Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

Розробка клієнт-серверних Web-застосувань з використанням Spring framework, ReactJS

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи

Старший викладач Борозенний С.О.

_____

(підпис)

“____” _____ 2021 р

Роботу виконав студент БП ІПЗ-3

Рожко А. С.

“____” _____ 2021 р

Київ 2021

Тема: **Розробка клієнт-серверних Web-застосувань з використанням Spring framework, ReactJS**

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	11.10.2020	
2.	Огляд літератури за темою роботи.	17.11.2020	
3.	Проектування архітектури застосунку	29.11.2020	
4.	Написання програмного коду серверної частини	14.02.2020	
5.	Тестування та відлагодження серверної частини	18.02.2020	
6.	Написання програмного коду клієнтської частини	25.03.2020	
7.	Тестування та відлагодження усього застосунку	29.03.2020	
8.	Написання текстової частини роботи	21.04.2021	
9.	Створення презентації доповіді	03.05.2021	
10.	Захист курсової роботи	17.05.2021	

Студент _____

Курівник _____

“_____”_____

Зміст

Вступ

Про доцільність розробки є сенс міркувати з точки зору студента-розробника, адже хоч сам програмний продукт, що був створений в процесі виконання курсовою роботи не є революційним на ринку IT, все ж має велику цінність для виконавця, адже гармонічно вписується у навчальну програму інженерії програмного забезпечення третього року навчання бакалаврату, адже має теми спільні з такими предметами: «Архітектури прикладних програм рівня підприємства», «Веб-програмування», «Бази даних», «Вступ до Java EE» - усі ці дисципліни так чи інакше сприяли розвитку знань та набуттю навичок, які отримали широке застосування під час виконання курсової роботи з написання соціальної мережі «Sprout».

Метою виконання курсової роботи є дослідження можливостей обраних технологій, набуття практичних навичок застосування запропонованого ними функціоналу, вивчення особливостей та осягнення «ідейних спрямувань» зазначених засобів розробки програмного забезпечення, набуття досвіду використання офіційної декументації розрозників для пошуку надійної інформації щодо рекомендацій використання відповідних інструментів, а саме: Spring Data, Spring Web, Spring Security, Spring Boot framework, ReactJS тощо.

Об’єкт дослідження – засоби розробки веб-застосунків.

Предмет дослідження – Spring Boot framework, ReactJS.

Мотивація до вибору технологій очевидна – java є однією з найпопулярніших мов для написання серверної частини веб застосунків, разом із тим Spring Boot framework є беззаперечним лідером серед наявних програмних каркасів для побудови сучасних веб застосунків з використанням java. Для створення графічного інтерфейсу в браузері клієнта використовується бібліотека ReactJS популярність якого порівнювана з такою в Angular. Оскільки сучасні програмні продукти мають набагато більший розмір, ніж їх попередники, постає проблема

контролю правильності нового коду, легкості та надійності внесення змін до наявного. Відомим недоліком мови програмування JavaScript є її «качина типізація», яка при цьому є ще і нестрогою. Це негативно впливає на досвід використання зазначеної мови та може призвести до того, що розробник у процесі роботи буде неодноразово «стріляти собі в ногу», саме тому було ухвалено рішення відмовитися від JavaScript у явному вигляді на користь TypeScript, який дозволяє використовувати статичну типізацію, чим значно полегшує та у довгостроковій перспективі пришвидшує розробку, роблячи процес відлагодження простішим (особливо коли мова йде про «давній» код).

У контексті сучасних умов розвитку України розробка прототипу також є актуальною - згадаймо події 2017 року, а саме – блокування російських інтернет-сервісів в Україні. Невдовзі після того як відповідний закон набув чинності, у мережі Інтернет з’явився стартап соціальної мережі Ukrainians. Ще під час просування соціальної мережі творці рекламували зокрема і тим, що розроблена вона канадськими програмістами, що суб’єктивно в корені неправильно беручи до уваги контекст розробки. Можливо так сталося через брак кваліфікованих спеціалістів на вітчизняному ринку праці, що негайно випливає у доцільність нарощення національної експертизи у відповідній області. Хоч проект і виглядав обнадійливо, мережа припинила своє функціонування 15 вересня 2017 року. Однією з вагомих причин невдачі стартапу безперечно є його незавершеність, оскільки довгий час були відсутні навіть базові функції, такі як обмін повідомленнями між користувачами, а новий функціонал з’являвся вкрай повільно, що знов-таки підсилює викладену вище думку про брак експертизи в області створення якісних соціальних мереж у стислі часові рамки. Порівнюючи Ukrainians та Sprout слід зазначити, що останній має повністю функціонуючий механізм пошуку користувачів та їх дописів, а також три режими рекомендації змісту для перегляду – функціонал покликаний зацікавити користувачів у використанні даного програмного рішення, а це те, чого бракувало проекту 2017 року.

Розділ 1. Опис вимог до застосунку

Усі вимоги щодо функціоналу застосунку розподілено між трьома групами:

- Обов’язкові
- Пріоритетні
- Факультативні

1.1 **Обов’язкові**

1. Новому користувачу повинна бути надана можливість зареєструватися, існуючому – аутентифікуватися у своєму обліковому записі для отримання доступу до функціоналу блогу.
2. Надати можливість користувачам відслідковувати останні дописи одне одного, при цьому дозволити користувачам самостійно довантажувати вміст сторінки з найсвіжішими публікаціями, щоб уникнути надмірного навантаження як на клієнтську частину так і на серверну.
3. Надати можливість публікувати власні дописи вказуючи при цьому назву та опис допису, надання опису не повинно бути обов’язковим.
4. Редагування та видалення опублікованих користувачем дописів, при цьому якщо допис було відредаговано, вказувати це при відображенні.
5. Можливість коментувати довільний допис, оцінювати його позитивно чи негативно, забезпечити довантаження старіших коментарів на вимогу.
6. На основі вподобань інших користувачів, які також позитивно оцінили поточний допис пропонувати даному користувачу рекомендовані дописи.
7. Кожен користувач повинен мати можливість кастомізувати свій обліковий запис змінивши аватар за замовчанням та додавши банер власної сторінки. Також повинні бути доступні поля для зазначення статусу користувача та його опису («про себе»).
8. Вкладки для показу підписників (ті, хто підписані) та підписок (ті, на кого підписані) кожного користувача.

9. Функціонал зміни паролю до облікового запису. Після зміни користувач повинен бути виписаний з облікового запису у поточній вкладці, а усі сторонні підключення мають бути визнані недійсними, їх доступ до перегляду та зміни даних застосунку відтак має бути обмежений. Вихід з облікового запису також повинен відбуватися після визначеного періоду часу або на вимогу користувача, якщо така відбулася раніше.
10. Надати функціонал пошуку інших користувачів та їх дописів. При демонстрації знайдених авторів показувати їх коротку статистику та останні дописи разом із банером (якщо такий наявний) та аватаром. При відображенні допису виводити коротку інформацію про кількість позитивних та негативних оцінок, а також коментарів до даної публікації. Користувач повинен мати змогу оцінити допис не відкриваючи його повний вид.
11. Реалізувати рекомендацію видавців спираючись на підписки користувачів, чиї вподобання частково збігаються з такими даного користувача.
12. Реалізувати рекомендацію дописів відповідно до того, які дописи вподобав даний користувач та інші користувачі з подібними смаками, брати до уваги антипатії користувача, зменшити ймовірність рекомендації потенційно небажаного допису.

1.2 **Пріоритетні**

1. Реєстрація нових користувачів повинна здійснюватися через підтвердження після переходу за підтверджувальним посиланням надісланим на вказану користувачем під час реєстрації пошту.
2. Показувати користувачу у якій якості буде відображено його фото після завантаження.
3. Забезпечити стискання завантажених фото задля пришвидшення роботи застосунку в умовах нестабільного та/або низько-швидкісного інтернет покриття в користувача.

4. Для вимірювання популярності користувачів надати статистичну інформацію про активність інших користувачів на сторінці даного. До метрик активності та популярності належать: кількість опублікованих дописів, кількість позитивних та негативних оцінок отриманих від інших користувачів загальна кількість коментарів написаних усіма користувачами до усіх дописів даного видавця, кількість користувачів, які підписані на даного.

1.3 **Факультативні**

1. Час активності реєстраційного посилання має бути обмежений у часі.
2. Забезпечити синхронізацію оцінювань, наприклад, якщо користувач з кількох вкладок або пристроїв та на одному оцінює певний допис, то при виставлені тієї ж оцінки з іншої неоновленої вкладки (пристрою) оцінка має зберігатися (не зніматися). Таким чином поведінка має залежати не від стану клієнтської частини, що може мати застарілу інформацію, а від серверної, яка гарантовано має найсвіжіші дані.
3. Надати функціонал адміністратора, який зможе видаляти дописи інших користувачів та банити їх облікові записи.

Усі обов’язкові та пріоритетні вимоги виконані, серед факультативних повністю виконані усі крім третьої. Функціонал адміністратора наданий на серверній частині, але відповідний UI на frontend частині відсутній.

Розділ 2. **Історія розвитку архітектури**

2.1 **Архітектура 1950-х років**

Перші програми виконувалися взагалі без операційних систем – вони були написані на асемблері, тому взаємодіяли із апаратною частиною напряму. Через це програміст мав усе робити сам (без використання програмних каркасів чи якої-небуть іншої допомоги).

Рисунок 1 - схематичне зображення архітектури застосунків 1950х років

2.2 **Архітектура 1960-х років**

З появою операційних системи з програміста було знято обов’язок займатися однотипними низькорівневими операціями, однак робота з пам’яттю та даними досі лишалася задачею програміста.

Рисунок 2-схематичне зображення архітектури застосунків 1960х років

2.3 **Архітектура 1970-х років**

У цей період здобули популярність системи керування базами даних, задача розробників спростилася, адже достатньо було лише декларативно вказувати як керувати даними. Основну роботу з отримання, сортування, збереження даних та файлів було абстраговано від програмістів за допомогою бібліотечних процедур.

Рисунок 3 - схематичне зображення архітектури застосунків 1970х років

Створення будь-якого застосунку завжди починається із планування, одним з ключових його етапів є вибір архітектури, яка визначає розміщення та спосіб комунікації між головними рівнями застосунку, а саме: представлення, бізнес-логіки, бази даних.

Подальший розвиток систем керування базами даних призвів до появи клієнт-серверної архітектури із типами «товстого» та «тонкого» клієнта залежно від обсягу роботи, який він виконує, це сприяло утворення рівневих архітектур про які далі йтиме мова.

Для створення Sprout розглядалися 4 види рівневих архітектур: однорівнева, дворівнева, трирівнева, багаторівнева. Нижче наведено їх порівняльну характеристику.

Розділ 3. Рівнева архітектура

3.1 Однорівнева

Рисунок 4 - схема однорівневої архітектури [7]

Є найпримітивнішою архітектурою, яка не передбачає розділення зазначених вище рівнів, натомість визначає їх сумісне використання на одній машині.

3.2 Дворівнева

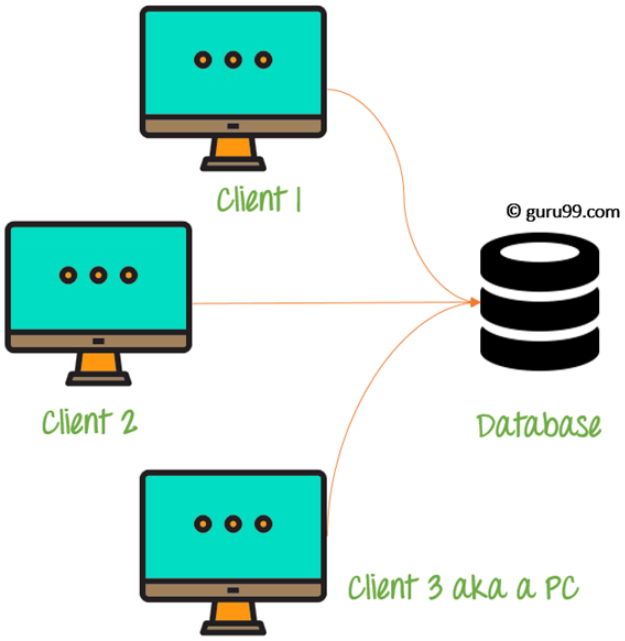

Рисунок 5 - схема дворівневої архітектури [7]

На відміну від однорівневої передбачає виділення рівня представлення та бізнес-логіки в один компонент, який працюючи на стороні клієнта звертається

до сервера бази даних для отримання даних. Таке використання уможливлює роботу багатьох користувачів зі спільними даними одночасно, але створює загрозу для безпеки даних, тому вимагає налаштування ролей та дозволів на рівні бази даних.

3.3 **Трирівнева**

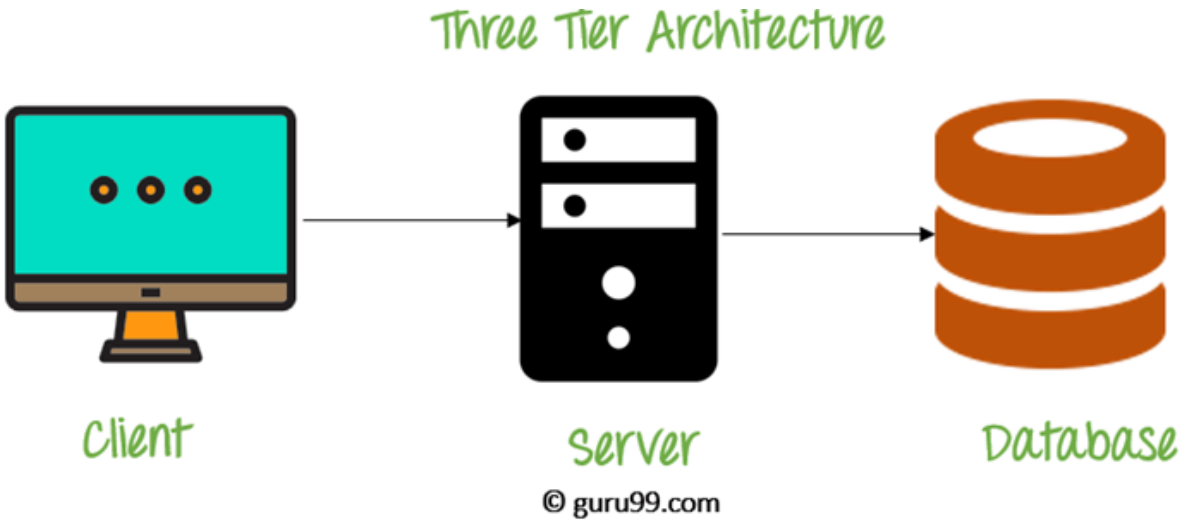

Рисунок 6 - схема трирівневої архітектури [7]

Передбачає повне розділення рівнів представлення, бізнес-логіки та бази даних, уможливлює їх роботу на трьох різних машинах. Як і у випадку із дворівневим аналогом спілкування відбувається через повідомлення, однак захистом даних опікується рівень бізнес-логіки розташованих на сервері, адже клієнтська частина має доступ лише до рівня представлення. Через це зникає необхідність у великій кількості налаштувань безпеки на рівні БД. Замість багатьох користувачів з різними правами маємо одного з найбільшою їх кількістю – рівень бізнес-логіки (для спрощення опустимо наявність інших користувачів, які виконують адміністративні функції).

3.4 **Багаторівнева**

Базуючись на ідеї трирівневої архітектури передбачає наявність додаткових рівнів, наприклад рівень агрегації, на якому можуть бути розташовані балансувальники навантаження, фаєрволи, кеш тощо.

Для виконання роботи було обрано трирівневу архітектуру, адже вона є класичним вибором для сучасних веб-застосунків. Багаторівневий аналог у масштабах Sprout видався надлишковим зокрема і через те, що застосунок працює безстаново (stateless), а тому не потребує наявності розподільника навантаження, який би підтримував сесію намагаючись направляти усі запити у межах сесії на один і той самий сервер[6].

Розділ 4. **Станові та безстанові застосунки**

Машинне масштабування довільного застосунку передбачає збільшення кількості серверів на яких буде розміщено копії даного застосунку та як результат підвищення обробляючої здатності системи.

Зазвичай робота користувача із програмним забезпеченням передбачає певний стан системи, адже наступні дії користувача залежать від інформації отриманої внаслідок виконання попередніх. Відповідно цей стан потрібно підтримувати, у випадку веб застосування це означає, що запити одного користувача повинен оброблювати один сервер, або повинно бути спільне сховище станів для усіх користувачів, до якого зможуть доступатися усі сервери. Такий спосіб роботи описує станозалежний підхід обробки користувацьких запитів, тобто такий, для правильної роботи якого потрібна не лише інформація передана одним запитом, а контекст створений попередніми зверненнями до сервера. Станонезалежні застосунки, однак, не мають такого обмеження і здатні опрацьовувати клієнтські запити автономно – необхідності зберігати дані з попередніх звернень. Відповідно, для станонезалежної взаємодії клієнту не обов’язково звертатися до того ж самого сервера для підтримання сесії. Така властивість уможливлює горизонтальне масштабування, яке має безліч переваг при створенні великої автоматизованої інформаційної системи.

Розділ 5. **Вертикальне та горизонтальне масштабування**

Традиційно у процесі використання успішні системи стають більш популярними, нарощують функціонал, та клієнтську базу, одним словом – ростуть. Відповідно збільшується навантаження на апаратне забезпечення на якому виконується програмна складова, тому рані чи пізно постає питання про розширення. Розрізняють 2 види масштабувань: вертикальне та горизонтальне, вони подібні тим, що передбачають нарощення обчислювальних ресурсів до інфраструктури, однак відрізняються тим як ця ціль досягається та який вплив має на продуктивність.

- Вертикальне масштабування – передбачає вдосконалення наявного сервера, наприклад встановленням більш потужного процесора або більшого обсягу оперативної пам’яті, швидшим сховищем тощо. Можлива також повна заміна старого сервера на новий із усіма покращеними характеристиками.
- Горизонтальне масштабування – передбачає встановлення додаткових серверів, які загально зможуть надати більше обчислювальних ресурсів за рахунок розподілення навантаження між ними.

Вертикальне масштабування обмежене продуктивністю самих компонентів, тобто продовжувати його можна до якоїсь міри, допоки існують більш потужне апаратне забезпечення та операційна система, яка здатна з ним працювати. Відповідно для амбітних проектів ріст яких не повинен обмежуватися вертикальне масштабування не є найкращим вибором, ба навіть більше, один сервер автоматично є єдиною точною відмови, яка у разі несправності припинить працездатність усього сервісу, системи. Теоретично, звичайно, можливо мати запасний сервер із дубльованими даними, але це вкрай не по-господарськи мати велику кількість дорогого обладнання, яке не повинно працювати лише для того, щоб підстраховувати робоче обладнання.

Однак, у вертикального масштабування є і перевага. Оскільки усі обчислення відбуваються на одній машині, відпадає необхідність спілкуватися з іншими серверами через мережу, а це суттєво підвищує продуктивність застосунку.

Горизонтальне масштабування наразі є вкрай популярним, тому для зручного та ефективного застосування цього методу розширення існує велика кількість надійних інструментів, підходів, таких як віртуалізація, контейнеризація, застосунків засобами Докера та Кубернетуса, наприклад. Детальний розгляд цих технологій виходить за межі дослідження даної курсової роботи, тому лише коротко зазначимо, що вони надають можливість ефективно та гнучко (еластично) керувати обчислювальними ресурсами та забезпечувати рівень надійності, що перевищує такий в окремих компонентах, а це вкрай важливо для сучасного бізнесу.

Недоліком горизонтального масштабування є підвищені накладні витрати на комунікацію між компонентами, синхронізацію, використання віртуалізації тощо. Така плата продуктивністю за розгортання застосунку горизонтально невідворотною і всіляко мінімізується за рахунок використання мінімалістичних контейнерів під керуванням різних модифікацій операційної системи Лінукс, тому є прийнятною.

Розділ 6. **Поняття високої доступності**

У гіперконкурентних умовах сучасного бізнесу важливим є кожен клієнт, адже якщо певна проблема відштовхнула одного користувача, скоріш за все вона стане причиною відмови інших потенційних користувачів від сервісів компанії. Користувацький досвід відіграє важливу роль у виборі потенційних клієнтів, а доступність як один з основних аспектів цього враження від продукту є базовою вимогою до системи, тому не може бути скомпрометований.

Розрізняють два види обчислення доступності:

1. «на основі співвідношення часу працездатності (uptime) та непрацездатності (downtime)» [8]

$$\text{доступність} = \frac{\text{час безвідмовної роботи}}{\text{увесь час}}$$

2. «на основі оцінки відсотка успішних звернень» [8]

$$\text{доступність} = \frac{\text{кількість успішних звернень}}{\text{загальна кількість звернень}}$$

Засобом надання високої доступності є кластеризація, вона надає можливість керувати множиною серверів як одним цілим, запобігти втраті працездатності системи у разі виходу з ладу одного з компонентів (завдяки надлишковості). Зазвичай, коли кажуть про кластеризацію, мають на увазі горизонтальне масштабування, у такому разі продуктивність кластера також залежить від кількості та потужності його компонентів і природньо такий кластер надає можливість спільно виконувати застосунок на кількох серверах.

Розділ 7. **Моделі кластеризації**

Розгляньмо дві моделі кластеризації:

- відповідно до розподілу активності між вузлами
  - ❖ Модель «активний-резервний» - передбачає наявність додаткових компонентів, які не перебувають в активному стані допоки справно працюють ті пристрої, обчислювальні можливості яких покривають вимоги клієнта щодо потужності. У разі якщо якийсь компонент виходить з ладу, його заміщує резервний, таким чином рівень продуктивності лишається стабільним, однак оскільки резервні пристрої простоюють доки працюють активні, таке використання обладнання є неефективним.
  - ❖ Модель «активний- активний» - передбачає одночасну роботу усіх виділених під кластер пристроїв, у разі виходу з ладу якогось компоненту, як і в попередньому випадку, працездатність кластеру не припиняється – вона знижується відповідно до кількості та потужності компонентів, що перейшли у неактивний стан.
- відповідно до розподілу даних.
  - ❖ Модель зі спільними даними – передбачає використання спільного сховища даних між усіма серверами, оскільки усі дані зберігаються в одному місці, легко досягти узгодженості даних. Якщо один із серверів вийде з ладу, то дані втрачено не буде, але для цього накладається вимога високої надійності на саме сховище даних, адже у разі його виходу з ладу робота усього кластеру стане неможливою. Висока зв’язаність усіх серверів з єдиним компонентом вимагає близьке географічне розташування усіх компонентів.
  - ❖ Модель з індивідуальними даними – передбачає наявність в кожного сервера власного сховища даних, це дозволяє географічно розподілити компоненти кластера, але також унеможливлює продовження повноцінної роботи кластеру, адже частина даних, яка зберігалася на сервері, який вийшов з ладу буде також втрачена.

Кожна модель має свої переваги і недоліки, тому на практиці прийнято використовувати комбіновану модуль. Залежно від того як часто використовуються дані їх обробляють у вузлах з індивідуальним чи спільним сховищем. Для запобігання втрати даних у випадку відмови сховища використовують резервне копіювання.

Розділ 8. **Порівняння Spring Boot та Java EE**

Java є однією з перших мова програмування, створених спеціально для розробки бізнес-застосунків. Відтак шлях її розвитку був тернистим і загалом відбувався «методом спроб та помилок». Наприкінці минулого століття розробники відчували гостру нестачу в інструменті, який би дозволив спростити процес написання коду, дозволи в би уникнути дублювання одноманітних фрагментів, які шаблонно повторюються на більшості проектів.

Першою спробою задовольнити цей попит стала Jakarta Enterprise Beans (EJB). Перша специфікація EJB була розроблена IBM у 1997 і була адаптована Sun Microsystems у 1999 році у стандартах EJB 1.0 та EJB 1.1. За своєю суттю ця подія була певною мірою революційною, адже це був перший стандартизований інструмент, який пропонував у готовому вигляді такий функціонал як масштабованість та авторизацію. На жаль, перша спроба виявилася вкрай незручна для використання і викликала хвилю розчарування у розробників, які вирішили спробувати її використати. Все ж прогрес на цьому не зупинився, перша невдача дала неоціненний досвід у розробці подібних інструментів покликаних поліпшити процес розробки великих програм рівня підприємства. Почали з’являтися і зникати аналоги Java EE, одним з цих аналогів був Spring розроблений у 2004 році командою Pivotal. Саме своєю простотою та компактністю він завоював популярність у двохтисячних, що дозволило йому не лише вижити, а і поступово розвиватися. Інверсія елементів управління запропонована у даному програмному каркасі навіть була додана до Java EE, однак це не змінило ситуацію для останньої. Поява Spring Boot, який дозволив спростити налаштування Spring сприяв подальшій популяризації останнього. Станом на п’яте лютого дві тисячі двадцятого року «Spring dominates the Java ecosystem with 60% using it for their main applications»[9].

За останнє десятиліття обидві технології дуже розвинулися, якщо звернутися до офіційного визначення того, що таке JavaEE у дві тисячі двадцять першому

році, то побачимо: «Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals.»[10]. Якщо ж порівняти його з визначенням Spring: «Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications»[11] та спрощеним описом Spring: «Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".»[12], то можна побачити, що за великим рахунком обоє ставлять перед собою одну мету – спростити розробку великих бізнес-застосувань, однак Spring Boot більше спрямований на мікро сервісну архітектуру.

Розділ 9. Опис серверної частини застосунку

9.1 Опис класів та інтерфейсів

У корінному каталозі знаходиться клас BlogApplication, з якого стартує застосунок. Офіційна документація Spring Boot зазначає: «When a class does not include a package declaration, it is considered to be in the “default package”. The use of the “default package” is generally discouraged and should be avoided. It can cause particular problems for Spring Boot applications that use the @ComponentScan, @ConfigurationPropertiesScan, @EntityScan, or @SpringBootApplication annotations, since every class from every jar is read.»[1]. З цієї причини вищезгаданий клас знаходиться у пакеті edu.ukma.blog. Клас BlogApplication є конфігураційним, у ньому частково налаштовуються веб та сек’юріті частини, вказуються базові класи для пошуку сутностей та репозиторіїв для роботи з ними, такий підхід є схвальним і пропонується документацією програмного каркасу: «Usually the class that defines the main method is a good candidate as the primary @Configuration.»[2]. Використання типо-безпечних (type safe) варіацій анотацій EnableJpaRepositories та EntityScan зумовило створення додаткових маркерних інтерфейсів IRelationalReposPackage, LoggedOutUserRepo, IModelsPackage та LoginResponse. Детальніше пакети в яких знаходяться дані інтерфейси та класи будуть описані далі.

Пакет constants.

Містить класи ImageConstants, PatternConstants, ValidationConstants. Це прості класи, які містять деякі константи пов’язані з опрацюванням завантажених користувачем картинок, регулярними виразами для валідації користувацьких імен та паролів та перевірки довжини текстового опису завантажуваних дописів відповідно.

Пакет controllers

Містить у собі такі підпакети:

- blog_features – контролери, які надають доступ до власне сервісу блогу – дайжесту, рекомендацій та пошуку.
- communication – контролери, які уможливлюють комунікацію між користувачами, а саме: коментування та оцінювання дописів, підписка на та відписка від інших видавців, публікація дописів та керування їх станом.
- user – контролери, які слугують для роботи пов’язаної безпосередньо із обліковим записом користувача, надають доступ до зміни банера та аватара користувача, текстового самоопису. Клас AdminCtrl надає функціонал видалення дописів, блокування та розблокування користувачів, створення нових адмінів.

Пакет exceptions.

Містить ExceptionHandler’и для очікуваних виключних ситуацій, усі вони доволі стандартні, тому зупинятися на них немає особливого сенсу, проте роль яку вони відграють під час відлагодження та експлуатації застосунку переоцінити складно.

Пакет models

Складається із пакетів comment, composite_id, record, simple_interaction, user та виділеного інтерфейсу IModelsPackage. Пакети record та comment схожі і прості за своїм вмістом – Entity клас та DTO для отриманна та надсилання відповідей. Composite_id, як каже назва, містить класи, що відповідають складеним первинним ключам деяких сутностей. Simple_interaction крім стандартних реляційних сутностей Evaluation та Follower також містить підпакет із графовими сутностями RecordGraphEntity та UserGraphEntity які проектуються у відповідні графові сутності та використовуються для підбору рекомендацій. User подібний до comment та record, але на додачу до аналогів вказаних класів містить підпакет authorization, в ньому зберігаються класи та переліки (enum),

які відповідають за розподіл ролей та дозволів користувачів (простих та адміністраторів). Адміністратор хоч і може видаляти дописи та банити користувачів, не є супер користувачем, адже не має права оцінювати інших користувачів та підписуватися на них.

Пакет repositories

Для зручності поділений на графову та реляційну складові. Також цей пакет містить інтерфейси-проекції, які дозволяють оптимізувати швидкодію виконання запитів уникаючи повернення надлишкових даних: «The goal of a DTO class is to provide an efficient and strongly typed representation of the data returned by your query. To achieve that, a DTO class typically only defines a set of attributes, getter and setter methods for each of them, and a constructor that sets all attributes.»[3] Однак такий підхід передбачає написання суб'єктивно надлишкового коду, тому було вирішено використати альтернативу: «Instead of defining a class with an all arguments constructor, you can also use an interface as your DTO projection. As long as your interface only defines getter methods for basic attributes, this is identical to the projection I showed you before.»[3]

Пакет security.

Містить власні підпакети models, repositories, services, чий функціонал спрямовано виключно на формування безпечних умов використання застосунку. Автентифікація відбувається завдяки json web token механізму, що забезпечує stateless функціонування застосунку, перевагами цього є потенційна висока горизонтальна масштабованість.

Пакет services.

Містить усі сервіси спрямовані на надання функціоналу бізнес логіки. Тут не розділяються окремо сервіси для роботи з графовими та реляційними репозиторіями - вони використовуються сумісно. Для зручної навігації у корені

даного пакету розміщено 2 підпакети – interfaces та implementations, їх внутрішня стурктура повністю збігається, але дає можливість швидко ознайомитися із API який задають та реалізують інтерфейси. Застосування принципу Dependency inversion реалізується за допомогою анотацій @Service розміщених над конкретними імплементаціями відповідних інтерфейсів.

Пакет utils.

Має у собі допоміжні класи, які використовуються сервісами та контролерами для обробки зображень, побудови дерева підпапок, у яких зберігатимуться зображення та упакування результатів опрацювання запиту в узагальнену обгортку пагінації.

9.2 Діаграма класів

Оскільки проект містить велику кількість модулів та класів для кращого загального сприйняття діаграми класів було вирішено розділити її на 4 частини:

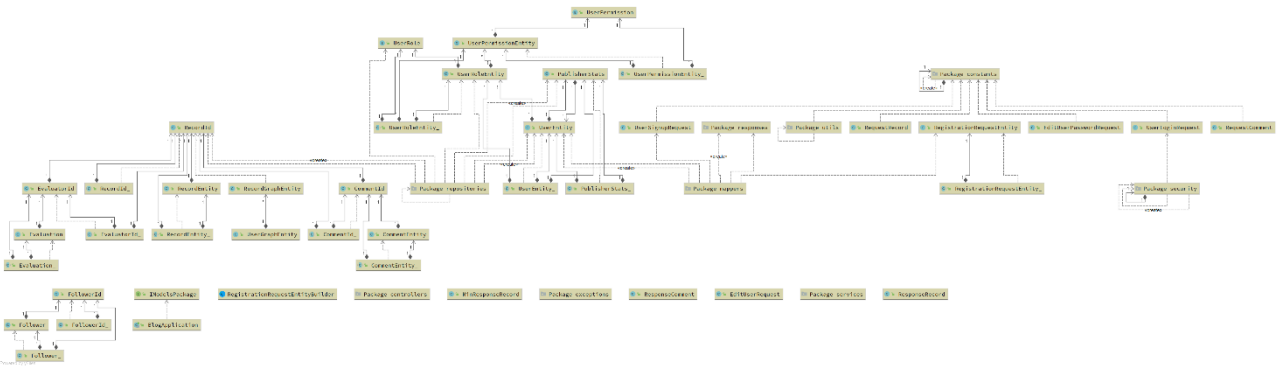

Рисунок 7 - діаграма моделей

Класи-сутності відповідають таблицям у реляційній базі даних і мають слово «Entity» наприкінці своєї назви. Також тут зображені класи для створення об’єктів перенесення даних, вони мають «Request» або «Response» як частину їх назви. До числа моделей також належать класи-сутності які відповідають

графовим сутностям, вони окрім слова «Entity» мають у своїй назві також слово «Graph».

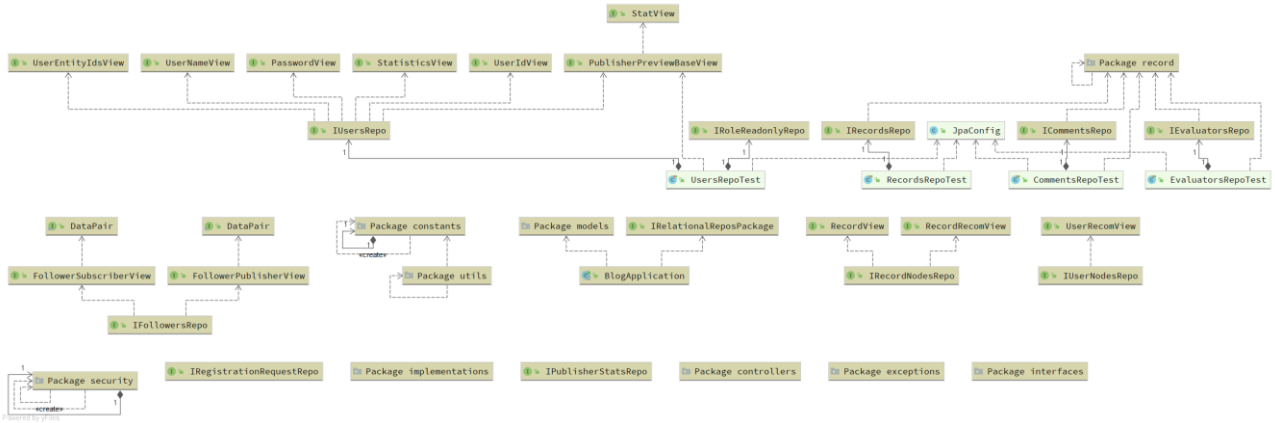

Рисунок 8 - діаграма репозиторіїв

Як і у випадку із класами-сутностям, репозиторії також є «реляційні» та «графові».

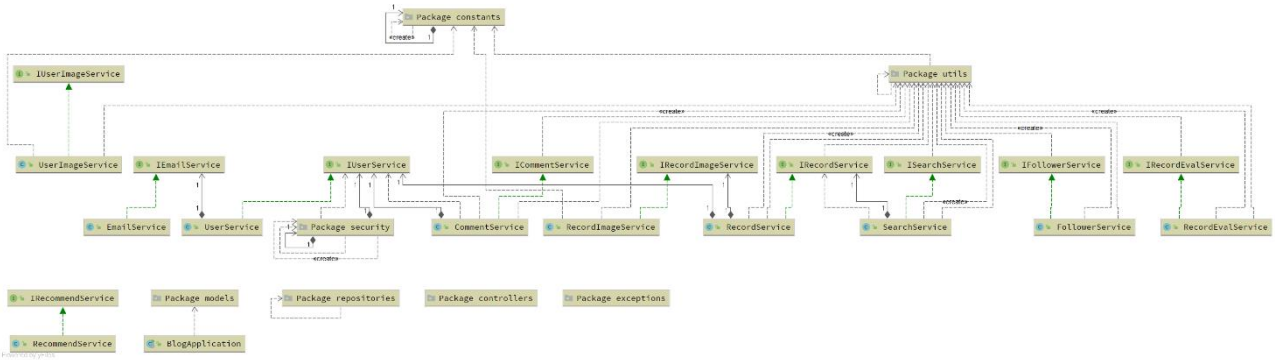

Рисунок 9 - діаграма сервісів

Оскільки рівень сервісів є проміжним між контролерами та репозиторіями, він має найбільшу кількість зв’язків. Деякі сервіси використовуються у багатьох контролерах, крім цього здебільшого кожен сервіс використовує кілька репозиторіїв.

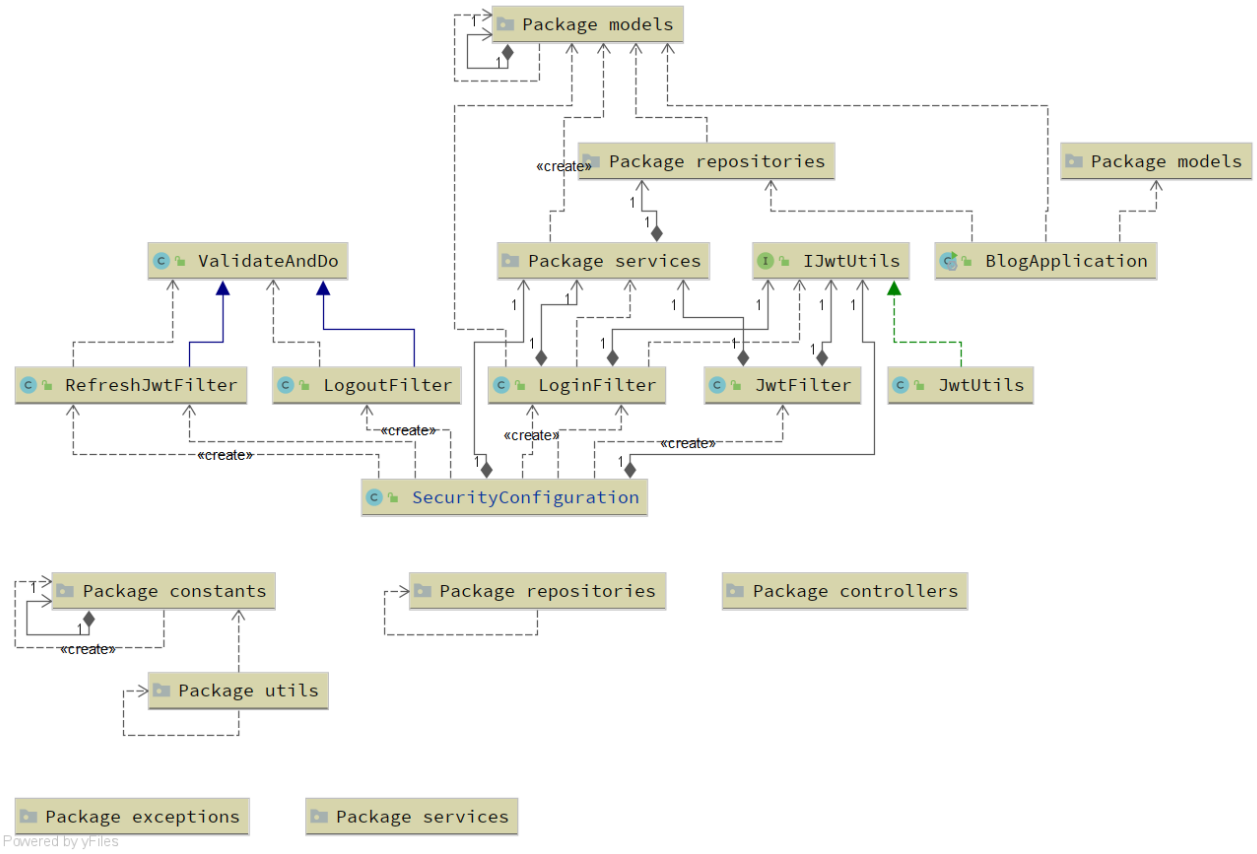

Рисунок 10 - діагарама класів, які відповідають за безпеку використання застосунку

Для авторизації користувача необхідно отримати деталі про нього, для цього AuthenticationManager використовує імплементацію UserDetailsService. IUserService, який знаходиться у пакеті сервісів, розширює даний інтерфейс, а відтак надає імплементацію, яка і використовується LoginFilter.

9.3 ER-діаграма

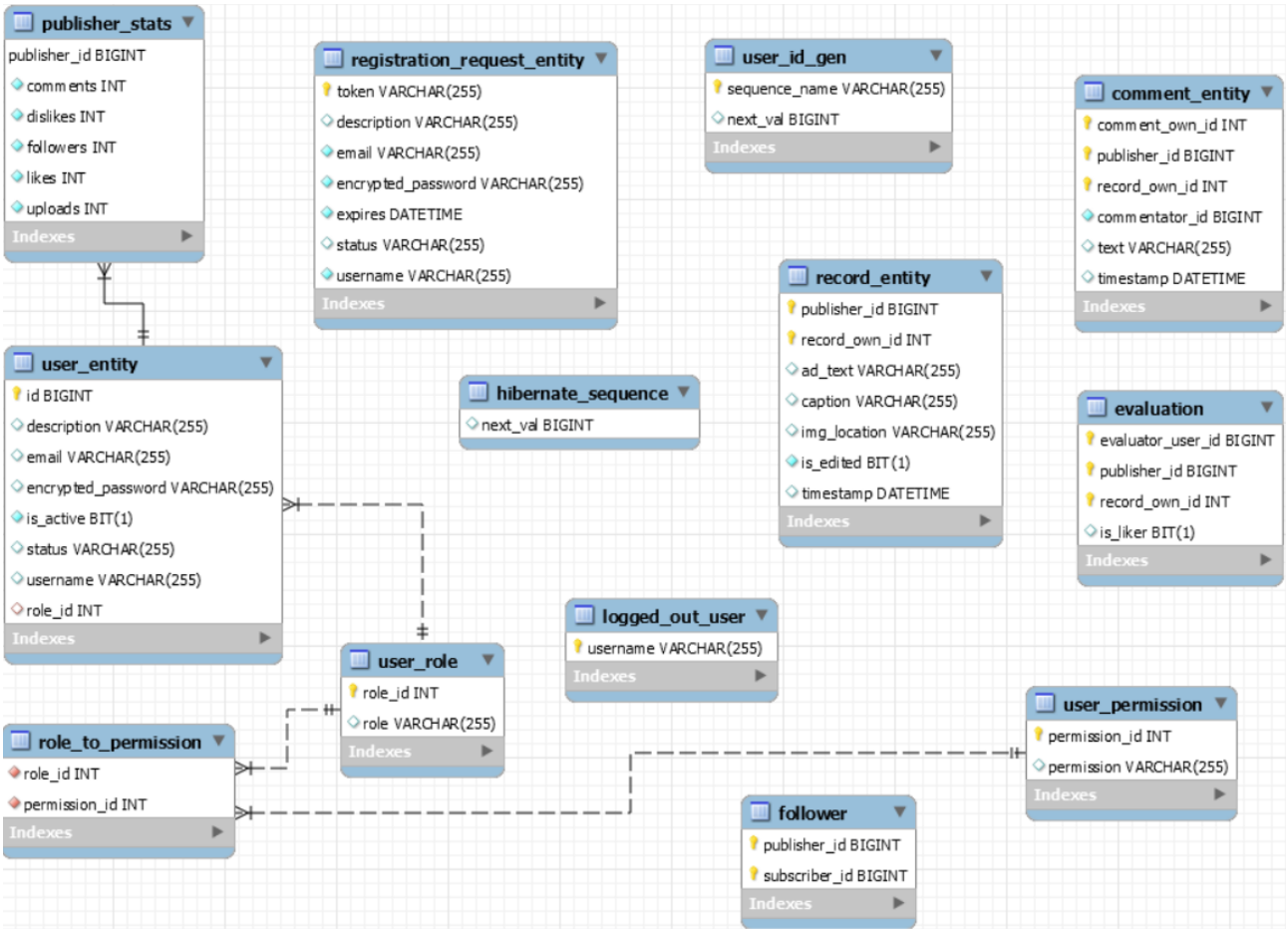

Рисунок 11 - згенерована ER-діаграма сутностей

9.4 Проблеми та способи їх розв'язання

Для роботи з даними здебільшого використовувався JPA, який хоч і надає гнучкий функціонал за допомогою query method'ів, все ж не є універсальним, тому для виконання деяких запитів, таких як getCommentsNumForRecords, getRecordsEvaluations, getLastRecordsOfPublishers було вирішено застосувати нативні запити до СКБД MySql.

Оновлення персональних даних приходить в єдиній формі і може містити, або не містити певні поля. Відповідно, для того, щоб передбачити усі можливі комбінації потрібно було б написати 2^n запитів до бази даних. Щоб цього уникнути було застосовано JPA Criteria API Queries. Для малого набору цей

підхід очевидно не робить великої різниці, але зі зростанням проекту доцільність його використання лише зростала б.

Генерація добірки рекомендацій з використанням SQL баз даних передбачає кількаразове виконання операції JOIN, що невідворотньо завдасть удару по швидкодії застосунку зважаючи на частоту генерації рекомендацій. Для того, щоб вирішити цю проблему було використано графову базу даний Neo4j. Новина з блогу цієї СКБД показує доцільність використання зазначеного інструменту: «In their new book, Neo4j In Action, Jonas Partner and Aleksa Vukotic perform a fascinating experiment. In theory, a graph database should be much faster than a relational databases in graph traversal. For example, in a social network, finding all the friends of a user’s friends. Even more so for friends of friends of friends. Aleksa and Jonas built this query in both MySQL and Neo4j with a database of 1,000,000 users and the results are striking. Execution Time is in seconds, for 1,000 users. For the simple friends of friends query, Neo4j is 60% faster than MySQL. For friends of friends of friends, Neo is 180 times faster. And for the depth four query, Neo4j is 1,135 times faster. And MySQL just chokes on the depth 5 query.»[4].

Розділ 10. Опис клієнтської частини застосунку

10.1 Опис компонентів frontend частини

Frontend частина знаходиться у виділеній однойменній директорії на рівні з java відповідником. Традиційно для ReactJS застосунку основна частина написаного розробником коду знаходиться у src підпакеті, далі розглянуто його вміст.

Пакет assets

Містить зображення за замовчанням для неперсоналізованого користувача та системні іконки, наприклад для пошуку користувачів та дописів.

Пакет auth

Має компоненти які займаються створенням безпечних умов використання frontend частини та опікуються наданням доступу до персональних даних з облікових записів користувачів. Перш ніж користувач аутентифікується у системі йому заборонено доступ до будь-яких даних. При спробі перейти за посиланням, яке підтвердженому користувачу показує шуканий вміст незареєстрований користувач побачить пропозицію зареєструватися. Такий функціонал став можливим завдяки використанню бібліотеки React Router.

Пакет cms_backbone

Містить каркас сайту, тут знаходиться компонент CMSNavbarRouting, який є контейнером для усіх змістовних компонентів, а також компоненти Header, Footer та інші допоміжні компоненти, які разом дійсно вибудовують каркас для типової сторінки аутентифікованого користувача. Більшість компонентів (крім вкладки користувача) відображаються незалежно від стану застосунку.

Пакет digest

Містить компоненти покликані дати користувачу перше враження про вміст системи, вподобання та ідейні спрямування видавців. Усі дописи впорядковано з датою публікації у порядку від найсвіжіших до найдавніших. Певна річ з міркувань швидкодії та елементарно роботоспроможності дописи не завантажуються усі одночасно, для цього передбачено кнопку довантаження «на вимогу». Оскільки кожен користувач переходячи на сторінку з нещодавніми дописами бачить той самий вміст, що і інші, існує простір для оптимізації серверної частини застосунку, а саме – створення кешу. Клік на довільний ескіз спрямовує користувача на сторінку із відображенням повної версії допису, де йому надається розширений функціонал.

Пакет expose_publisher

Даний пакет є складеним і містить у собі такі підпакети:

- full_publisher_page – містить єдиний компонент PublisherMainPage, який відображає вміст сторінки користувача. Тут показується статистика видавця, його банер (тут і лише тут) та власне опубліковані дописи. Якщо вміст надто багато, щоб відобразити його на 1 сторінці, пагінація вкінці сторінки полегшує використання застосунку. Розмір сторінки визначається не серверній частині, у конфігураційному файлі. Таким чином недоброчесний користувач не зможе завантажити одразу увесь вміст якогось потенційно активно видавця і тим самим надмірно навантажити систему. Разом із цим адміністратор зможе еластично змінювати розмір сторінки без потреби шукати відповідні налаштування у самому коді застосунку.
- multiple_publishers – містить компоненти для відображення сторінки користувачів. Такий функціонал потрібен у випадках, коли використовується функція пошуку, при показі рекомендованих видавців, або коли користувач прагне побачити підписки або підписників іншого користувача.
- user_profile_details – компоненти для зміни особистих даних користувача, таких як пароль, аватар, банер, статус, «про себе» а також компонента статистики, яка повторно використовується не лише при показі головної сторінки користувача і при демонстрації його допису.

Пакет expose_record

Має аналогічну мету, що і вищеописаний expose_publisher, складається з двох підпакетів: multiple_records, single_record. Задача компонентів, що містяться в них теж інтуїтивно зрозуміла – забезпечення функціоналу перегляду та взаємодії з дописами: показ коментарів, редагування дописів, їх оцінювання, відображення контекстних рекомендацій.

Пакет utils

Містить файли з функціями, які складно віднести до якогось конкретного класу, але які використовуються кількома компонентами у рівних долях. На відміну від решти компонентів ці файли мають розширення «.ts», а не «.tsx», адже ніякі ReactJS компоненти чи їх частини тут не створюються.

10.2 Проблеми та способи їх розв’язання

Основною проблемою frontend частини став її дизайн, більш точно – верстка. Оскільки використання застосунку з графічним інтерфейсом створеним за допомогою самого лише HTML5 є доволі складним та таким, що погіршує користувацький досвід, а також через специфіку виконуваної роботи, питання про необхідність використання CSS отримало однозначну позитивну відповідь. Через це мало не в кожному пакеті разом із ReactJS файлами-компонентами знаходяться .css файли зі стилями, які застосовуються у даному пакеті. Для полегшення стилізації та пришвидшення виконання роботи було вирішено використати React-Bootstrap, адже відповідно до опису цієї бібліотеки на офіційному сайті розробника вона ідеально пасує для розробки застосунків подібних до даного: «React-Bootstrap replaces the Bootstrap JavaScript. Each component has been built from scratch as a true React component, without unneeded dependencies like jQuery.»[5].

Здебільшого підходи продемонстровані в документації були прийняті та використанні при написанні коду Sprout, однак в питанні використання синтаксису для позначення контейнерів було вирішено відмовитися від використання тегів <Container> та <Row> із додатковим вказанням кількості колонок у якості окремого атрибуту на користь більш ручного написання класів, адже суб’єктивно це підвищує читабельність коду, оскільки в такому випадку усі стилі зосереджено в одному місці, відтак ними легше керувати і тримати під контролем.

Для більшості простих випадків функціоналу React-Bootstrap вистачало, однак для нетривіальних розташувань елементів доводилося використовувати flex-box вручну і прописувати стилі власноруч.

Ще однією проблемою було оновлення аутентифікаційного токену, адже якщо користувач на тривалий проміжок часу залишить застосунок без дій, токен втратить свою цінність, адже оновлюється він у кореневому компоненті і спочатку передавався у синівські компоненти при їх створенні. Зрештою було вирішено передавати не самі токени, а callback-функцію, яка повертає чинний токен, який регулярно оновлюється кореневою компонентою.

Висновки

Внаслідок виконання даної курсової роботи дослідили такі технології розробки веб-застосунків як Spring Data, Spring Web, Spring Security, Spring Boot framework, ReactJS, ReactBootstrap та інші. Набули практичних навичок написання слабо зв’язаного коду, роботи з базою даних MySql, ORM Hibernate та JPA. Ознайомилися із механізмом відлагодження та перехоплення помилок, які пропонує Spring, оцінили переваги розгорнутих повідомлень про виключні ситуації, які трапляються протягом роботи застосунку внаслідок помилок розробника або неправильних переданих користувачем даних. У процесі зневадження клієнтської частини застосунку випробували вбудований у Firefox відлагоджувача JavaScript коду, здобули досвід використання засобів розробника вбудованих у браузер. Маючи досвід використання мови програмування JavaScript оцінили переваги застосування TypeScript для швидшої та надійнішої розробки клієнтської частини.

Внаслідок історичної розвідки присвяченої етапам розвитку індустрії інформаційних технологій прослідкували за зміною ролі розробника при створенні автоматизованих інформаційних систем рівня підприємства, прослідкували зміни доступних для розробника інструментів та розглянули їх вплив на архітектуру відповідного програмного забезпечення, усвідомили причинно-наслідкові зв’язки між зростанням обчислювальних потужностей апаратного забезпечення, зростанням попиту на програмні продукти та появою нових підходів до гнучкого керування великою кількістю об’єднаних у кластер серверів, порівняли найбільш поширені методи збереження та обробки користувацьких даних, розглянули методи забезпечення високої надійності та причини з яких ця вимога є критичною для сучасного бізнесу.

Розглянувши історію розвитку JavaEE як першого стандартизованого інструменту для розробки великих веб-застосувань та проаналізувавши проблеми які мала перша версія EJB з’ясували причини зростання Spring на ранніх етапах, що дозволило цій компактній на початкових етапах бібліотеці згодом розростися до повноцінного каркасу програмного забезпечення та фактично витіснити старіший аналог.

Розробка самого застосунку почалась із проектування веб фреймів, через зміну бачення дизайну веб сайту цей етап доводилося кілька разів починати заново. Спираючись на функціонал, який передбачали веб фрейми було розроблено API серверної частини Sprout. Виходячи з потреб контролерів які відповідали точкам доступу до програмного інтерфейсу (endpoint) було розроблено інтерфейси сервісів конкретні імплементації яких у подальшому вставлялися у контролери за допомогою IoC функціоналу Spring. У процесі створення класів-реалізацій отриманих інтерфейсів сервісів ставало очевидно які специфічні (тобто такі, що їх не надавав JpaRepository за замовчанням) способи обробки

даних потрібні, залежно від складності створювалися методи-запити (query method) або використовувалися рідні запити (native query). Для реалізації функціоналу рекомендацій використовувалася графова база даних Neo4j, її підтримка Spring хоч і наявна, все ж на момент розробки застосунку була не така потужна як аналогічне рішення для реляційних баз даних, тому звернення до неї відбувалися переважно за допомогою рідних запитів. Під час розгортання застосунку на Heroku виявилося, що остання версія Neo4j яка була окремо розгорнута на GCP не підтримувала старий синтаксис підстановки параметрів у запити Cypher, тому довелося провести оновлення (refactoring) графових репозиторіїв, використавши новий синтаксис, який підтримувався як віддаленою версією на GCP так і встановленою локально. Для тестування серверної частини здебільшого використовувався Postman, а для перевірки остаточної версії застосунку були залучені фахівці з факультету соціальних наук та соціальних технологій.

Список використаних джерел

1. Spring Boot Reference Documentation: [Веб-сайт]. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-structuring-your-code (дата звернення: 21.04.2021).
2. Spring Boot Reference Documentation: [Веб-сайт]. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-configuration-classes (дата звернення: 21.04.2021).
3. Spring Data JPA: Query Projections - Thorben Janssen: [Веб-сайт]. URL: https://thorben-janssen.com/spring-data-jpa-query-projections/ (дата звернення: 21.04.2021).
4. How much faster is a graph database, really?: [Веб-сайт]. URL: https://neo4j.com/news/how-much-faster-is-a-graph-database-really/ (дата звернення: 21.04.2021).
5. React-Bootstrap · React-Bootstrap Documentation: [Веб-сайт]. URL: https://react-bootstrap.github.io/ (дата звернення: 21.04.2021).
6. Techniques for managing state - IBM: [Веб-сайт]. URL: https://www.ibm.com/docs/en/was-nd/9.0.5?topic=management-techniques-managing-state (дата звернення: 21.04.2021).
7. Database Architecture in DBMS: 1-Tier, 2-Tier and 3-Tier: [Веб-сайт]. URL: https://www.guru99.com/dbms-architecture.html (дата звернення: 21.04.2021).
8. Черкасов Д. І. Технології сучасних датацентрів // Тема 4. Висока доступність в датацентрі. URL: https://distedu.ukma.edu.ua/pluginfile.php/127631/mod_resource/content/1/ТСДЦ. Тема 4. Висока доступність в датацентрі.pdf (дата звернення: 21.04.2021).
9. Spring dominates the Java ecosystem with 60% using it for their main applications: [Веб-сайт]. URL: https://snyk.io/blog/spring-dominates-the-java-

ecosystem-with-60-using-it-for-their-main-applications/ (дата звернення: 21.04.2021).

10. Java Platform, Enterprise Edition (Java EE) | Oracle: [Веб-сайт]. URL: https://www.oracle.com/java/technologies/java-ee-glance.html (дата звернення: 21.04.2021).
11. Spring Boot - Introduction: [Веб-сайт]. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm (дата звернення: 21.04.2021).
12. Spring Boot: [Веб-сайт]. URL: https://spring.io/projects/spring-boot (дата звернення: 21.04.2021).