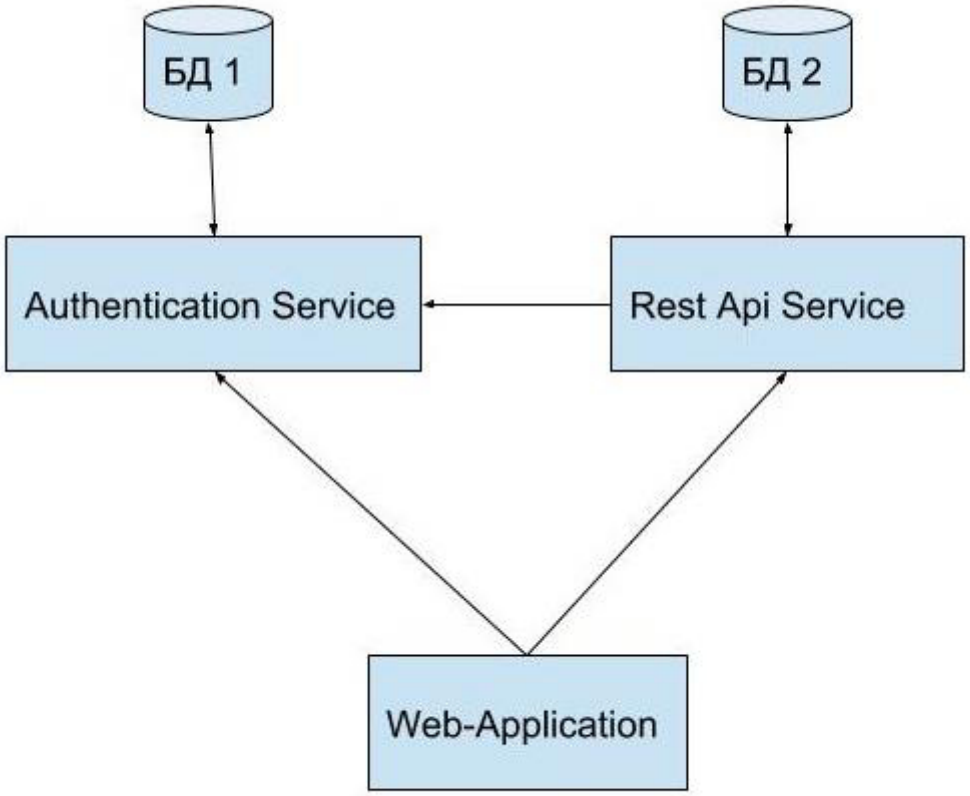

Рис. 2.1.3. Схема взаємодії мікросервісів.

2.1.4. Spring Framework та його компоненти

Як зазначалося раніше, головним виступаючим фреймворком для роботи є *Spring Framework*. За допомогою нього, у Java з'являється можливість будувати додатки швидко, легко і ефективно. Вони позбавлять програміста від необхідності писати код з нуля. Також вони дають можливість використовувати вже написаний код повторно.

Spring - одна з найпопулярніших платформ Java. Більшість розробників Java вибирають це. Однією з причин такої любові є можливість введення залежності. Впровадження залежностей (DI) - це процес надання зовнішньої залежності програмного компоненту. Це особлива форма інверсії контролю (IoC) стосовно до управління залежностями. Робота каркаса реалізації залежностей описується наступним чином. Додаток, незалежно від дизайну, запускається всередині контейнера IoC, що надається платформою.

Деякі об'єкти в програмі як і раніше створюються звичайним способом мови програмування, деякі створюються контейнером на основі наданої йому конфігурації. Умовно, якщо об'єкту потрібен доступ до певної службі, він приймає на себе відповідальність за доступ до цієї служби: він або отримує пряме посилання на місце розташування служби, або посилається на відомий «локатор служби» і запитує посилання для реалізації певний тип послуг.

Використовуючи введення залежностей, об'єкт просто надає властивість, яке може зберігати посилання на потрібний тип сервісу; і коли об'єкт створюється, посилання на реалізацію бажаного типу послуги автоматично вставляється в це властивість (поле), використовуючи кошти середовища.

Розгортання залежностей є більш гнучким, оскільки стає простіше створювати альтернативні реалізації цього типу сервісу, а потім вказувати, яку реалізацію слід використовувати, наприклад, у файлі конфігурації, без змін в об'єктах, які використовують цей сервіс. Це особливо корисно при модульному тестуванні, оскільки вставити реалізацію заглушки служби в тестований об'єкт дуже просто.

Тому Spring є ідеальним середовищем для розробки корпоративних додатків і моделей конфігурації на Java. Це допомагає розробникам зосередитися на бізнес-логіці програми. В даний час термін «весна» часто відноситься до цілого сімейства проектів. В рамках магістерської дипломної роботи задіяні такі компоненти, які наведені нижче.

Перший на черзі - **Spring MVC.** Він забезпечує архітектуру паттерна Model - View – Controller.

Нижче наведена послідовність подій, яка відбувається при HTTP-запиті:

1. Після отримання запиту з параметрами, DispatcherServlet звертається до інтерфейсу HandlerMapping, який визначає на який конкретно контролер повинен піти запит.

2. Контролер приймає запит, шукає потрібний метод і починається бізнес процес.
3. Викликаний метод визначає дані моделі, засновані певною бізнес-логікою і повертає в DispatcherServlet ім'я представлення (View).
4. За допомогою інтерфейсу ViewResolver, DispatcherServlet визначає, яке представлення потрібно використовувати на підставі отриманого імені.
5. Після того, як представлення створено, DispatcherServlet відправляє дані моделі у вигляді атрибутів в представлення, яке в кінцевому підсумку відображається в браузері.

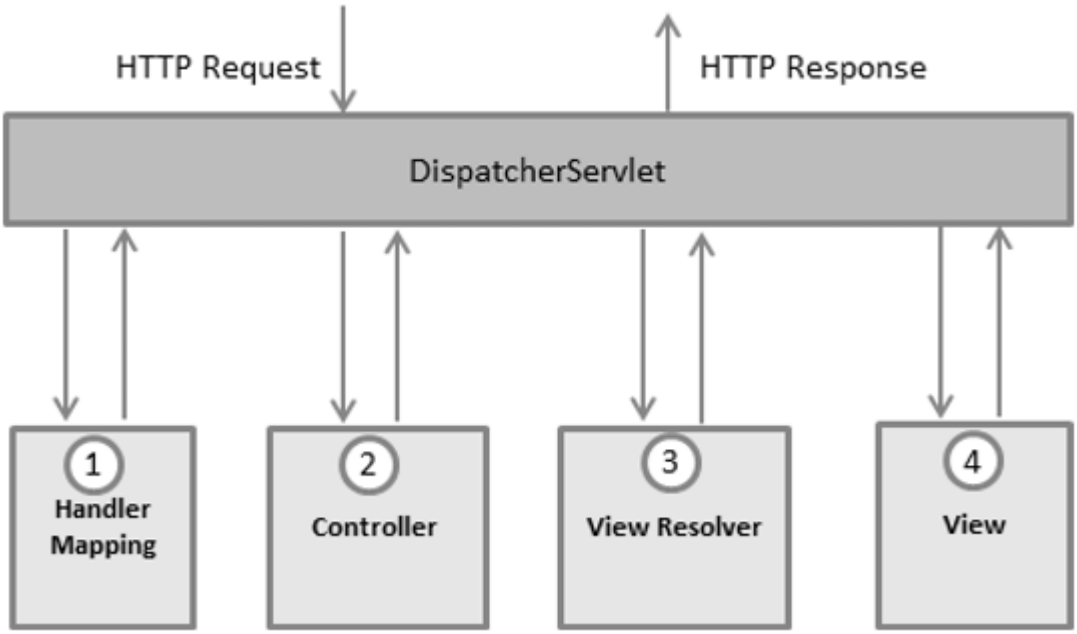

Рис. 2.1.4. Принцип роботи Spring MVC.

Наступна важлива ступінь в побудові сервісу – взаємодія з БД за допомогою **Spring Data**. Мета абстракції сховища Spring Data - значно скоротити обсяг коду, необхідного для реалізації рівнів доступу до даних для різних репозиторіїв. Структура полегшує використання технологій доступу до даних, реляційних і нереляційних баз даних, хмарних баз даних. Spring DATA - це базовий проект, який включає в себе безліч інших підпроектів, які працюють з конкретними базами даних.

Завдяки реалізації одного з інтерфейсів сховища в DAO, деякі базові методи CRUD (і запити) вже будуть визначені і реалізовані.

Щоб визначити більш конкретні методи доступу, Spring JPA підтримує досить багато опцій:

• просто визначити новий метод в інтерфейсі;

• надати актуальний JPQL-запит, використовуючи анотацію @Query;

Переваги цього API будуть більш помітні при роботі з великою кількістю фіксованих запитів, які потенційно можуть бути виражені більш коротко через меншої кількості повторно використовуваних блоків, які продовжують з'являтися в різних комбінаціях.

Реальна реалізація DAO з Spring Data дійсно прихована, так як ми не працюємо з нею безпосередньо. Однак це досить проста реалізація - SimpleJpaRepository - яка визначає семантику транзакції з використанням анотацій.

Чіткіше, анотація @Transactional доступна тільки для читання на рівні класу, яка потім використовується для методів, які будуть недоступні тільки для читання. Решта семантика транзакції використовується за умовчанням, але її можна легко перевизначити вручну для кожного методу.

Слід також зазначити, що існує додаткова функціональність, а саме Data JDBC. Ідея JDBC Spring Data полягає в наданні доступу до реляційних баз даних без використання всієї складності JPA.

JPA пропонує такі функції, як відкладена завантаження, кешування і брудне відстеження. Хоча ці функції дуже корисні, якщо вони, звичайно, дійсно необхідні, вони можуть дуже ускладнити розуміння логіки доступу до даних.

Spring Data JDBC фокусується на набагато більш простої моделі. Чи не буде кешування, відстеження змін або відкладених завантажень. Замість цього SQL-запити будуть виконуватися тоді і тільки тоді, коли викликали метод сховища. Повернений результат буде повністю завантажений в пам'ять після

виконання методу. Не буде ніякого «сеансового» механізму або проксі-об'єктів для сутностей. І все це повинно зробити Spring Data JDBC простішим і зрозумілим інструментом для доступу до даних.

Синхронізація OAuth2 сервісу та **Spring Security**. Це все, що потрібно для забезпечення якісної безпеки даних користувача.

Ключови об'єкт контексту Spring Security – це SecurityContextHolder. Він містить інформацію про поточний контексті безпеки програми, яка включає в себе детальну інформацію про користувача, який працює з додатком в поточний час. За замовчуванням SecurityContextHolder використовує ThreadLocal для зберігання такої інформації, що означає, що контекст безпеки завжди доступний для методів, які працюють в одному потоці. Щоб змінити стратегію зберігання цієї інформації, також можна використовувати статичний метод цього класу setStrategyName (). Наступний важливий об’єкт в безпеці - SecurityContext, який містить об'єкт Authentication і в разі необхідності інформацію системи безпеки, пов'язану із запитом від користувача.

Наступний важливий компонент – **Spring AOP**. В основі аспектно-орієнтованого програмування лежить поняття crosscutting concerns, яке не має поки усталеного еквівалента в українській мові. Найбільш близьким за змістом вважається і найчастіше використовується словосполучення «наскрізна функціональність», якому і буде слідувати дана стаття в подальшому.

В основі аспектно-орієнтованого програмування лежить поняття crosscutting concerns, яке не має поки усталеного еквівалента в російській мові. Найбільш близьким за змістом вважається і найчастіше використовується словосполучення «наскрізна функціональність», якому і буде слідувати дана стаття в подальшому.

Під наскрізний функціональністю розуміється функціональність, реалізувати яку в окремому компоненті мови програмування традиційними

засобами процедурного або об'єктно-орієнтованого програмування або дуже складно, або взагалі неможливо, оскільки ця функціональність необхідна в більшій частині модулів системи. Крім того, ця функціональність не мають прямого відношення до предметної області.

Прикладом такої функціональності є протоколювання роботи системи (logging). Якщо треба реєструвати час початку і закінчення виконання методів деяких класів не використовуючи аспектно-орієнтованого програмування, то в вихідному коді кожного методу 2 рази явно використовується функція запису в журнал.

Уявивши прикладну систему навіть середньої складності, можна легко зрозуміти ту величезну кількість необхідної роботи, яку необхідно зробити, в разі зміни інтерфейсу протоколювання. Використання аспектно-орієнтованого програмування допомагає дотримуватися принципу поділу відповідальності (separation of concerns), що позитивно позначається на багатьох характеристиках розробляється інформаційної системи, деякими з яких є наступні:

* Більш ефективний процес створення системи
* Більш якісна реалізація
* Підвищені можливості тестування
* Покращена модульність системи

Останній в списку, але не по важливості компонент, який використовується – **Spring Boot**.

Spring Boot - це покоління інструментів, що спрощують процес налаштування додатків Spring. Це не засіб автоматичної генерації коду, а плагін для системи автоматизації збору проектів (підтримує Maven і Gradle).

Плагін надає можливості для тестування і розгортання додатків Spring. Команда mvn spring-boot: run дозволяє запускати вашу програму на порте

8080. Крім того, Spring Boot дозволяє упакувати додаток в окремий файл jar з повноцінним контейнером Tomcat.

Основною перевагою Spring Boot є конфігурація ресурсів на основі вмісту шляху до класів. Наприклад, якщо файл pom.xml проекту Maven містить JPA-залежності і драйвер PostgreSQL, Spring Boot налаштує модуль персистентності для PostgreSQL. Якщо додавати веб-залежність, можливо отримати налаштований Spring MVC за замовчуванням. Якщо потрібна узгодженість і більше нічого, Spring Boot налаштує Hibernate в якості постачальника JPA з базою даних HSQLDB. Якщо створювати веб-додаток, але не вказувати нічого іншого, Spring Boot налаштує засіб дозволу для системи шаблонів Thymeleaf.

Говорячи про конфігурацію за замовчуванням, Spring Boot досить інтуїтивний в цьому відношенні. Програміст можете не завжди погоджуватися з вибором налаштувань, але це забезпечить працює модуль. Це дуже корисний підхід, особливо для початківців розробників, які можуть почати з налаштувань за замовчуванням, а потім, вивчаючи існуючі альтернативи, вносити зміни в конфігурацію.

В результаті Spring надає основу для майбутньої програми, якщо хочете «порожню» для вашої майбутньої програми. В цьому випадку фреймворк диктує правила побудови програми - існує певна архітектура додатки, в якій необхідно побудувати свій функціонал. Ця функціональність фактично стане бізнес-логікою вашого застосування. Функцію створення об’єктів можна делегували контейнеру Spring. Це і є інверсія контролю (IoC) - передача функції встановлення необхідних залежностей (об'єктів) контейнеру.

Spring звільняє не тільки від необхідності створювати об'єкти, але і зв'язувати їх. Наприклад анотація @Autowired дозволяє автоматично пов'язувати компоненти. Автоматичне зв'язування дозволяє зменшити кількість коду при визначенні залежностей компонентів.

І остання важлива особливість - Spring налаштування компонентів відділені від програмного коду. Винесення конфігурації (управління залежностями) в окремий файл полегшує подальші зміни в проекті.

2.2. Опис збереження даних

Кожне сучасне обладнання чи веб-додаток використовує одну або кілька баз даних для зберігання інформації. Бази даних надають інструменти для організації, додавання, пошуку, оновлення, видалення та обчислення даних. У більшості випадків сервери веб-додатків взаємодіють безпосередньо з серверами завдань.

Крім того, кожна серверна служба може мати відповідну базу даних, ізольовану від іншої частини програми. Тут варто згадати SQL і NoSQL. SQL означає «Мова структурованих запитів». Він був винайдений в 1970-х роках для створення стандартного способу запиту реляційних наборів даних, доступних для широкої аудиторії.

Бази даних SQL зберігають дані в таблицях, які пов'язані загальними ключами. Такі ключі зазвичай представлені цілими числами. Кожна система управління базами даних (далі СУБД) реалізує одну з моделей бази даних для логічного структурування використовуваних даних. Ці моделі є основним критерієм того, як додаток буде працювати і управляти інформацією. Існує кілька таких моделей, серед яких найбільш популярною є реляційна. Хоча він дуже потужний і гнучкий, є ситуації, які він не може запропонувати.

Тут на допомогу прийде відносно нова модель під назвою NoSQL. Він набирає популярність і пропонує дуже цікаві рішення і додатковий функціонал. Оскільки ці системи не використовують суворе структурування даних, вони пропонують більше можливостей для обробки інформації.

Хоча все системи управління базами даних виконують одну і ту ж основну задачу (тобто дозволяють користувачам створювати, редагувати і отримувати доступ до інформації, що зберігається в базах даних), процес виконання цього завдання сильно розрізняється. Крім того, функції і можливості кожної бази даних можуть мати відчутні відмінності. Різні бази даних документуються по-різному: більш-менш ретельно. Технічна підтримка також надається по-різному.

При порівнянні різних популярних баз даних слід враховувати, чи зручна ця конкретна база даних, і переконатися, що вона добре інтегрується з іншими вже використовуваними продуктами. Крім того, вартість системи і підтримка, що надається розробником, повинні бути прийняті до уваги при виборі.

Коли справа доходить до вибору бази даних для компанії, потрібно враховувати здатність бази даних «рости» з розвитком організації. Малим підприємствам можуть знадобитися тільки базові функції і можливості, а також невеликий обсяг інформації, розміщеної в базі даних. Але вимоги можуть значно зрости з часом, і перехід на іншу базу даних може стати проблемою. Існує кілька популярних СУБД, як платних, так і безкоштовних, які можна рекомендувати для використання в організації.

Виконуючим сервером БД для магістерського дипломного проекту обрано *PostgreSQL*.

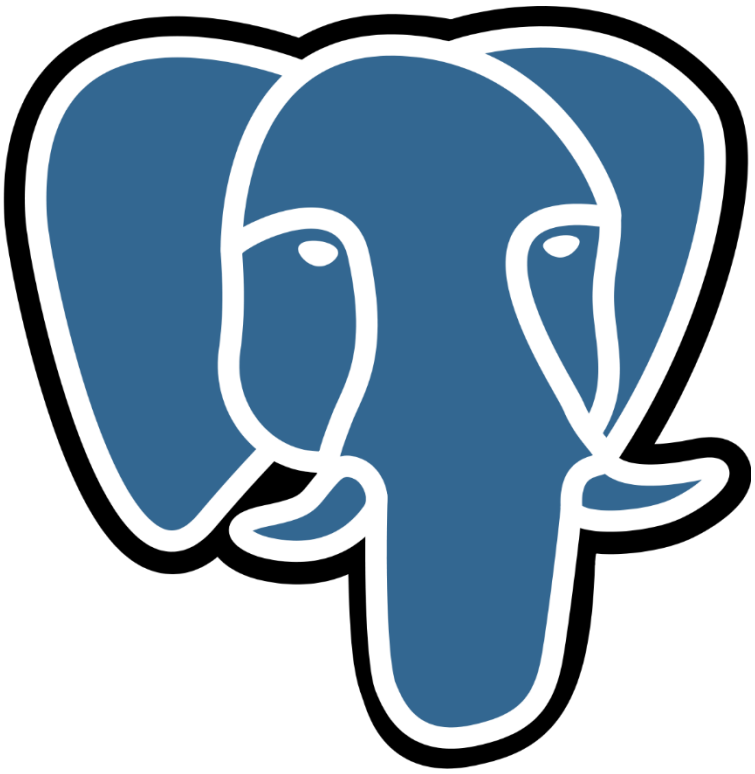

Рис. 2.2. Логотип PostgreSQL.

На розробку структури БД були виділені такі цілі, як: захищеність даних користувача в OAuth2 мікросервері, швидкодоступність даних в бізнес-мікросервісі.

Існує великий список типів даних, які підтримує Постгрес. Крім числових, з плаваючою точкою, текстових, булевих і інших очікуваних типів даних (а також безлічі їх варіацій), PostgreSQL може похвалитися підтримкою uuid, грошового, що перераховується, геометричного, бінарного типів, мережевих адрес, бітових рядків, текстового пошуку, xml, json , масивів, композитних типів і діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів.

Підтримка JSON в PostgreSQL дозволяє перейти до зберігання schema-less даних в SQL базі даних. Це може бути корисно, коли структура даних вимагає певної гнучкості: наприклад, якщо в процесі розробки структура все ще змінюється або невідомо, які поля буде містити об'єкт даних.

Тип даних JSON забезпечує перевірку коректності JSON, який дозволяє використовувати спеціалізовані JSON оператори і функції, вбудовані в Постгрес для виконання запитів і маніпулювання даними. Також доступний тип JSONB - двійкова різновид формату JSON, у якій прогалини видаляються, сортування об'єктів не зберігається, натомість вони зберігаються найбільш оптимальним чином, і зберігається тільки останнє значення для ключів-дублікатів. JSONB зазвичай є кращим форматом, оскільки вимагає менше місця для об'єктів, може бути проіндексовані і обробляється швидше, так як не вимагає повторного синтаксичного аналізу.

У Постгреса безліч можливостей. Створений з використанням об'єктно-реляційної моделі, він підтримує складні структури і широкий спектр вбудованих і обумовлених користувачем типів даних. Він забезпечує розширену ємність даних і заслужив довіру дбайливим ставленням до цілісності даних.

2.3. Опис компонентів побудови та розгортання кінцевої платформи

Щоб коректно розуміти контекст, визначимо для початку, що означає CI та CD.

1. Постійна інтеграція (Continuous integration)

Розробники, які практикують постійну інтеграцію, об'єднують свої зміни назад в основну гілку розробки якомога частіше. Тобто якомога частіше роблять так звані «мерж ріквести». Зміни розробника підтверджуються шляхом створення збірки та запуску автоматизованих тестів проти збірки. Тим самим програміст уникає пекла інтеграції, яка зазвичай трапляється, коли люди чекають дня випуску, щоб об'єднати свої зміни у гілку випуску. Це означає, що не потрібно боятися за свої зміни у разі випуску до клієнта. Постійна інтеграція дозволяє виявляти помилки в коді на самому раньому етапі.

Постійна інтеграція робить великий акцент на автоматизації тестування, щоб перевірити, чи програма не порушена, коли нові об'єкти інтегруються в основну гілку.

2. Постійна доставка (Continuous delivery)

Безперервна доставка - це розширення постійної інтеграції, щоб переконатися, що можливо швидко та стабільно опублікувати нові зміни для своїх клієнтів. Це означає, що, крім того, що автоматизували тестування, також автоматизовано процес випуску, і можливо розгорнути свою програму в будь-який момент, натиснувши кнопку. Тим самим не потрібно очікувати ніяких складнощів у цьому процесі.

Теоретично, при безперервній доставці потрібно вирішити: випуск щодня, щотижня, щотижня або як завгодно, що відповідає бізнес-вимогам. Однак якщо бізнес справді хоче отримати переваги безперервної доставки, слід якомога раніше розгорнутися до виробництва, щоб випустити невеликі партії, які легко усунути в разі проблеми.

3. Постійне розгортання (Continuous deployment)

Постійне розгортання йде на крок далі, ніж безперервна доставка. При такій практиці кожна зміна, яка проходить усі етапи вашого виробничого

трубопроводу, передається відразу вашим клієнтам. Ніякого втручання людини немає, і лише невдалий тест не дозволить перетворити нову зміну на виробництво.

Безперервне розгортання - це відмінний спосіб прискорити цикл зворотного зв’язку зі своїми клієнтами та зняти тиск з боку команди, оскільки вже немає Дня випуску. Розробники можуть зосередитись на створенні програмного забезпечення, і вони бачать, як їх робота розпочинається впродовж декількох хвилин після того, як вони закінчили роботу над цим.

Схематично можна зобразити наступним чином:

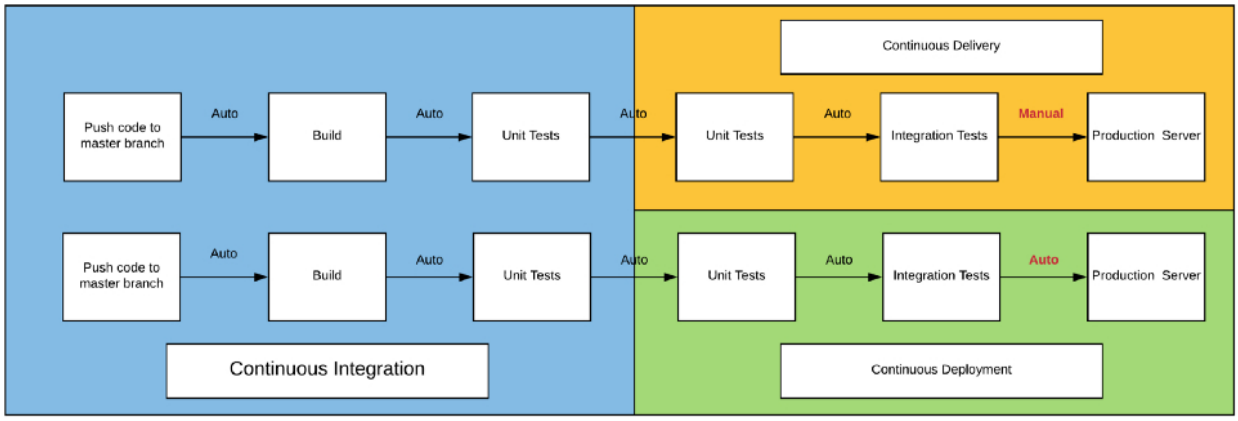

Рис. 2.3.1. Принципи розгортання сервісів.

Процес розгортання починається з сбірки. Що таке збірка? Наприклад, потрібно скопіювати модулі в одне місце і запустити компіляцію програми. Якщо все вийшло, то збірку можна вважати успішною, якщо немає - то у команди з'являється привід розібратися подетально, і вирішити проблему поки все не зайшло занадто далеко.

У інтерпретованих мовах типу PHP, Python і Ruby компілювати нічого. У них складанням може бути запуск юніт-тестів, Деплой веб-додатки на тестовий сервер, і прогін acceptance-тестів на цьому тестовому сервері.

Разом, Continuous Integration - це практика. Призначення практики - зменшити кількість інтеграційних помилок, підвищити якість виробленого софта. Спосіб - запуск збірки проекту кілька разів в день.

Допускаю що Граді Буч в стародавні часи починав практикувати CI взагалі вручну, бігаючи по відділах своєї компанії, і змушуючи всіх давати йому дискети з останніми версіями модулів, а потім з мовою напереваги все це вручну компілював. На 2020 рік все трохи простіше - процес CI автоматизований в купі систем.

Рис. 2.3.2. Приклади існуючих сервісів для CI.

Навіщо це все звичайному розробнику?

Чим більше аспектів розробки здатні закрити, тим більше цінність розробників. Якщо програміст подбав про тести і про деплой додатки – це дуже ціниться. Наведемо пару ситуацій з практики

Ситуація №1

Два розробника розробляють на різних гілках. У кожного з них тести проходять успішно, в тому числі і після Мержа с майстром. Але як тільки обидві гілки вмержені в майстер, тести починають падати. Проблема в тому що без CI такий код запросто може піти в продакшн, і наявність помилки з'ясується вже тільки там.

Ситуація №2

Команда з п'ятнадцяти чоловік розробляє веб-додаток. У Василя скоро сімейне свято. Він відпросився у начальства піти раніше, не надто ретельно перевіривши працездатність коду, запустив в командному рядку Деплой і пішов з роботи. Комп’ютер він заблокував перед виходом і заодно вимкнув телефон - сім'я ж важливіше роботи!

В результаті продакшн поламаний, логи деплоя залишилися на заблокованому комп’ютері, розробникам в офісі лишається рвати на собі волосся в спробах зрозуміти що ж було задеплоено, і що пішло не так в процесі деплоя, і хто взагалі винен.

Якщо в команді використовується CI-система, то всі логи деплоя (і будь-яких інших завдань) зберігаються в ній, так що завжди можна подивитися що пішло не так.

Більшість систем налашновані на деплоймент скріпти за допомогою Ansible Framework та систему старту на базі Docker Framework. Ці дві технології застосовано в магістерській дипломній роботі.

2.3.1. Технологія Docker

Технології контейнеризації додатків знайшли широке застосування в сферах розробки ПО і аналізу даних. Ці технології допомагають зробити програми більш безпечними, полегшують їх розгортання і покращують можливості по їх масштабування. Ріст і розвиток технологій контейнеризації можна вважати одним з найважливіших трендів сучасності.

Docker - це платформа, яка призначена для розробки, розгортання і запуску додатків в контейнерах. Слово «Docker» останнім часом стало чимось на зразок синоніма слова «контейнеризація».

В основі роботи докера лежить поняття «контейнер». Як і звичайний пластиковий контейнер, контейнер Docker володіє наступними характеристиками:

- У ньому можна щось зберігати. Щось може перебувати або в контейнері, або за його межами.

- Його можна переносити. Контейнер Docker можна використовувати на локальному комп'ютері, на комп'ютері колеги, на сервері постачальника хмарних послуг (на кшталт AWS).
- У контейнер зручно щось класти і зручно щось з нього виймати.
- У звичайного контейнера є кришка на засувках, яку треба зняти для того, щоб щось покласти в контейнер або щось з нього вийняти. У контейнерів Docker є щось подібне, що представляє їх інтерфейс, тобто - механізми, що дозволяють їм взаємодіяти із зовнішнім світом. Наприклад, у контейнера є порти, які можна відкривати для того, щоб до додатка, що працює в контейнері, можна було б звертатися з браузера.
- Працювати з контейнером можна і засобами командного рядка.
  Якщо потрібен контейнер, його можна замовити в інтернет-магазині.

Звичайно, пластикові контейнери, на відміну від контейнерів Docker, ніхто не буде надсилати безкоштовно, та й коли програміст їх отримаєте, вони будуть порожніми. А ось в контейнерах Docker завжди є щось цікаве.

Контейнери Docker можна порівнювати не тільки зі звичайними контейнерами або з живими організмами. Їх можна порівняти і з програмами. Зрештою, контейнери - це програми. І, на фундаментальному рівні, контейнер являє собою набір інструкцій, який виконується на якомусь процесорі, обробляючи якісь дані.

Віртуальні машини

Попередниками контейнерів Docker були віртуальні машини. Віртуальна машина, як і контейнер, ізолює від зовнішнього середовища додаток і його залежності. Однак контейнери Docker мають переваги перед віртуальними машинами. Так, вони споживають менше ресурсів, їх дуже легко переносити, вони швидше запускаються і приходять в працездатний стан. У цьому матеріалі можна знайти докладний порівняння контейнерів і віртуальних машин.

Образ контейнера Docker

Вище ми вже говорили про «образах». Що це таке? Те, що в термінології Docker називається «чином», або, по-англійськи, «image», це зовсім не те ж саме, що, наприклад, фотографія (це - одне із значень слова «image»).

Файл Dockerfile

Файл Dockerfile містить набір інструкцій, дотримуючись яких Docker буде збирати образ контейнера. Цей файл містить опис базового образу, який буде представляти собою вихідний шар образу. Серед популярних офіційних базових образів можна відзначити python, ubuntu, alpine.

Docker Compose

Docker Compose - це інструмент, який спрощує розгортання додатків, для роботи яких потрібно кілька контейнерів Docker. Docker Compose дозволяє виконувати команди, які описуються в файлі docker-compose.yml. Ці команди можна виконувати стільки раз, скільки буде потрібно. Інтерфейс командного рядка Docker Compose спрощує взаємодію з многоконтейнернимі додатками. Цей інструмент встановлюється при установці Docker.

В образ контейнера, поверх базового образу, можна додавати додаткові шари. Робиться це відповідно до інструкцій з Dockerfile. Наприклад, якщо Dockerfile описує образ, який планується використовувати для вирішення завдань машинного навчання, то в ньому можуть бути інструкції для включення в проміжний шар такого способу бібліотек Java, Python і JS.

І, нарешті, в образі може міститися, поверх всіх інших, ще один тонкий шар, дані, що зберігаються в якому, піддаються зміні. Це - невеликий за обсягом шар, що містить програму, яку планується запускати в контейнері.

2.3.2. Технологія Ansible

Ansible - це програмне рішення для віддаленого управління конфігураціями. Воно дозволяє налаштовувати віддалені машини. Головна його відмінність від інших подібних систем в тому, що Ansible використовує існуючу інфраструктуру SSH, в той час як інші (chef, puppet, та ін.) вимагають установки спеціального PKI-оточення.

В останні пару років все частіше використовують Ansible для вирішення практично будь-яких завдань пов'язаних з автоматизацією, будь то конфігурація, резервне копіювання або деплой проектів. Сама система дуже добре документована, Ansible містить у собі плейбук, ролі, змінні, шаблони і файли необхідні для автоматизації розгортання серверів, коду і всього іншого, що можна зробити за допомогою Ansible.

Отже, Ansible це дуже гнучкий і легкий інструмент для написання сценаріїв автоматизації будь-якої складності. Ви можете описати в ньому як просте оточення розробника так складну структуру великого проекту з декількома оточеннями (dev / stage / prod).

З Ansible можна вирішувати такі завдання:

- Установка / видалення ПО;
- Конфігурація ПЗ;
- Створення / видалення користувачів;
- Контроль призначених для користувача паролів / ключів;
- Створення / видалення контейнерів / віртуальних машин;
- Деплой коду вашого ПО;
- Запуск різних скриптів / тестів.

Всі дії (tasks) які потрібно виконати можна записувати в плейбук (якщо їх не більше десятка і не використовується файли і шаблони) в інших випадках варто використовувати ролі.

У плейбук як мінімум повинні бути вказані:

- Цільова група хостів (**hosts**) тут можна задавати виключення за маскою (-hosts: app-servers:! App-04. *), Або кілька груп через двокрапку (-hosts: db-servers: cdn);
- Дії (**tasks**) або ролі (**roles**).

Додатково можна вказати:

- Користувача для ssh-connect (**remote_user**);
- Користувача sudo (**become_user**);
- Використання підвищення привілеїв (**become**: True / False);
- Змінні (**vars**)
- Файли змінних (**vars_files**)
- Кількість одночасних коннектов (**serial**), якщо виставити 1 - то буде відбуватися послідовне виконання плейбук по одному хосту за раз.

Шифрування змінних з Ansible-vault

Ansible-vault утиліта для шифрування (default AES256) файлів групових або хостових змінних і в принципі будь-яких файлів в яких ви хочете зберігати секретні змінні (паролі, ключі і т.д.). Таким чином, можна зберігати в репозиторії (навіть в публічному, хоча це все ж не бажано) будь-які дані і не переживати за їх безпеку.

Паролі має сенс зберігати в групових файлах (group_vars) на ім'я групи або як all якщо на всіх середовищах одні і ті ж користувачі.

ВИСНОВОК ДО РОЗДІЛУ 2

Виходячи з вищезазначеного матеріалу, веб-додаток, який будується в рамках даної магістерської дипломної роботи, складається з клієнтської і серверної частин, тим самим реалізуючи технологію «клієнт-сервер».

Клієнтська частина генерує запити до сервера і обробляє відповіді від нього. Серверна частина отримує запит від клієнта, виконує обчислення і відправляє відповідь клієнту через мережу, використовуючи протокол HTTP.

Цей веб-додаток може виступати в якості клієнта для інших служб, таких як база даних або OAuth2 сервісу, розташоване на іншому порті або ж навіть сервері.

Також за безпеку технічних деталей (паролей, доступів та шифр-кодів) відповідає частина Ansible, яка називається Ansible-vault. Вона допомогає безпечно користуватися паролями всередині системи, не випускаючи та не показуючи їх ніде.

Найкращим вибором серед існуючих для побудови платформи є використання середовища Spring. Spring як і раніше залишається великий і не найпростішим структурою, але це ціна абстракцій високого рівня, які він надає.

Додаткове налаштування за допомогою Docker та Ansible дають змогу впевнено та швидко розгорнути систему будь-де. І така система буде готова до роботи. Це і є найвагоміший плюс Docker та Ansible. Крім того, немає ніяких сумнівів в тому, що багато складних речей (занадто багато) можна зробити в одній анотації або залежності. На даному етапі не можливо виділити чи побачити явні недоліки фреймворків, скоріш якщо і виділяти такі – то лише через недостатність досвіду.

РОЗДІЛ 3
РОЗРОБКА ТА ПОКРАЩЕННЯ ПРОГРАМНОГО ДИСТАНЦІЙНОГО НАВЧАННЯ

3.1. Покращення захисту даних користувача

Процес збереження даних користувача, його паролі, ролі, взаємодії та інше були глобально перероблені з додаванням новітньої технології та винесенням логіки збереження даних користувача в окремий мікро сервіс. Мікро сервіс з OAuth 2 являє собою фреймворк для авторизації, що дозволяє додаткам здійснювати обмежений доступ до призначених для користувача аккаунтів на HTTP сервісах, наприклад, на Facebook, GitHub і DigitalOcean. Він працює за принципом делегування аутентифікації користувача сервісу, на якому знаходиться аккаунт користувача, дозволяючи сторонньому додатку отримувати доступ до акаунту користувача.

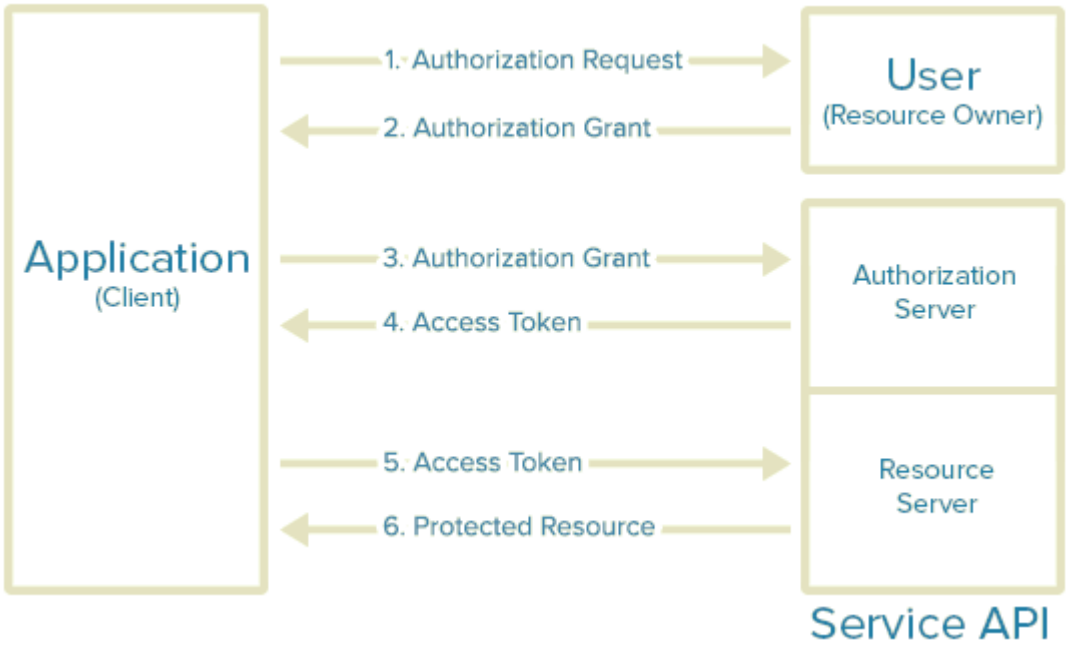

Рис. 3.1. Взаємодія OAuth2.

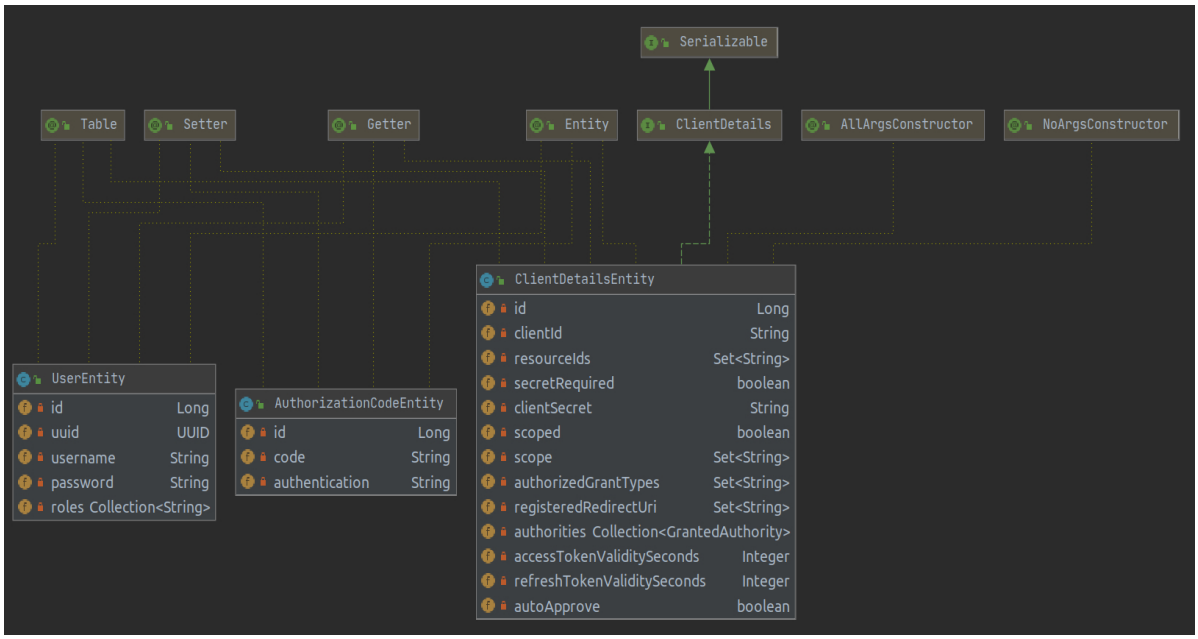

Рис. 3.1.1. Основні моделі OAuth2.

В основі сервісу зберігається три моделі:

1) UserEntity – зберігає основну інформацію користувача, а саме: ідентифікатор, логін ім’я, пароль та список ролей;
2) AuthorizationCodeEntity – зберігає інформацію для клієнтів під виглядом коду доступу та унікальної аутентифікацію;
3) ClientDetailsEntity – зберігає основну інформацію клієнта.

Ідентифікатор клієнта і секрет клієнта.

Після реєстрації додатки сервіс створить облікові дані клієнта - ідентифікатор клієнта (client ID) і секрет клієнта (client secret). Ідентифікатор клієнта є публічно доступну рядок, яка використовується API сервісу для ідентифікації додатка, а також використовується для створення авторизаційних URL для користувачів. Секрет клієнта використовується для аутентифікації справжності додатки для API сервісу, коли додаток запитує доступ до аккаунту користувача. Секрет клієнта повинен бути відомий тільки з додатком і API.

В результаті проведення такого покращення, доступ до даних користувача безпечно збережений і ніхто не може отримати конфіденційну інформацію, поки не отримає відповідний токен доступу. У разі, якщо дехто намагається проникнути в систему – він отримує наступне повідомлення:

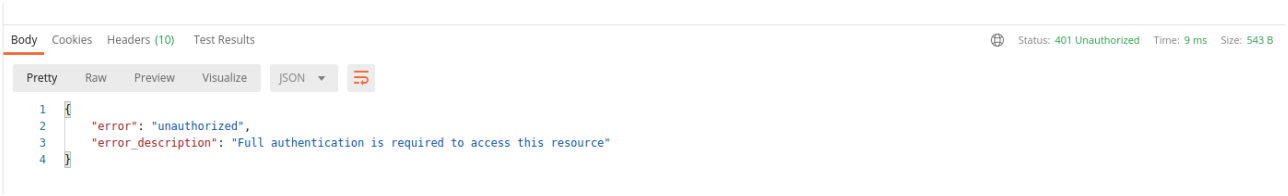

Рис. 3.1.2. Повідомлення про відсутність дозволу OAuth2.

Важливо нагадати, що системою передбачено створення клієнту по замовчуванням та двома користувачами, щоб адміністратом мав можливість почати роботу з сервісом.

Процес створення клієнту по замовчуванням:

1) Перевірка чи існує клієнт в системі;
    a. Якщо клієнт існує – створення не відбувається;
    b. Якщо клієнт не існує – продовження створення;
2) Завантаження «сікрету» та потрібних даних для клієнта;
3) Створення запису в базі даних.

3.2. Аспектно-орієнтована складова

В рамках покращення виконання технічної платформи, було реалізовано аспектно-орієнтована складова системи, яка відповідає за певний спектр дій, які можуть виконуватися незалежно від основного процесу.

Коли ми розглядали модулі Spring Framework, ми згадували про Аспект-орієнтованому програмуванні (далі - АОП). АОП є одним з ключових компонентів Spring. Сенс АОП полягає в тому, що бізнес-логіка програми розбивається нема на об'єкти, а на "відносини" (concerns).

Ключовий одиницею в ООП є "об'єкт", а ключовий одиницею в АОП - "аспект". Як приклад "аспекту" можна привести безпеку, кешування, логирование і т.д. Впроваджень залежностей (DI) дозволяє нам відокремлювати об'єкти додатки один від одного. АОП, в свою чергу, дозволяє нам відокремлювати наскрізні проблеми (cross-cuttings) від об'єктів, до яких вони належать.

Модуль AOP в Spring забезпечує нас такими сутностями, як "перехоплювачі" (interceptors) для перехоплення додатки в певні моменти. Наприклад, коли виконується певний метод, ми можемо додати якусь функціональність (наприклад, зробити запис в лог-файл програми) як до, так і після виконання методу.

Один з таких перехоплювачів було реалізовано в рамках магістерської роботи, а саме:

1) Реалізуємо перелік можливих виконань (можливі також і інші варіанти):

```
public enum EmailType {

    START_COURSE,
    FINISH_LESSON

}
```

2) Створимо аннотацію, яка буде містити в собі перелік з пункту 1:

```java
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface TriggerSendEmail {

    EmailType target();

}
```

3) Реалізуємо аспектний процес, який за логікою буде виконувати відправлення повідомлення на email:

```java
@Around("@annotation(annotation) && args(event)")
public Object processingAspect(ProceedingJoinPoint point, TriggerSendEmail annotation, Object event) throws Throwable {
    Object proceed = point.proceed();
    if (isEnable) {
        sendEmailNotification(annotation, event);
    }
    return proceed;
}
```

В параметрах методу потрібно вказати відразу три компоненти: point, annotation, event. Кожен з них відповідає за свою частину роботи. Перший потрібен для коректної роботи усього аспектного механізму. Другий (annotation) – потрібен для указання конкретного місця, в якому буде задіяна аспектна логіка. Останній параметр виконує функцію передачі усіх необхідних даних безпосередньо до аспектної функції.

3.3. Побудова та розгортання платформи на віддаленому сервері

Обов'язковим параметром для сучасної платформи являється швидка побудова та розгортання на віддаленому сервері. Сьогоденні тенденції значно облегшили спосіб такого розгортання. Як було вказано в розділах раніше, платформа, а саме її виконуючий код обгортається в спеціальну оболочку, з якою можна працювати як з єдиним цілим об'єктом.

Базовий будь-який додаток на мові Java компілюється і на основі нього створюється певний jar архів, який може зчитати будь-яка JVM. Так як в платформі організовано два основних діючих мікросервіси (app, auth) – то і архівів, відповідно, буде два. За допомогою «мавен» плагінів та команди mvn clean install платформа з легкістю будує фінальні виконуючі архіви.

Наступним кроком йде побудова докер контейнера. В основі контейнеру лежить певний image, надалі – образ. За допомогою докер інструкції, після побудови архіву jar, програма вмонтовує його в середину образу, а інструкція, в свою чергу, має JVM основу. Тому подальші будь-які виконання з архівом призведуть до коректної роботи в середині контейнеру. Приклад докер інструкції:

```
FROM openjdk:8
ADD target/app-module.jar /opt/app-module.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/opt/app-module.jar"]
```

Порядково:

- Основа образу
- Вмонтування архіву в середину образу
- Відкриття необхідного порту
- Виконання команди для запуску джава програми

Таким чином, на основі цієї інструкції створюється образ для майбутнього старту в середині докеру. По аналогії створюється і образ для auth модуля.

На основі вищезазначених образів було створено докер “compose” інструкція (надалі – компонування). Така інструкція максимально описує поведінку контейнерів і допомагає докеру зрозуміти що саме і в який момент має бути піднятим. Приклад такої інструкції:

```
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.5.2
    container_name: zookeeper
    ports:
      - 54132:54132
    command: /etc/confluent/docker/run
    environment:
      - ZOOKEEPER_CLIENT_PORT=54132
  kafka:
    image: confluentinc/cp-kafka:5.5.2
    container_name: kafka
    ports:
      - 54133:54133
    command: /etc/confluent/docker/run
    environment:
      - KAFKA_ADVERTISED_HOST_NAME=kafka
      - KAFKA_ADVERTISED_LISTENERS=LISTENER_DOCKER_INTERNAL://kafka:19092,LISTENER_DOCKER_EXTERNAL://localhost:54133
      - KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      - KAFKA_INTER_BROKER_LISTENER_NAME=LISTENER_DOCKER_INTERNAL
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper:54132
      - KAFKA_BROKER_ID=1
      - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
      - TOPIC_AUTO_CREATE=TRUE
    depends_on:
      - zookeeper
```

Рис. 3.3.1. Інструкція компонування асинхронних сервісів.

В вищезазначеній інструкції описано як правильно побудувати додаткові виконуючі сервіси задля коректної асинхронної роботи, а саме: Zookeeper & Kafka.

```
auth-postgres-db:
  image: mdillon/postgis:10
  container_name: auth-postgres-db
  ports:
    - 5434:5432
  environment:
    - POSTGRES_DB=education_auth
    - POSTGRES_USER=education-auth
    - POSTGRES_PASSWORD=education-auth
  # Default script execution when container is run
  #volumes:
  #  - ./init.sql:/docker-entrypoint-initdb.d/init.sql
auth-education:
  image: education_auth:latest.20.9-SNAPSHOT
  container_name: auth-education
  ports:
    - 8081:8081
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://auth-postgres-db:5432/education_auth?autoReconnect=true&useSSL=false
  depends_on:
    - auth-postgres-db
```

Рис. 3.3.2. Інструкція компонування основних сервісів.

На вищезазначеній інструкції показано коректне розташування бази даних та основного виконуючого сервісу.

Всю інструкцію та всю інфраструктуру тепер можна підняти лише однією командою, а саме: *docker-compose up -f filename.yml -d*

Результатом виконання цієї команди стане повністю піднята платформа, але поки локально. Також за допомогою мавен-докер плагіну, платформа з легкістю була автоматизована. Тепер щоб згенерувати архів, побудувати образ і підняти компонування, достатньо ввести наступну команду*:*

mvn clean install -Pdocker

Побудова архітектури для розгортання платформи віддалено.

Перед налаштуванням слід сказати, що провайдером віддалених серверів була вибрана AWS cloud, компанії Amazon. Також було створено віддалений сервер EC2 задля отримання як такої віддаленої машини та її IP. Інструкції Ansible, які дозволяюсь під’єднатися до серверу та провести усі необхідні налаштування, побудовані з урахуванням так званих ролей, які дозволяють більш активніше розробляти подібні механізми.

```
- name: Upgrade all packages
  apt:
    upgrade: dist

- name: Install required packages
  apt:
    name: [ 'apt-transport-https',
            'ca-certificates',
            'curl',
            'gnupg-agent',
            'software-properties-common',
            'python-pip' ]
    state: latest
    update_cache: yes
  become: true

- name: Add Docker GPG apt Key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: Save the current Ubuntu release version into a variable
  shell: lsb_release -cs
  register: ubuntu_version

- name: Add Docker Repository
  apt_repository:
    repo: "deb [arch=amd64] https://download.docker.com/linux/ubuntu {{ ubuntu_version.stdout }} stable"
    state: present

- name: Install Docker
  apt:
    name: "docker-ce"
    state: present
```

Рис. 3.3.3. Інструкція для встановлення софту.

Вищезазначена інструкція під’єднується до віддаленого серверу та інсталлює усі необхідні програми. Щоб запустити скріпт, потрібно виконати наступну команду:

ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=test first_playbook.yml

3.4. Пошуковий механізм

Для доступу до даних з таблиці БД був з нуля створений новий механізм пошуку, який активно працює для пошуку курсів та юзерів. Пошук може реалізуватися по будь-яким параметрам. Наприклад для пошуку курсу доступна безліч параметрів, таких як: дата початку, дата закінчення, назва, вартість, складність, кількість лекцій і тд. Цей механізм був реалізований на основі SpecificationBuilder.

Spring Data JPA надає багато способів мати справу з сутностями, включаючи методи запитів та власні запити JPQL. Однак іноді потрібен більш програмний підхід: наприклад, Criteria API та її специфіка з SpecificationBuilder.

API критеріїв пропонує програмний спосіб створення набраних запитів, що допомагає нам уникнути синтаксичних помилок. Навіть більше, коли ми використовуємо його з API Metamodel, він виконує перевірку під час компіляції, чи використовували ми правильні імена та типи полів.

Однак це має і свої мінуси: доводиться писати багатослівні логіки, роздуті шаблоном коду.

Тому як результат, на платформі реалізовано три способи використання запитів критеріїв:

- створення класу DAO - це найпростіший і найбільш гнучкий спосіб
- розширення інтерфейсу @Repository до безшовної інтеграції з автоматичними запитами
- використання предикатів у примірниках специфікації, щоб зробити прості регістри чистішими та менш детальними

Приклад формування критеріїв для пошуку:

```
private String title;

@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long costStart;
@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long costEnd;

@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long createdStartDate;
@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long createdEndDate;

@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long beginStartDate;
@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long beginEndDate;

@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long finishStartDate;
@Min(value = 0)
@Max(value = Long.MAX_VALUE)
private Long finishEndDate;
```

Рис. 3.4. Критерії для пошуку.

Таким чином, з появленням нових полів та нових критеріїв, програмний код зможе адаптивно приймати зміни і не порушувати загальний контракт і не ламати існуючі правила.

3.5. Аналітика та збір відгуків користувачів

Для мотивуючого навчання та налагодженної взаємодії студента та викладача, була розроблена система збору відгуків користувачів. Будь який користувач, який проходить курс, має декілька варіантів синхронізації в спілкуванні між викладачем або іншими студентами. Насамперед, це коментарі під кожною лекцією.

Бізнес логіка процесу створення коментаря наступна:

1. Користувач заходить на свій курс та ознайомлюється з потрібною лекцією.
2. Після проходження лекції користувач бачить поле для вводу коментаря.
3. Користувач може ввести свій коментар або переглянути існуючі.

Наступний варіант взаємодії – відгуки від студентів, які надсилаються безпосередньо на пошту викладачу. Лишити такий відгує є можливість лише 2 рази за весь курс: після проходження половини лекцій, та після закінчення курсу. Користувач має право відмовитися від проходження анкети опитування та відгуку. Але в разі заповнення – студент бачить свій відгук у себе на сторінці аккаунту.

Аналітичний комплекс, який доступний виключно викладачам. Користувач з роллю викладач має можливість зробити запит на пошук користувачів за будь-якими критеріями, наприклад – усі користувачі, які підписані на певний курс. Для викладача доступна можливість перегляду даних студента і генерувати PDF файли, в яких буде міститися інформація про обраних студентів.

ВИСНОВОК ДО РОЗДІЛУ 3

Таким чином, була розроблена система по обробці, транспортуванню, створенню або видаленню даних. Слід зауважити, що платформа Spring дає можливості відстежувати всі процеси, які відбуваються на кожному рівні, які в свою чергу, описані вище.

Відповідно до кожного рівня, процеси працюють лише з тими даними, про які й потрібно їм знати на програмному рівні. Кожен сервіс чи модуль відповідає лише за своє призначення і не більше. Система покрита інтеграційними тестами задля перевірки, що після внесення нових функцій нічого колишнього не було зламано.

Також система по розгортанню платформи на віддаленому сервері дозволила максимально наблизити реальність задач, які існують в сучасних проектах компаній-гігантів.

Отже, правильно побудована архітектура веб додатку - просота до реалізації нового майбутнього функціоналу та здатність до розширення. Це підтверджується новою реалізацією пошукового механізму та сучасного сервісу для збору аналітичних даних.

Слід зауважити, що система та системні дані захищені від стороннього проникнення за допомогою єдиного сховища токенів доступу. Тобто не маючи токену – ніхто не в змозі зробити запит в систему.

РОЗДІЛ 4
ОПИС РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Демонстрація практичних можливостей розробленого програмного комплексу

Для того, щоб розпочати роботу з сервером - користувач повинен ввійти в систему вже з існуючим логіном і паролем.

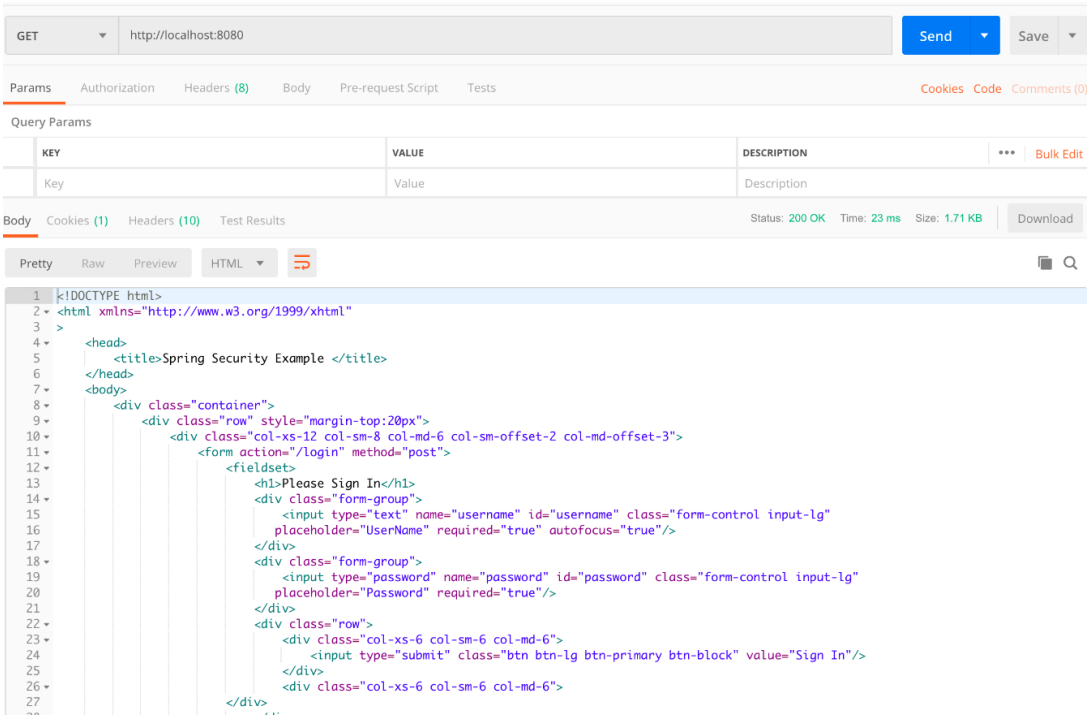

Рис. 4.1.1. Запит до авторизації.

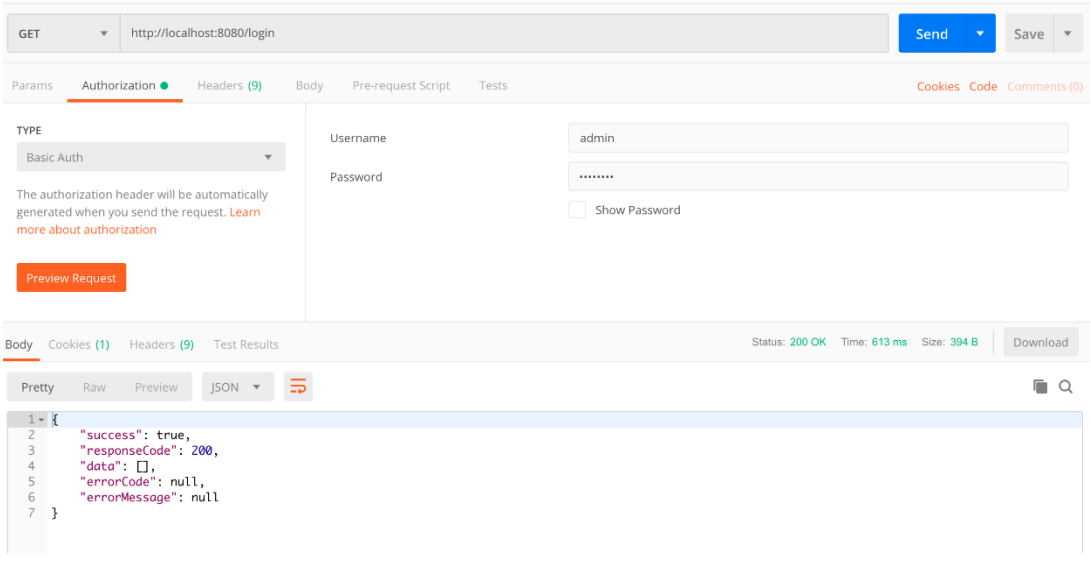

Рис. 4.1.2. Успішний логін користувача.

Базово, користувач має змогу переглянути усі курси, але щоб розпочати навчання – йому потрібно завершити реєстрацію в системі, заповнивши свій профіль.

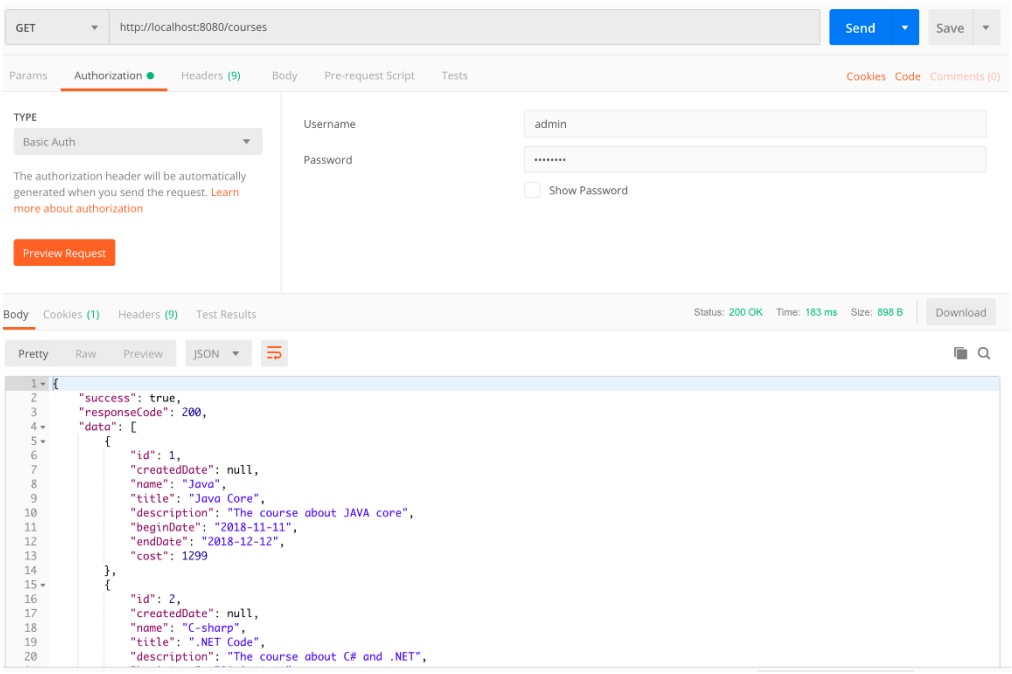

Рис. 4.1.3. Перегляд усіх доступних курсів.

Якщо користувач не має права доступу до певного курсу, або заповнення даних на відбулося – він не зможе отримати доступ до курсу.

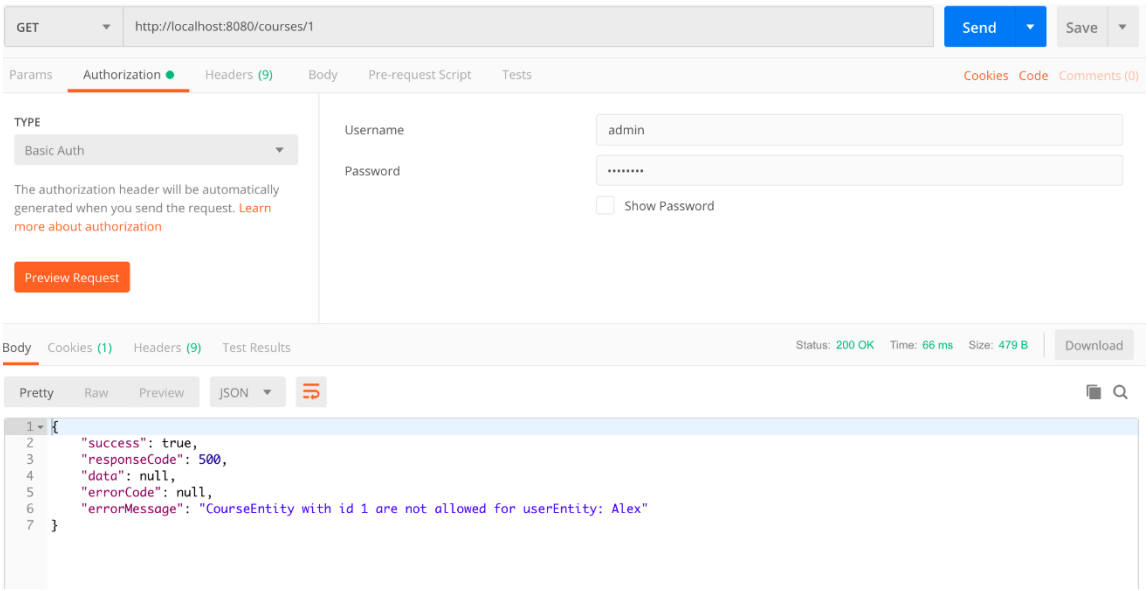

Рис. 4.1.4. Спроба переглянути не дозволений курс.

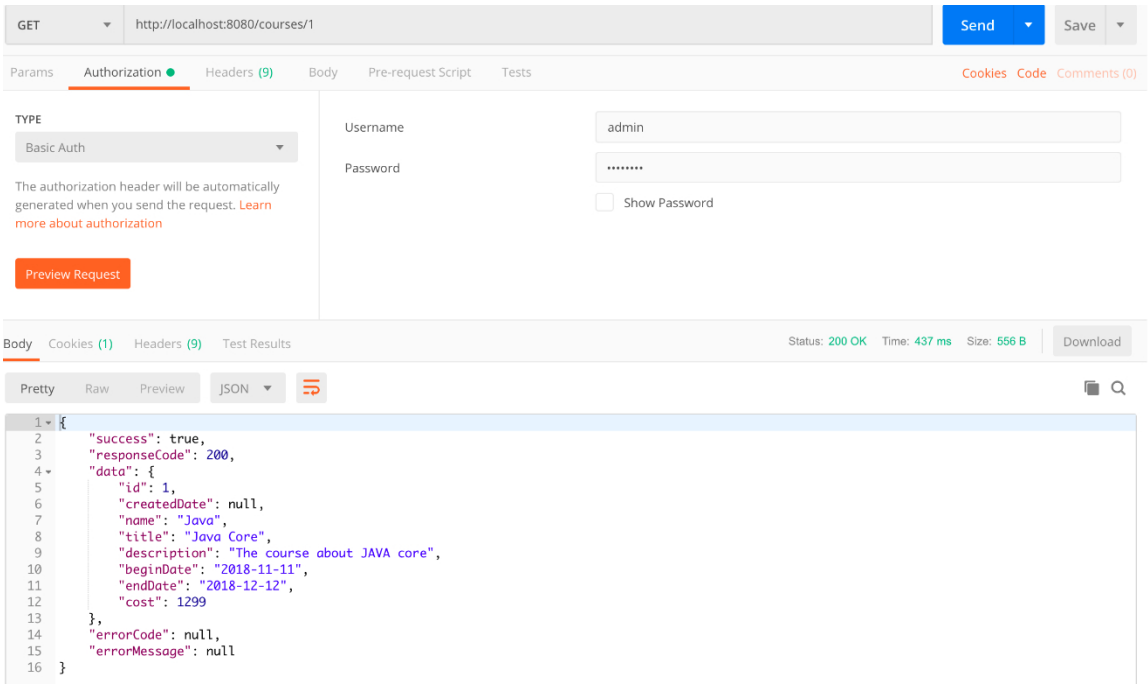

Рис. 4.1.5. Перегляд дозволеного курсу.

Нижче зазначено приклад заповнення необхідної інформації задля завершення реєстрації:

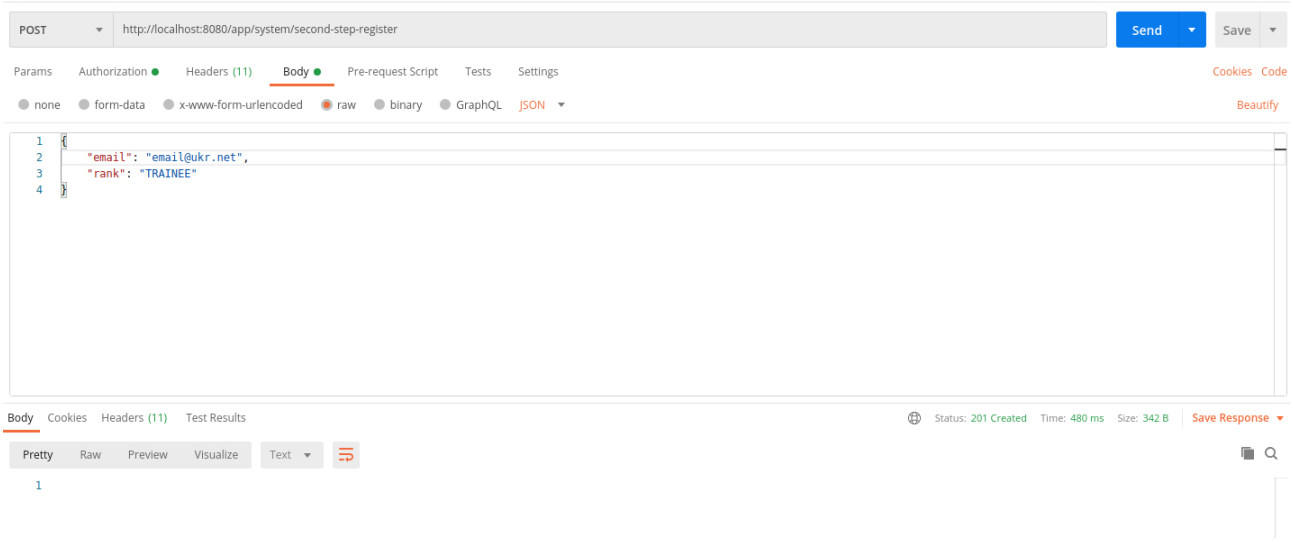

Рис. 4.1.6. Фінальна реєстрація користувача.

Бізнес логіка такої поведінки наступна:

1. Користувач створений в OAuth сервісі і намагається отримати доступ до курсу;

2. Користувач має пройти завершуючу реєстрацію, де введе свою пошту, рівень знань та інші параметри анкети;
    a. Система зв’яже конкретного користувача з його UID, який постачається через токен;
3. Користувач надсилає запит зі своєю інформацією;
4. Реєстрація завершена.

Бізнес логіка та приклад старту курсу. Примітка – користувач має бути зареєстрований.

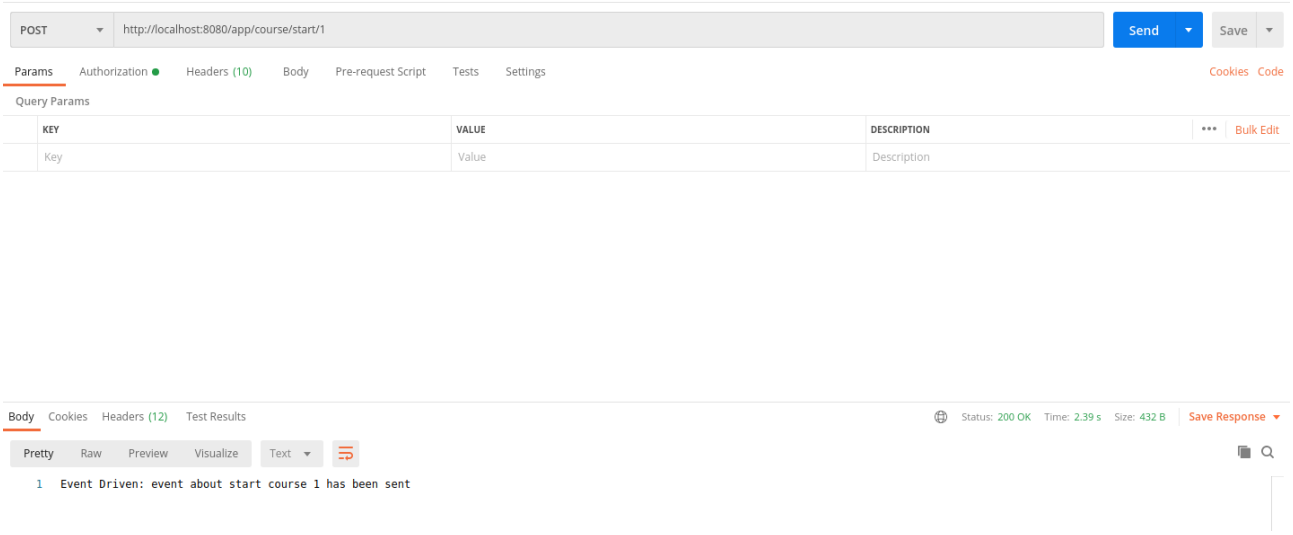

Рис. 4.1.7. Старт курсу.

Механізм старту курсу для користувача:

1. В систему надходить запит про старт курсу;
    a. Перевірка на валідність запиту;
    b. Перевірка на те, що користувач не починав раніше цей курс;
2. Створення івенту, який сигналізує про початок курсу;
3. Створення аспекту, який займається відправкой повідомлення на електронну адресу;
4. Оновлення інформації користувача на основі отриманого івенту;
5. Користувач має доступ до курсу.

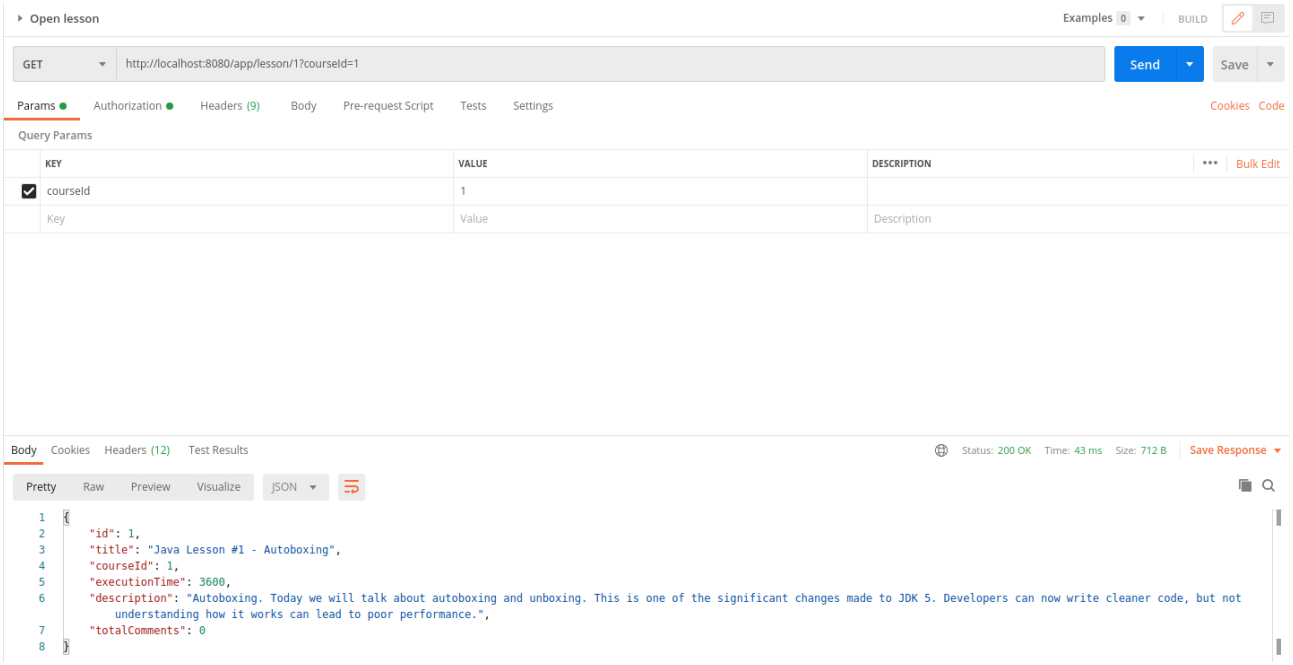

Рис. 4.1.8. Ознайомлення з лекцією.

Процес ознайомлення з лекцією та надсилання івенту про завершення лекції:

1. Надсилання запиту про лекцію;
2. Ознайомлення та надсилання запиту про завершення проходження лекції;
    a. Валідація даних;
    b. Створення івенту для надсилання повідомлення на електронну пошту;
3. Оновлення інформації на основі отриманого івенту;
4. Користувач завершив лекцію.

Приклад отриманого листа на пошту:

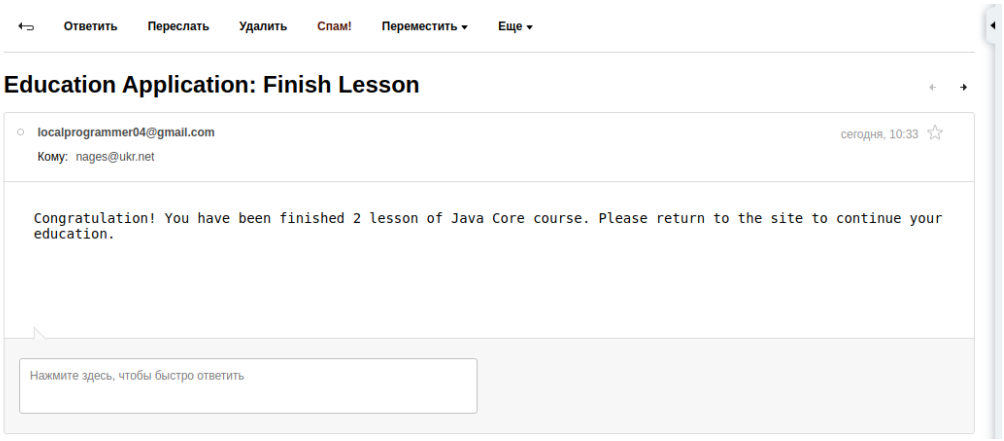

Рис. 4.1.9. Отриманий лист на пошті користувача.

У доступному функціоналу адміністратора (або звичайного користувача) є можливість переглянути свій профіль в системі.

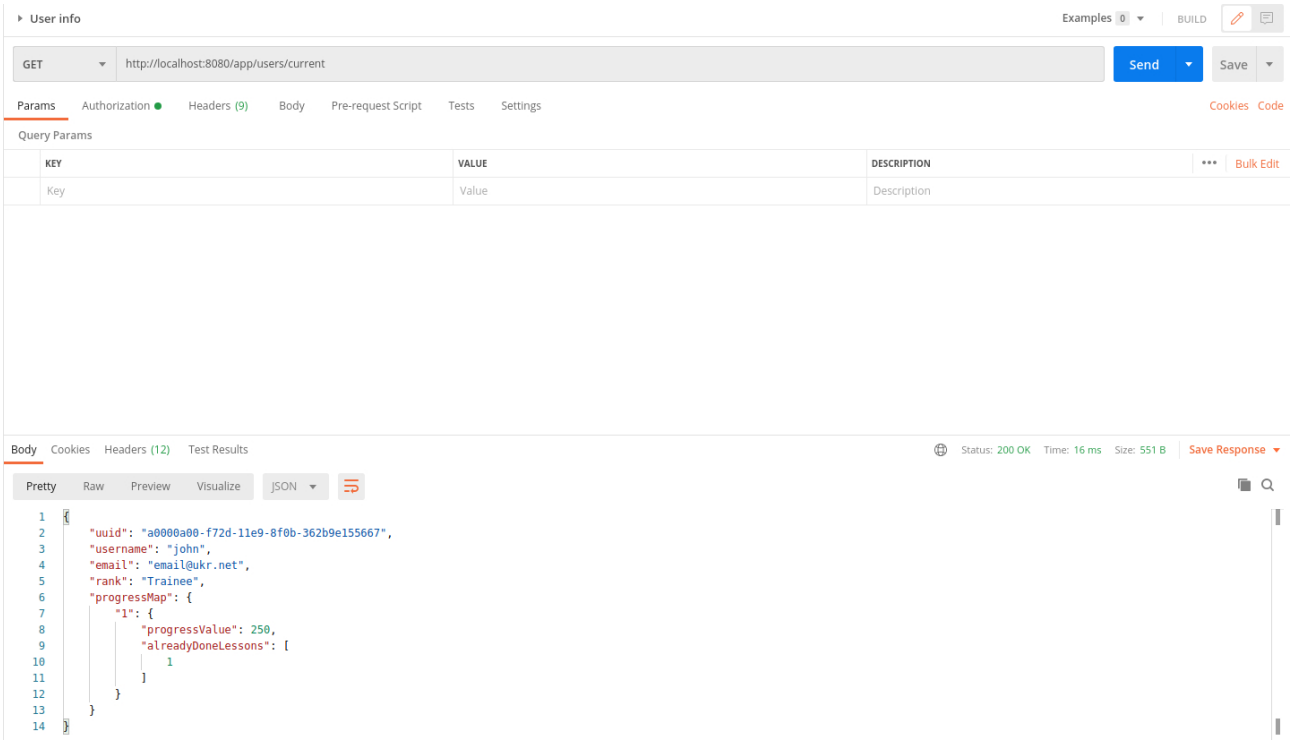

Рис. 4.1.10. Отриманий лист на пошті користувача.

Процес заповнення форми відгуку від студенті. Система запрограмована таким чином, що власник курсу має можливість заповнити принаймні два посилання, за яким користувач (студент) матиме змогу залишати відгук.

Наразі в системі доступно дві події, при досягненні студентом яких, він матиме змогу лишити такий відгук. А саме: після проходження рівно половини матеріалів курсу, студентові приходить нагадування і лист на пошту (додатково) з проханням лишити відгук. Форма відгуку про проходження половини курсу виглядає наступним чином:

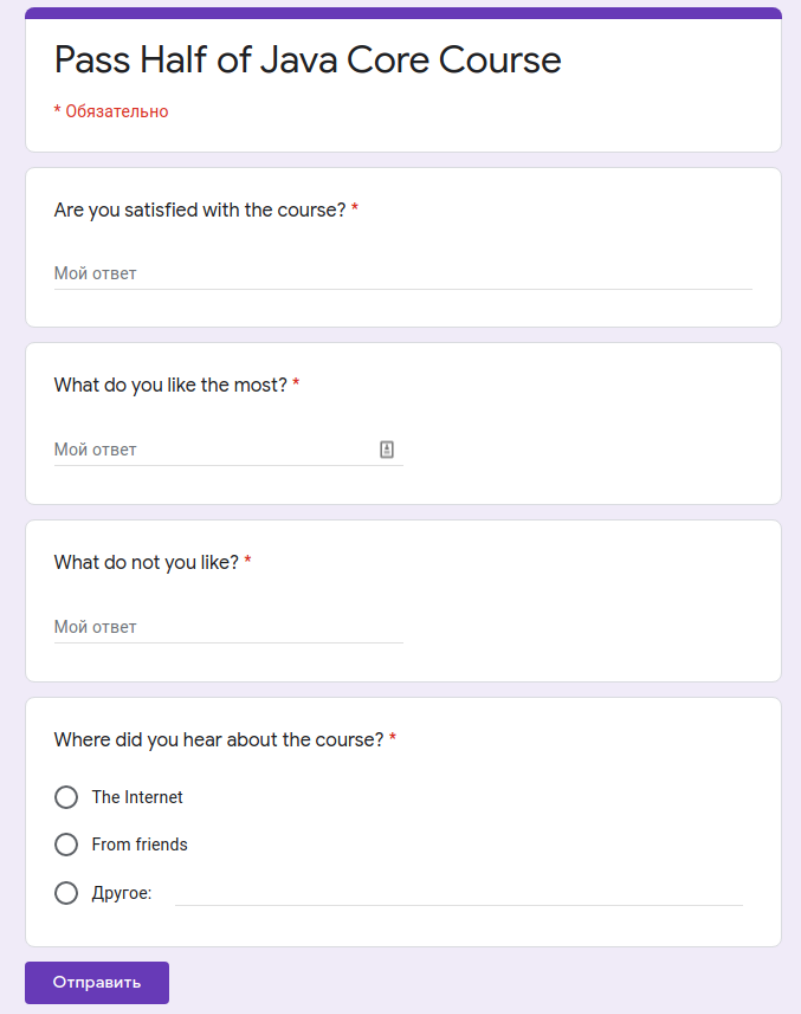

Рис. 4.1.11. Форма опитування 1.

Після заповнення такої форми, викладачеві будуть доступні усі відповіді у його особистому кабінеті. Слід зауважити, що реалізація відгуків була створена за допомогою гугл форм.

На поточний момент, в системі також існує можливість залишати відгук після проходження курсу в цілому. В такому випадку студент, вивчивши всі лекції, та після проходження усіх матеріалів, має змогу завершити курс. Процес завершення курсу був описаний вище. Після офіційного завершення курсу, користувач (студент) має змогу бачити посилання на форму з відгуком про завершення курсу. Ця форма також реалізована за допомогою гугл форм та виглядає наступним чином:

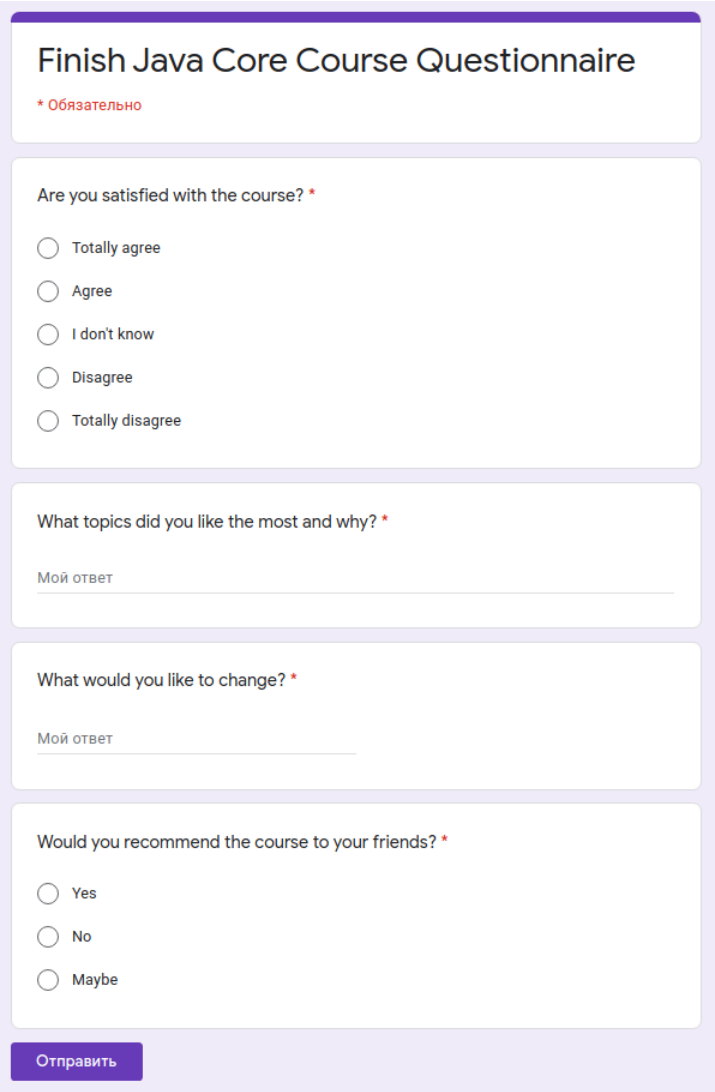

Рис. 4.1.12. Форма опитування 2.

Також додатково посилання на заповнення відгуку міститься в листі, який отримує студент при завершенні курсу. Приклад повідомлення наступний:

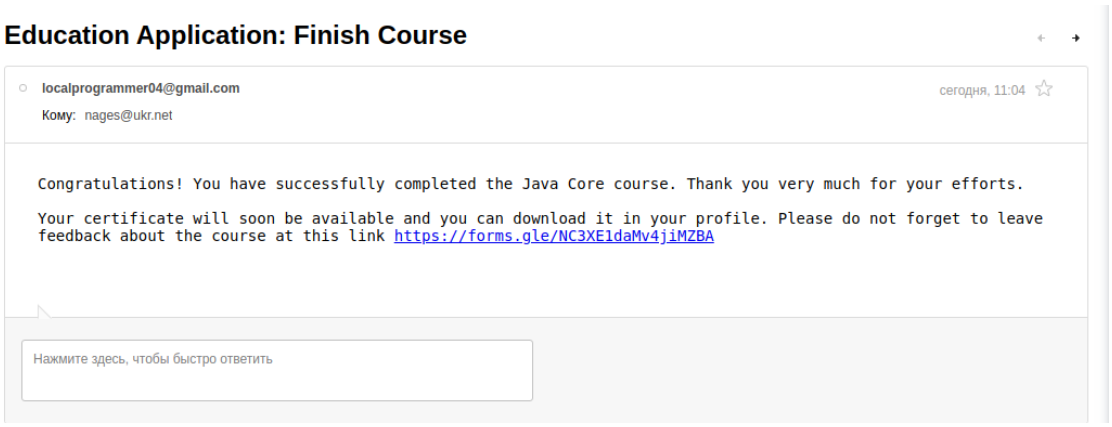

Рис. 4.1.13. Повідомлення на пошті студента.

Перейдемо до останніх найвагоміших функціональних можливостей.

Насамперед для користувача з правами адміністратора доступний функціонал імпорту інформації про студентів. Імпорт відбувається в формат CSV і можуть бути відкриті будь-яким текстовим редактором. На поточний момент для імпорту даних доступно 6 колонок з інформацією про юзерів. Кількість колонок та дані легко конфігуруємі. Також передбачена можливість додавання нового функціоналу в майбутньому. Нижче відображено приклад такого імпорт-файлу:

	A	B	C	D	E	F
1	User UUID	User Name	Email	Rank	Total Finished Courses	Total In Progress Courses
2	b1111b11-f72d-11e9-8f0b-362b9e15560e	admin	nages@ukr.net	Trainee	0	0
3	a0000a00-f72d-11e9-8f0b-362b9e15560e	john	nages@ukr.net	Trainee	1	0
4						
5						
6						
7						
8						
9						
10						
11						

Рис. 4.1.14. Імпорт даних користувачів.

Фінальна, але також вагома логіка платформи – генерація сертифікату після успішного проходження курсу студентом. Студент після завершення усіх активностей на курсі має можливість зайти на свій профіль та скачати сертифікат по пройденому курсу. Сертифікат генерується в форматі PDF та виглядає наступним чином:

Рис. 4.1.15. Сертифікат студента.

4.2. Програмна документація

Документація платформи представлена у вигляді JavaDoc, це особливий підхід до написання документації, яка корисна як для розробників, так і для клієнтів.

Покриття документацією досягається за рахунок написання інформації під кожним кодовим методом, який написаний на Service рівні. Структура коментарів Javadoc дуже схожа на звичайний багаторядковий коментар, але ключовою відмінністю є зайва зірочка на початку.

Коментарі Javadoc можуть бути розміщені над будь-яким класом, методом або полем, яке потрібно задокументувати.

Існує багато тегів блоків, які допомагають створити належну документацію, і ми можемо включати всіляку різну інформацію. Також існує можливість використовувати основні теги HTML у коментарях.

- @param надає будь-який корисний опис параметра методу або введення, якого слід очікувати
- @return надає опис того, що метод може або може повернути
- @see створить посилання, подібне до тегу {@link}, але більше в контексті посилання, а не вбудованого
- @since вказує, яку версію до проекту додано клас, поле чи метод
- @version визначає версію програмного забезпечення, яка зазвичай використовується з макросами% I% та% G%
- @throws використовується для подальшого пояснення випадків, коли програмне забезпечення очікує винятку
- @deprecated дає пояснення, чому код був застарілим, коли він, можливо, був застарілий, і які альтернативи

Хоча обидва розділи технічно необов’язкові, нам знадобиться принаймні один для інструменту Javadoc, щоб створити щось значуще.

4.3. Проектування інтерфейсу

Майбутня розробка інтерфейсу для доступу до виконуючого сервісу не включає в себе громіздкі напрацювання, так як основні етапи в проектуванні інтерфейсу були успішно пройдена в рамках поточної магістерської дисертації.

За основний фреймворк для написання повноцінного UI було обрано ReactJS, але не виключено, що після ознайомлення і більш ретельного підбору, фреймворк буде змінений.

Виділимо інтерфейс та вигляд початкової сторінки для користувача:

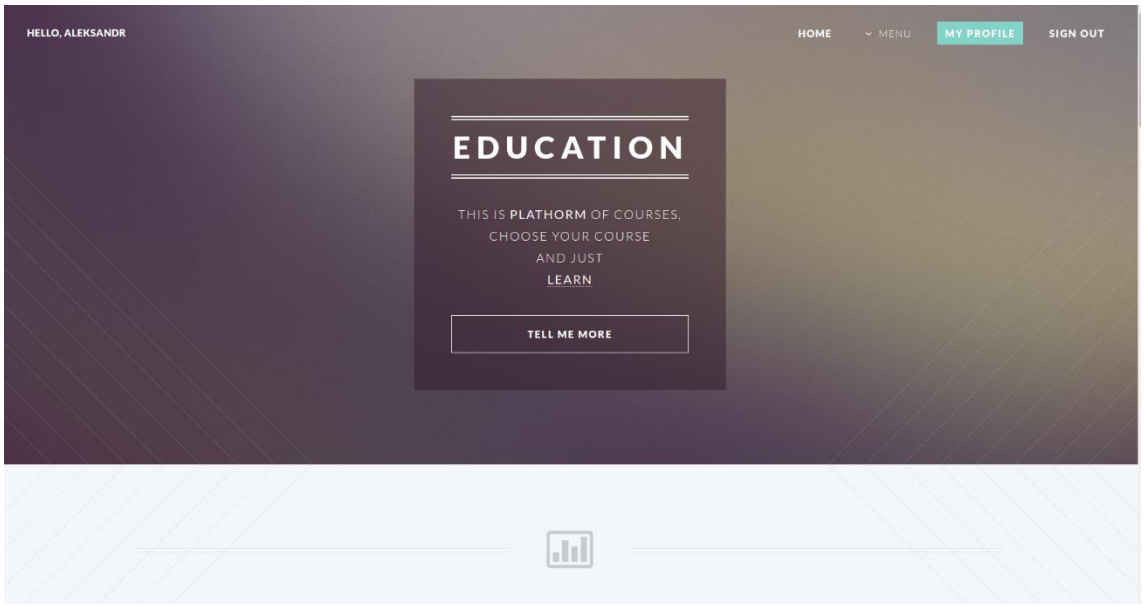

Рис. 4.3.1. Інтерфейс користувача, головна сторінка.

Як тільки користувач пройшов аутентифікацію та авторизувався в системі – йому надано можливість бачити список з доступних курсів на вибір.
Плановий вигляд інтерфейсу наступний:

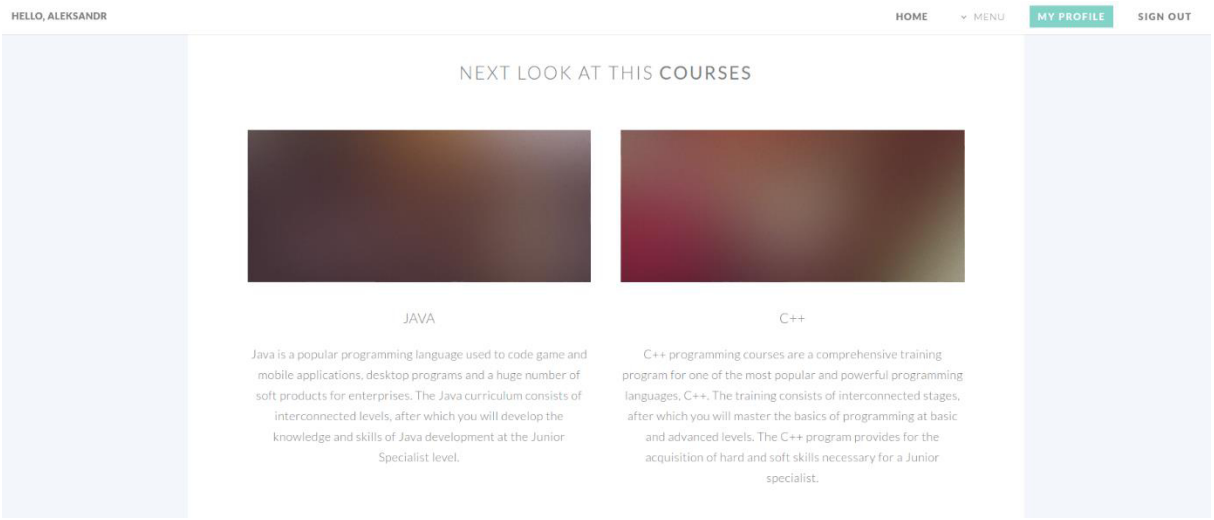

Рис. 4.3.2. Інтерфейс користувача, вибір курсів.

Після натискання, проходження реєстрації на курс, користувачу надається доступ до лекційних матеріалів.

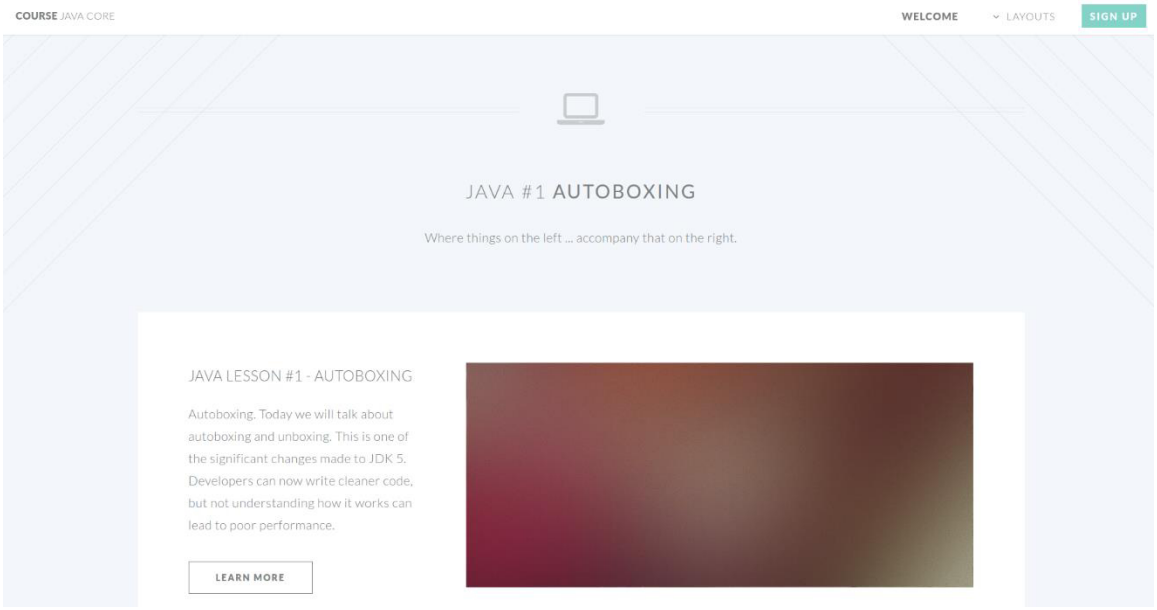

Рис. 4.3.3. Інтерфейс користувача, лекція.

4.4. Розроблення стартап-проекту

Розроблення стартап-проекту є невід'ємною частиною магістерської дисертації.

4.4.1. Опис ідеї проекту

Проект програмна платформа підтримки процесів дистанційного навчання.

Відмінними якостями продукту є:

- надає розширювану функціональність для навчання абсолютно не виходячи з дому;
- має розподілений функціонал як для користувача студента, так і для користувача викладача;
- можливість отримувати сертифікат за проходження курсів;
- проведення аналітичних розрахунків для викладача та захищеність даних в цілому.

Унікальність технології – вибірка та опрацювання даних проходить через сервернуу частину, де в кінцевому випадку дані представляються у форматі JSON для легкого подальшого маніпулювання зі фронтенд сторони.

Таблиця 4.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
1	2	3
Створення модуля доступу до навчальних матеріалів	IT навчання (програмування, дизайн, менеджмент, тощо)	Використання даного модуля забезпечить в короткі терміни доступ до навчальних матеріалів без перешкод, що надасть цільове забезпечення для студентів а також аналітичні дані для викладачів. Таким чином скоротиться час на навчання в цілому, так як процес пробігає виключно онлайн.

Основними аналогами даного проекту є системи навчання на різних інтернет-ресурсах.

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів наведено у таблиці 4.2.

Таблиця 4.2

Аналіз потенційних техніко-економічних переваг

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Mate Academy	Coursera	Udemy			
1	2	3	4	5	6	7	8	9
1	Простота освоєння	Не є складною для базових дій. Новий функціонал додається легко.	Невисока	Легка	Невисока		+	

Таблиця 4.2 (закінчення)

Аналіз потенційних техніко-економічних переваг

1	2	3	4	5	6	7	8	9
2	Вартість використання	Безкоштовний	Система оплати після навчання	Є платні курси	Є платні курси		+	
3	Регіони забезпечення	Весь світ, вибірково місто чи країна	Вибіркове місто чи країна	Весь світ, вибірково місто чи країна	Весь світ, вибіркове місто		+	
4	Види медіа-ресурсів для пошуку	Google, Інтернет-портали, новини	Інтернет-картинки, новини, YouTube, Google	Блоги, новини	Соціальні мережі, блоги, форуми			+

4.4.2. Технологічний аудит ідеї проекту

Розробка модуля вибірки навчальних матеріалів з серверної частини не потребує складної логіки для користувача. Лище викладач має змогу змінювати курси або ж додавати нові.

Таблиця 4.3

Технологічний аудит

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	2	3	4	5
1	Застосування архітектурного підходу MVC	Існуючі приклади реалізації підходу	Наявна	Засоби є загальнодоступними
2	Перетворення даних з БД до формату JSON	Відкриті бібліотеки та модулі, приклади реалізації	Наявна	Засоби є загальнодоступними
3	Можливість розширювати функціонал системи	Відкриті бібліотеки та модулі, приклади реалізації	Необхідно розробити	Засоби є загальнодоступними
4	Підготовка даних для аналізу на стороні серверу	Відкриті бібліотеки та модулі, приклади реалізації	Необхідно розробити	Засоби є загальнодоступними
Обрані технології проекту: Java, Spring, Docker, Ansible, Hibernate, PostgreSQL				

4.4.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення сприятливих обставин, які можна використати для отримання переваг ринкових можливостей є важливою частиною при запуску стартап-проекту. Це дозволяє спланувати напрями розвитку проекту, визначити потреби можливих клієнтів та оцінити існуючі ризики.

У таблиці 4.4 відображено характеристику потенційного ринку стартап-проекту.

Таблиця 4.4

Характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	4
2	Загальний обсяг продаж, грн/ум.од	-
3	Динаміка ринку (якісна оцінка)	Стабільна
4	Наявність обмежень для входу (вказати характер обмежень)	Існують, реєстрація користувача
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	30%

У таблиці 4.5 наводяться визначені потенційні клієнти, а також їх характеристика.

Таблиця 4.5

Потенційні клієнти

№ п/п	Потреба, яка формує поточний ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп	Вимоги споживачів
1.	Потреба у вивченні нової інформації з певних IT тематик для подальшого використання	Великий, середні та малий бізнес, що працюють над вдосконаленням знань своїх працівників	Не існує.	1. Надійна робота системи. 2. Швидкий доступ до матеріалів 3. Можливість вибрати курс під конкретні потреби

Для виявлення потенційних загроз можна спиратися на наступний список питань: які нові компанії на ринку мають в порівнянні кращий функціонал, які найбільш привабливі умови і продукти пропонують клієнтам конкуренти.

У таблиці 4.6 наведені фактори загроз, що перешкоджають ринковому впровадженню.

Таблиця 4.6
Фактири загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Поява нових сильних конкурентів	Проблеми в просуванні системи	Запуск системи в Україні та за її межами, запуск рекламної кампанії
2	Погіршення фінансового стану організацій	Зменшення доходу сервісу	Аналіз поточних доходів, зменшення витрат
3	Відсутність зацікавленості у потенційних користувачів	Сервіс не викликає довіру у користувачів	Підвищення якості, додання акцій
4	Нестача об’ємів сховища для збереження даних	Клієнти можуть мати обмежені локальні технічні ресурси, недостатні для повноцінної роботи системи	Купівля більшого об’єму сховища

У таблиці 4.7 наведено більш детальні риси конкуренції на ринку за М. Портером.

Таблиця 4.7

Риси конкуренції на ринку за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари замінники
	1	2	3	4	5
	Udemy, Coursera	Інші стартапи, що містять сервіси для навчання	Прості клієнти, учасники великого, середнього та малого бізнесу	Прості клієнти, учасники великого, середнього та малого бізнесу	Кожен з продуктів є частковим замінником, адже не є універсальним сервісом

Порівняльний аналіз сильних та слабких сторін модуля навчання відображено у таблиці 4.8.

Таблиця 4.8

Порівняльний аналіз сильних та слабких сторін модуля

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з модулем вибірки даних з медіа-ресурсів						
			–3	–2	–1	0	+1	+2	+3
1	Інновації	18				+			
2	Цінова політика	20		+					
3	Можливість розширення системи	20				+			
4	Адаптація системи до різних ринків	18						+	

Враховуючи вище наведені таблиці зробимо висновок ринкового аналізу, та сформуємо його у вигляді SWOT- аналізу (таблиця 4.9).

Таблиця 4.9

SWOT-аналіз

Сильні сторони:	Слабкі сторони:
Інновації Цінова політика Можливість розширення системи Адаптація системи до різних ринків	Відсутність співпраці з інноваційними центрами
Можливості:	Загрози:
Великий потенціал розвитку ринку онлайн-послуг Вихід на український ринок Продаж бізнесу на початковій стадії росту Орієнтованість сервісу на користувача	Поява нових сильних конкурентів Погіршення фінансового стану організацій Відсутність зацікавленості у потенційних користувачів Нестача об’ємів сховища для збереження даних

На основі SWOT-аналізу визначено альтернативи ринкового впровадження стартап-проекту (таблиця 4.10).

Таблиця 4.10

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	2	3	4
1	Розробка системи з основним функціональним забезпеченням	Висока	6 місяців
2	Пошук замовників системи	Середня	6 місяців
3	Покращення основних можливостей системи на основі відгуків користувачів	Середня	8 місяців

Обрано 2 альтернативу, оскільки отримання ресурсів є більш простим та ймовірним, а строки реалізації – більш стислими.

4.4.4. Розроблення маркетингової програми стартап-проекту

Для формування маркетингової концепції продукту, спершу треба визначити результати аналізу конкурентоспроможності продукту (таблиця 4.11).

Таблиця 4.11

Результати аналізу конкурентоспроможності продукту

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Універсальність системи	Можливість вибрати різні курси	Сучасні методи побудови курсів та їх вибір
2	Змога розширити функціонал для власних потреб	Можливість змінити функціонал модуля під власні потреби	Впровадження сучасних рішень з сучасною архітектурою

Певну популярність отримала модель навчання, що складається з двох рівнів:

- студенти;
- викладачі.

ВИСНОВОК ДО РОЗДІЛУ 4

Таким чином, в цьому розділі був проведений ретельний огляд функціонування розробленого сервісу та його поведінки при різних вхідних даних. Були розглянуті як неправильні дані, так і правильні.

Також, в ході перевірки було виявлено, що при вводі користувачем коректних даних - він отримує бажаний результат та цілком може продовжувати коректну роботу з сервісом. Сервер був перевірений у так званому «активному помилковому тестуванні». При вводі різних помилкових даних, сервер відповідає коректно і так як запрограмовано, сервер вміє виводити різні повідомлення про помилку, що дає змогу адміністратору чи модератору досить швидко знайти причину некоректних даних.

ВИСНОВКИ

В рамках магістерського дипломного проекту був покращений і оптимізований сервер, який являє собою програмну платформу підтримки дистанційного навчання. Були реалізовані готові функції існуючих систем, визначено їх основні переваги та недоліки, і на основі цих даних був складений список основних нововведень, які потрібно було реалізувати, а саме: покращена взаємодія викладача та студента, генерація сертифікату за проходженням курсу, зворотній зв’язок студента з викладачем та автоматизація розгортання на віддаленій платформі.

Згідно з вимогою, була розглянута технологічна частина сервера. Аналіз сучасних фреймворків, які використовуються для роботи з мовою програмування Java показав, що наразі популярний спрінг фреймворк, тому було обрано Spring Boot, одну з найпопулярніших мовних середовищ Java, для реалізації сервера, вона проста, гнучка, а також містить безліч інших корисних вбудованих реалізацій для серверу.

Також в даному проекті розглядаються шаблони проектування такі як Builder, Factory, MVC, Controller-Service-Repository, Singleton, Prototype. Ця архітектура дає сервера значну гнучкість, простоту, оскільки логіка розділена на кілька рівнів, і будь-яка реалізація в майбутньому може бути легко замінена на іншу.

Відповідно до обраної архітектурою сервер легко буде піддаватися зміні чи модифікаціям в майбутньому. Деталі серверу і його функціонал чітко залогований та не дасть змоги допустити помилку. Під час тестування було перевірено, що сервер працює коректно і так як планується. Сервер активно працює з некоректними даними, виводячи відповідні повідомлення про помилки. З тим самим, навіть отримавши таку помилку, сервер запрограмовано на дії, які не призведуть до зупинки всієї системи.

СПИСОК ЛІТЕРАТУРИ

1. IT lessons [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://ruseller.com/lessons.php?id=666

2. Habrabahr [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/181772/

3. Web-architecture [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://tproger.ru/translations/web-architecture-101/

4. Spring Web MVC [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/336816/

5. Spring [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/333756/

6. Remote Job [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://dou.ua/lenta/articles/remote-job/

7. Gradle Build Tool [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://gradle.org/

8. Portal IT [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://dou.ua/lenta/articles/portrait-2016/

9. PostgreSQL [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://www.postgresql.org/docs/9.5/functions-json.html

10. Spring Data [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://dzone.com/articles/magic-of-spring-data

11. Spring Data [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/333756/

12. Spring Data [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/420901/

13. Spring MVC Interceptors [Електронний ресурс] : [Веб-сайт] – Режим доступу:
https://www.baeldung.com/spring-mvc-annotations

14. Security [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://docs.spring.io/autorepo/docs/spring-security/4.0.x/

15. Data, JPA [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://spring.io/blog/2011/02/10/getting-started-with-spring-data-jpa/

16. Security Filters [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/346628/

17. Boot Spring [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/post/352954/

18. Microservices [Електронний ресурс] : [Веб-сайт] – Режим доступу: https://habr.com/ru/company/otus/blog/413567/