Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

\_Навчально науковий інститут інформаційних технологій та робототехніки\_

(повна назва факультету)

\_Кафедра комп’ютерних та інформаційних технологій і систем\_

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проекту (роботи)**

\_\_\_\_\_\_\_\_\_\_\_\_\_бакалавра\_\_\_\_\_\_\_\_\_\_\_\_\_

(освітньо-кваліфікаційний рівень)

на тему

Автоматизація процесу лідогенерації для маркетингу за допомогою CRM платформи

Виконав: студент 4 курсу, групи 401-ТК

спеціальності

\_\_123 Комп’ютерна інженерія \_

(шифр і назва напряму)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шахно В.О\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_\_\_\_Ляхов О.Л \_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Полтава – 2021 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА**

**спеціальність 123 «Комп’ютерна інженерія»**

**на тему**

**«Автоматизація процесу лідогенерації для маркетингу за допомогою CRM платформи»**

**Студента групи 401-ТК Шахна Владислава Олександровича**

Керівник роботи

доктор технічних наук,

професор Ляхов О.Л.

Консультант

старший викладач

Черницька І.О.

Завідувач кафедри

кандидат технічних наук,

доцент Головко Г.В.

Полтава – 2021

# РЕФЕРАТ

Кваліфікаційна робота бакалавра: 70 с., 14 рисунків, 2 таблиці, 1 додатку, 11 джерел.

**Об’єкт дослідження**: модель Web платформи для автоматизації процесу лідогенерації в маркетинговій сфері.

**Мета роботи**: розробка Web додатку автоматизованої системи управління даними та процесом їх збору та обробки, що дозволить значно прискорити цей процес.

**Методи**: проектування та розробка бази даних для автоматизованої системи управління даними в PostgreSQL, створення CRUD архітектури, розробка інтерфейсу для користувача з використанням технологій фреймворку Spring.

**Ключові слова**: CRUD, java, модель, база даних, лідогенерація, лід.

# SUMMERY

Bachelor's qualification work: 70 p., 14 pictures, 2 tables, 1 appendix, 11 sources.

**Object of research:** model of Web platform for automation of the process of lead generation in the marketing industry.

**Purpose:** development of a Web application for an automated data management system and the process of their collection and processing, which will significantly speed up this process.

**Methods**: design and development of a database for an automated data management system in PostgreSQL, creation of a CRUD architecture, development of a user interface using Spring framework technologies.

**Keywords**: CRUD, java, model, database, lead generation, lead.

**ЗМІСТ**

[**РЕФЕРАТ 2**](#_Toc74754022)

[**SUMMERY 3**](#_Toc74754023)

[**ВСТУП 6**](#_Toc74754024)

[**РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 9**](#_Toc74754025)

[1.1 Опис предметної області 9](#_Toc74754026)

[1.1.1 Структура мережі інтернет 10](#_Toc74754027)

[1.1.2 Принцип роботи та структура Web-додатків 11](#_Toc74754028)

[1.2 Переваги та недоліки Web-додатків 14](#_Toc74754029)

[1.2.1 Сучасний стан та перспективи 15](#_Toc74754030)

[1.3 Огляд наявних програмних рішень 16](#_Toc74754031)

[**РОЗДІЛ 2 ПРОЕКТУВАННЯ WEB-ДОДАТКУ 20**](#_Toc74754032)

[2.1 Структура та технології реалізації 20](#_Toc74754033)

[2.2 Проектування архітектури додатку 29](#_Toc74754034)

[2.2.1 Шаблони архітектури програмного забезпечення 30](#_Toc74754035)

[2.3 Проектування бази даних 33](#_Toc74754036)

[**РОЗДІЛ 3 РОЗРОБКА CRUD WEB-ДОДАТКУ 35**](#_Toc74754037)

[3.1 Вибір платформи 35](#_Toc74754038)

[3.2 Розробка бази даних 38](#_Toc74754039)

[3.3 Розробка користувацького інтерфейсу 40](#_Toc74754040)

[**РОЗДІЛ 4 ТЕСТУВАННЯ 47**](#_Toc74754041)

[4.1 Вибір виду тестування 47](#_Toc74754042)

[4.2 Тест план 47](#_Toc74754043)

[**ВИСНОВКИ 51**](#_Toc74754044)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52**](#_Toc74754045)

[**ДОДАТОК А ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ 53**](#_Toc74754046)

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ**

**ERP** – Enterprise Resource Planning

**UML** – Unified Modeling Language

**HTTP** – HyperText Transfer Protocol

**SQL** – Structured Query Language

**XAML** – eXtensible Application Markup Language

**JDBC** – Java Database Connectivity

**MVC** – Model-View-Controller

**ORM** – Object-Relational Mapping

**POJO-класи** – Plain Old Java Objects – прості Java-об’єкти

**JSP** – JavaServer Pages

**JavaEE** – Java Platform, Enterprise Edition

**DS** – DispatcherServlet

**GPL** – General Public License

**ПЗ** – програмне забезпечення

**ОС** – операційна система

**БД** – база даних

**СУБД** – Система управління базами даних

# ВСТУП

Перехід до цифрової економіки – це необхідна вимога сьогоднішнього дня. Діджиталізація економіки дає можливість людині полегшити вирішення багатьох завдань, пов'язаних з роботою, з пошуком інформації, з якими він неодноразово стикається. Діджиталізація відкриває для людини широкі можливості в розвитку бізнесу. Особливе значення тут набувають комунікативні можливості цифрових каналів. Велика швидкість, зручність дозволило появи такого виду маркетингу, як digital-маркетинг.

Цифровий маркетинг (англ. Digital marketing, Діджитал-маркетинг) – загальний термін, який використовується для позначення таргетованого та інтерактивного маркетингу, товарів і послуг, що використовує цифрові технології для залучення потенційних клієнтів і утримання їх в якості споживачів. Головними завданнями цифрового маркетингу є просування бренду і збільшення збуту за допомогою різних методик. Цифровий маркетинг містить в собі великий вибір маркетингових тактик з просування товарів, послуг і брендів. Крім мобільних технологій, традиційних ТБ і радіо методи цифрового маркетингу використовують інтернет в якості основного комунікаційного посередника [1].

Основним засобом таргетованого маркетингу є лідогенерація. В маркетингу генерація лідів – це ініціювання споживчого інтересу або запиту на продукти або послуги бізнесу. Потенційні клієнти можуть створюватися для таких цілей, як побудова списків, отримання електронних розсилок або отримання потенційних клієнтів. Методи отримання потенційних клієнтів, як правило, потрапляють під парасольку реклами, але можуть також включати не оплачувані джерела, такі як результати звичайних пошукових систем або переходи від наявних клієнтів [2].

Потенційні клієнти можуть надходити з різних джерел або видів діяльності, наприклад, цифровим способом через Інтернет, через особисті перенаправлення, через телефонні дзвінки компанії або телемаркетингу, через рекламу та події. Дослідження 2015 року показало, що 89% респондентів назвали електронну пошту найчастіше використовуваним каналом для отримання потенційних клієнтів, а потім контент-маркетинг, пошукову систему [3].

Простими словами, вся суть лідогенерації зводиться до того, що клієнт надсилає запити на отримання лідів під свої критерії. Бізнес відразу говорить, яка цільова аудиторія йому потрібна, виходячи з послуг чи продукту, який бізнес хоче запропонувати. В свою чергу, компанія яка займається цифровим маркетингом, підбирає цільову аудиторію, виходячи з вимог клієнта і починає вести маркетингову кампанію, окрему для кожного бізнесу. Основним недоліком такого метода є те, що все зводиться до електронної адреси ліда. Вся маркетингова кампанія будується на тому, що в кожної структури, компанії, бізнесу є своя унікальна доменна адреса. І кожному працівнику по заданому патерну, створюється електронна адреса, яка буде деактивована, якщо працівник залишає компанію. Тому електронна адреса ліда, на яку буде відправлено повідомлення, може вже бути не активною або взагалі просто не валідною. Саме тут ситуацію рятують люди, які будуть перевіряти актуальність персональної інформації ліда, валідність електронної адреси за допомогою різних веб-сервісів. З одного боку весь цей процес спрощує те, що є соціальна мережа “Linkedin”, яка й існує, для того щоб люди в пошуках роботи, могли розповісти про свій досвід, посаду, навички, знання певних галузей, та навіть могли указати точну дату початку роботи на якійсь попередній посаді та дату її завершення. Мережа є дуже популярною в усьому світі і так, як на послуги лідогенерації в першу чергу націлені західні та американські компанії, то ми отримуємо просто величезну базу цільової аудиторії для цих компаній. З іншого ж боку, тут ми і отримуємо найбільший недолік такого способу, а саме те, що кожен лід повинен буди оброблений на валідність даних, електронних адрес, та купи інших даних, які запросив клієнт та, які прийшли з бази даних. Весь цей процес в більшості компаній відбувається за допомогою Google Sheets. На перший погляд, це може здатися зручним способом, але коли на одному листі, отриманому з бази даних ми маємо понад 1000 контактів з ім’ям, прізвищем, посадою, назвою компанії, адресою, номером телефону і ще купою полів, то це перетворюється на дуже не зручний та довгий процес. Навіть Google Sheets може не впоратись з об’ємом даних отриманих з бази даних. Саме тому дедалі більше компаній задумуються над іншим способом автоматизації такого процесу, або хоча б його прискоренням.

Тому, виходячи з актуальності та комерційного потенціалу для бізнесу, було вирішено присвятити тему дипломної роботи “Автоматизації процесу лідогенерації для маркетингу за допомогою CRM платформи”. Створивши за допомогою мови програмування Java, веб додаток, який буде являти собою CRUD додаток для додавання, читання, оновлювання та видалення даних з бази даних.

# РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Проект World Wide Web зародився на початку 90-х років у швейцарському інституті CERN. Спочатку він задумувався, як засіб, що полегшує пошук бібліотечної інформації, розподіленої між декількома серверами в CERN. Для цього він використовував концепцію "гіпертексту", розроблену раніше Тедом Нельсоном і Ванневаром Бушем.

У WWW було кілька важливих відмінностей від інших систем гіпертексту, які тоді вже існували (Note Code, HyperCard, Gopher):

* в WWW використовувалися тільки односпрямовані посилання, що дозволяло посилатися на інші ресурси без будь-яких дій з боку власника цього ресурсу;
* WWW не належав будь-якій компанії, тому можна було створювати програмне забезпечення незалежно і без всяких ліцензійних обмежень.

Ці риси зіграли велику роль у вибуховому зростанні популярності WWW в середині 90-х років, коли вийшов веб-браузер Mosaic і наступний за ним комерційний аналог Netscape Navigator. WWW був побудований на трьох основних стандартах: URL (RFC2396), HTTP (RFC2616) і HTML. Наймолодший з них і, отже найбільш розвинений – HTML. Спочатку він був задуманий виключно для семантичної розмітки тексту (посилання, розділи, цитати та інше), потім у зв'язку з комерціалізацією до цього стандарту стали додаватися різні, часто не сумісні один з одним теги для художнього оформлення тексту. Надалі ця тенденція була названа неправильною (з причин відсутності стандартизації та відсутності поділу смислової й оформлювальної частини документації) і консорціумом W3C, який розробляє стандарт HTML, був узятий зворотний курс – на залишення в HTML тільки семантичної розмітки, для оформлення ж був придуманий спеціальний стандарт CSS (каскадні списки стилів).

Виходячи з цього потрібно сформулювати поняття Web-вузла або говорячи по іншому Web-сторінки.

1.1.1 Структура мережі інтернет. Web-вузол (Web-сайт) – це комплекс Web-сторінок та інших ресурсів, об'єднаних відповідно за змістом, розміщених в на одному домені.

Мережа Internet складається з великого числа з'єднаних між собою комп'ютерів, маршрутизаторів та іншого, необхідного для правильної роботи. Кожен елемент мережі Internet (вузол) має неповторний ключ – IP-адресу. Знаючи IP-адресу вузла, є можливість спробувати під'єднатися до нього, а маючи невеликі навички можна з'ясувати, кому ця адреса належить і в якому регіоні світу знаходиться. IP-адреси заведено записувати у вигляді чотирьох груп цифр, розділених крапками, наприклад: 192.168.100.003 або 10.10.0.123.

Запам'ятати адреси всіх часто відвідуваних сторінок практично неможливо, для цього в мережі Internet існують спеціальні сервери DNS (Domain Name System), на яких виконується зберігання всіх списків зіставлення IP-адрес і символьних імен. Завдяки цьому, до серверів, користувач завжди переходить по потрібній IP-адресі, набравши в браузері тільки ім'я сторінки.

Після цього, як ми ввели в рядок браузера ім'я необхідної сторінки, браузер автоматично отримує з DNS IP-адресу потрібного сервера і шле за цією адресою спеціальний запит на отримання сторінки (HTTP-запит). Спеціальна програма, що працює на сервері (Web-сервер) обробляє цей запит і відправляє назад в браузер необхідну сторінку.

Очевидно, що всі дії по відображенню сторінки можна розділити на дві категорії, що:

* виконуються на стороні клієнта (клієнтський код або front-end);
* виконуються на стороні сервера (серверний код або back-end).

Бекенд і фронтенд терміни в програмній інженерії, які розрізняють згідно з принципом поділу відповідальності між зовнішнім поданням і внутрішньою реалізацією відповідно.

При цьому сервер зовсім нічого не знає про поточний стан клієнта, а клієнт – про стан сервера.

При розробці алгоритмів обміну потрібно завжди знати про це і своєчасно

пересилати потрібні дані, що описують поточний стан або необхідні дії.

Залежно від середовища використання розрізняються і засоби реалізації частин. На стороні клієнта використовується зазвичай тільки HTML, JavaScript(AJAX), CSS, Flash. Важливо відрізняти веб-додатки від звичайних веб-сайтів, адже це хоч і схожі, але абсолютно різні веб структури.

Веб-додаток – клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер. Логіка веб-додатку розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається через мережу. Однією з переваг такого підходу є той факт, що клієнт не залежать від конкретної операційної системи користувача, тому веб додатки є кроссплатформенними сервісами.

1.1.2 Принцип роботи та структура Web-додатків. Веб-додатки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на веб-сайтах з великим обсягом інформації.

Даний вид веб-додатків дозволяє здійснювати пошук у вмісті, впорядковувати вміст і переміщатися по ньому зручним для відвідувачів способом.

Веб-додаток дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані та формувати звіти на основі отриманих даних для аналізу. Як приклад можна навести інтерактивні сторінки банків, сторінки для контролю товарних запасів, соціологічні дослідження та опитування, а також форми для зворотного зв'язку з користувачами.

Найчастіше веб-додатки складаються як мінімум з трьох основних компонентів:

1) Клієнтська частина – це те, що ми бачимо на сторінці (графічний інтерфейс). Графічний інтерфейс відображається в браузері. Користувач взаємодіє з веб-додатком через браузер, клікаючи по посиланнях і кнопках.

2) Серверна частина – це програма або скрипт на сервері, що обробляє запити користувача (точніше, запити браузера). Найчастіше серверна частина веб-додаткe програмується на PHP або Java. При кожному переході користувача за посиланням браузер відправляє запит до сервера. Сервер обробляє цей запит, викликаючи деякий скрипт, який формує веб-сторінку, описану мовою HTML, і відсилає Клієнту по мережі. Браузер тут же відображає отриманий результат у вигляді чергової веб-сторінки.

3) База даних (БД або СУБД) – програмне забезпечення на сервері, що займається зберіганням даних і їх видачею в потрібний момент. У разі форуму або блогу, збережені в БД дані – це пости, коментарі, новини, і так далі. База даних розміщується на сервері. Серверна частина веб-додатку (PHP або Java код) звертається до бази даних, витягуючи дані, які необхідні для формування сторінки, запитаної користувачем. Браузер через Інтернет відсилає HTTP-запити веб-серверу. Веб-сервер викликає код, написаний розробником веб-програми. Написаний скрипт звертається до бази даних, якщо це потрібно. В результаті скрипт повертає клієнту веб-сторінку, яку і відображає браузер [4].

Основна мова, якою програмується графічний інтерфейс веб додатків – це HTML. Дана мова описує структуру веб-сторінки, розміщення на ній компонентів.

Оформлення веб-сторінок, їх стиль і колірна схема описуються в таблицях стилів – CSS. Для "оживлення" графічного інтерфейсу, надання йому динамічності, використовуються додаткові технології: скрипти JavaScript, а також вбудовані в веб-сторінку компоненти, створені на Flash, Java або Silverlight.

Відсутність необхідності повністю перезавантажувати сторінку після кожного отримання даних від сервера може істотно прискорити роботу веб-додатки. Така концепція має назву Asynchronous JavaScript and XML (асинхронний JavaScript і XML, Ajax). При використанні даного підходу динамічні запити до сервера відбуваються без видимого перезавантаження веб-сторінки: користувач не помічає, коли його браузер запитує дані [5].

Згідно з класичним визначенням, база даних – це впорядкована сукупність інформації, що зберігається у вигляді множин, кожна з яких містить записи уніфікованого виду. Системи управління базами даних (СУБД) надають програмісту найпотужніший інструментарій для створення, оновлення та обробки великих обсягів інформації, що має складну структуру. У класичній теорії виділяють три типи, три структури баз даних: ієрархічну, мережеву і реляційну. В даний час домінуюче положення займають реляційні бази даних. Лідером серед баз даних, що застосовуються для розробки WEB-додатків, на сьогодні, безумовно, є MySQL. Головна перевага MySQL (плавно переходить в недолік:) – її простота. Як наслідок – найвища швидкість виконання SQL-запитів і необхідність явного програмування основних правил підтримки цілісності і несуперечливості даних на рівні сервера.

Серед інших баз даних, що застосовуються для WEB-розробок, відзначаються Oracle і PostgreSQL. PostgreSQL – це вільно розповсюджувана СУБД з відкритим вихідним кодом, орієнтована головним чином на роботу в UNIX-подібних системах [6].

Найбільш відомі приклади веб-додатків: веб-пошта (Http://gmail.com), інтернет-магазини (http://amazon.com), онлайн аукціони (http://ebay.com). Однак область застосування веб-додатків набагато ширша, ніж електронний бізнес, вони використовуються у багатьох наукових і комерційних областях.

Хоча можливі різні варіації, найчастіше використовується трирівнева архітектура для побудови веб-додатків: веб-браузер, якась технологія надання динамічного веб-контенту і база даних. Веб-браузер посилає запити середнього рівня, який обслуговує їх роблячи запити до бази даних і представляючи результати в користувацький інтерфейс.

Наприклад, розглянемо найпопулярнішу на сьогоднішній день платформу для веб-додатків LAMP – Linux, Apache, MySQL, PHP. Тут роль середнього рівня відіграє веб-сервер Apache з встановленим на ньому модулем підтримки мови програмування PHP, а в якості бази даних виступає MySQL.

## 1.2 Переваги та недоліки Web-додатків

Переваги:

* незалежність від клієнтської, а часто і серверної платформи;
* невимогливість до ресурсів клієнта;
* не вимагає встановлення на клієнтському комп'ютері будь-якого програмного забезпечення крім браузера;
* легкість оновлення версій веб-додатку (оновлюється тільки один раз – на сервері);
* «вбудовані» мережеві можливості (можливість працювати з декількома клієнтами одночасно);
* зберігаються всі дані при переході клієнта з машини на машину.

Недоліки та способи їх обходу:

* низький час реакції – орієнтація на клієнтські технології, типу JavaScript, DOM, Flash, XUL;
* протокол HTTP не зберігає стану – механізм cookies, сесії;
* мала захищеність – захищені з'єднання HTTPS, авторизація вбудована в HTTP або на основі сесій;
* нерозвиненість мови HTML в сенсі форм – розробка стандарту XForms, Flash, Java applets.

Виходячи з цієї інформації, можемо підсумувати. Як вже було зазначено раніше, одна веб-програма може зв’язатися з усіма пристроями. Звичайно, веб-сайт чи додаток повинен бути запрограмований таким чином, щоб його можна було показати незалежно від операційної системи пристрою. Якщо це не адаптивний веб-сайт, то можуть виникнути проблеми з його відображенням на iOS, Android або Windows Phone. Останній вже майже і не підтримується. В такому випадку бізнесу доведеться витратити гроші, які мали на меті заощадити на розробці програми, на покращення веб-сайту. Немає користі створювати веб-програму, якщо не адаптувати свій веб-сайт так, щоб його можна було якісно показати на будь-якому пристрої.

Після розглядання плюсів і мінусів веб-додатків необхідно проаналізовано 6 найактуальніших форм веб-додатків. Ця класифікація складена на основі того, як веб-додатки показують вміст, з яким вони працюють. Ми можемо розрізнити до 6 різних типів веб-додатків:

* статичний веб-додаток;
* динамічний веб-додаток;
* інтернет-магазин або електронна комерція;.
* веб-додаток порталу;
* анімований веб-додаток;
* веб-додаток із „системою управління вмістом”.

1.2.1 Сучасний стан та перспективи. Сьогодні веб-додатки продовжують набирають популярність. Найбільш відвідуваними сайтами стають не чисто інформаційні, гіпертекстові сайти, а ті, які надають будь-який сервіс, яку-небудь взаємодіють з користувачем. Але навіть і звичайні інформаційні сайти часто використовують системи управління контентом для зручності управління інформацією, так що і їх теж можна зарахувати до веб-додатків. Деякі вже ставлять веб в один ряд з Windows, Linux та іншими в якості нової платформи для виконання додатків [7].

Розвиток веб-технологій йде дуже широким кроком. Наразі найцікавішими та найперспективнішими технологіями вважаються:

* XForms – новий стандарт для опису інтерфейсів на XML, що враховує недоліки форм в HTML. Він додає до стандартних можливостей HTML-форм перевірку правильності заповнення, обмін даними у форматі XML і використання стандартних XML технологій, в незалежності від платформи;
* XMLHttpRequest – дає можливість послати на сервер будь-які дані без повного перезавантаження сторінки;
* XUL – ще один стандарт для опису інтерфейсів, який використовується в Mozilla та інших, створених на його основі браузерах, завдяки чому отримав зараз велике поширення в якості технологій для створення «rich» веб-додатків, тобто тих, що володіють більшим набором можливостей на стороні клієнта, ніж традиційні веб-додатки.

Серед цікавих серверних технологій можна виділити сервери веб-додатків. Вони включають в себе функціональність звичайного веб-сервера, але при цьому більш орієнтовані на виконання додатків, а не на роздачу статичних сторінок. Це виражається в наявності різних кешей для прискорення додатків, зручного інтерфейсу адміністрування та підтримки різних технологій розробки додатків.

## 1.3 Огляд наявних програмних рішень

Розробка CRUD web-додатку неможлива без аналізу наявних рішень і накопиченого досвіду в цій сфері. Перш за все потрібно потрібно в деталях проаналізувати і зрозуміти, що собою являє CRUD web-додаток.

В рамках комп'ютерного програмування абревіатура CRUD означає створення, читання, оновлення та видалення. Це чотири основні функції всіх подібних додатків. Крім того, кожна буква в абревіатурі може посилатися на всі функції, що виконуються в реляційних програмах баз даних і зіставляються зі стандартним методом HTTP, оператором SQL або операцією DDS.

CRUD також описує контракт на встановлення зв’язку з користувацьким інтерфейсом, що дозволяє переглядати, шукати та змінювати інформацію за допомогою зручного інтерфейсу. Сутності читаються, створюються, оновлюються та видаляються. Ці самі сутності можна змінити, взявши дані зі сервісного шару (частина архітектури) перед тим, як відправити дані назад до сервісу для оновлення. Крім того, CRUD орієнтований на дата класи та стандартизоване використання методів HTTP.

Більшість програм мають свою реалізацію функцій CRUD. Насправді кожному програмісту в певний момент доводиться стикатися з CRUD. Не кажучи вже про те, що додаток CRUD – це програма, яка використовує форми для отримання та повернення даних із бази даних, що є основним при роботі з базами даних.

Перше посилання на операції CRUD було зроблено Хаїмом Кіловим у 1990 році у статті під назвою “Від семантичного до об'єктно-орієнтованого моделювання даних”. Однак цей термін вперше став популярним у книзі Джеймса Мартіна 1983 року, “Управління середовищем бази даних”. Основні принципи:

* CREATE : виконує оператор INSERT для створення нового запису;
* READ: зчитує записи таблиці на основі первинного ключа з вхідним параметром;
* UPDATE: виконує оператор UPDATE в таблиці на основі вказаного первинного ключа для запису в вираз спільно з оператором WHERE;
* DELETE: Видаляє вказаний рядок у виразі спільно з оператором WHERE.

Залежно від вимог системи у різних користувачів можуть бути різні цикли CRUD. Клієнт може використовувати CRUD для створення облікового запису і доступу до нього при поверненні на певний сайт. Потім користувач може оновити персональні дані або змінити платіжну інформацію.

В епоху Web 2.0 операції CRUD лежали в основі більшості динамічних веб-сайтів. Однак потрібно відрізняти CRUD від методів HTTP. Наприклад, якщо потрібно створити новий запис, то необхідно використовувати метод "POST". Для оновлення запису необхідно використовувати "PUT" або "PATCH". Якщо потрібно видалити запис, то варто скористатися "DELETE" методом. Та ці правила скоріше є думкою деякої частини програмістів. Існує й інше твердження та навіть поняття, що хорошим тоном програмування, буде використовувати тільки POST метод для передачі даних. При цьому реалізуючи свою окрему логіку для кожного метода, що взаємодіє через POST.

Розробник додатків має безліч варіантів виконання операцій CRUD. Одним з найбільш ефективних варіантів є створення набору збережених процедур в SQL для виконання операцій. Іншими словами регулярний вираз SQL. Що стосується збережених процедур CRUD є кілька загальних конвенцій про імена:

* ім'я процедури має закінчуватися реалізованим ім'ям операції CRUD;
* префікс не повинен збігатися з префіксом, використаним для інших користувацьких збережених процедур;
* процедури CRUD для однієї й тієї ж таблиці повинні бути згруповані разом, якщо ми використовуємо ім'я таблиці після префікса.

Після додавання процедур CRUD можна оновити схему бази даних, визначивши сутність бази даних, в якій будуть реалізовані операції CRUD.

CRUD запобігає випадковому перегляду і змінам[8]. Ролі додатків – це метод SQL Server, який дозволяє коду перемикати ідентифікатори без повідомлення користувачу. Для роботи зі спеціальними інструкціями SQL користувачі повинні мати необхідні дозволи для таблиць бази даних. Після отримання дозволу користувачі можуть читати і маніпулювати даними в таких додатках, як Excel, Word та інших. Користувачі можуть навіть обійти бізнес-правила програми.

Проте, це небажана ситуація, яку можна запобігти за допомогою ролі програми. Завдяки інтегрованій безпеці доступу до бази даних через ролі програми, ці типи лазівок можуть бути закриті. Після додавання ролі, надається доступ і призначається пароль. Пароль також закодований в додатку, що ускладнює його зміну. Для маніпулювання даними слід використовувати метод CRUD.

Проаналізувавши велику кількість інформації і спробувавши знайти схожі рішення, був сформульований висновок, що на даному етапі, не можливо знайти якийсь додаток або якесь програмне рішення, яке хоча б частково могло задовольнити вирішення поставленої задачі. Основна складність пошуку подібних реалізацій полягає в тому, що вони створюються для задоволення потреби конкретного бізнесу. І виходячи з цього, код або архітектура цих програмних рішень не може бути розміщена для публічного доступу. Однак, отриманої інформації достатньо для того, щоб з упевненістю почати етап вибору методів розробки для реалізації та проектування саме CRUD web-додатку, який повинен повністю задовольнити поставлені бізнес задачі, а саме: бути крос платформенним, мати автентифікацію та авторизацію, простий та зрозумілий інтерфейс і при цьому реалізовувати CRUD операції.

# РОЗДІЛ 2 ПРОЕКТУВАННЯ WEB-ДОДАТКУ

## 2.1 Структура та технології реалізації

Враховуючи специфіку маркетингової індустрії, для якої розробляється веб-додаток, процес переходу до його використання може виявитися складним і доволі тривалим, якщо говорити про реальне використання на справжньому проекті, де повинен бути плавний перехід з тестуванням кожного модуля та налагодженням системи (рис. 2.1).

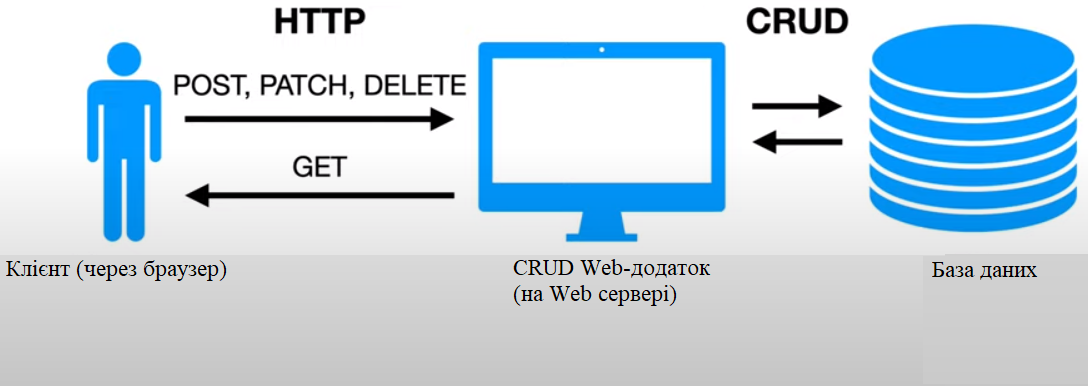


Рисунок 2.1 – Структура додатку, що має бути реалізована

Для написання додатку, було обрано мову програмування Java. Мова Java спочатку позиціонувався розробником, як мова для інтернету. З десктопними додатками у Java не склалося, але зате на стороні сервера Java застосовується дуже широко і має величезну кількість різних цікавих можливостей. Перш за все існує кілька конкуруючих серверів веб-додатків, які незважаючи на відмінності дотримуються деяких стандартів, встановлених Sun, а значить більшість додатків без будь-яких значних модифікацій можуть бути перенесені з сервера на сервер. Крім того, існує кілька різного рівня складності і з різними підходами фреймворків для розробки веб-додатків (тобто бібліотек класів, на основі яких будується веб-додаток). Це фреймворки для структурування додатків на основі патерну MVC (Struts, Spring), бібліотеки для побудови шаблонів веб-сторінок (JSTL, Velocity, Java ServerFaces), бібліотеки для відображення реляційної таблиці на об'єкти і назад (Hibernate).

Саме такий потужний фреймворк, як Spring був обраний для розробки основного функціоналу та модуля безпеки. В сучасних ентерпрайз проектах важко уявити java проект, який міг би обійтися без використання Spring. І говорячи про цей фреймворк, потрібно розуміти, що це не один клас чи технологія. Він складається з певної кількості пакетів не залежних один він одного і об’єднаних під однією назвою. Перш ніж приступити до проектування, потрібно проаналізувати і зрозуміти, які пакети можуть бути використані.

Spring Framework (або коротко Spring) – універсальний фреймворк з відкритим кодом для Java-платформи. Він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків ентерпрайз масштабу. Крім того, особливості ядра Spring можуть бути застосовні в будь-якому Java додатку. Також існує безліч розширень і удосконалень для побудови веб-додатків на Java Enterprise платформі. Саме цим Spring заробив велику популярність і визнається розробниками як стратегічно важливий фреймворк (рис 2.2.).

Одним з компонентів фреймворку є Spring Security, який являє собою інструмент, що надає механізми побудови систем аутентифікації та авторизації, а також інші можливості реалізації безпеки для корпоративних додатків, створених з допомогою Spring Framework.

Основні характеристики:

* аутентифікація та авторизація користувачів;
* захист від атак, таких як фіксація сесії, підробка міжсайтових запитів і так далі;
* можливість інтеграції з Servlet API;
* опціонально підключається модуль Spring Web MVC.

Рисунок 2.2 – Схематичне зображення основних технологій Spring

Spring Boot – комплексний фреймворк для створення і запуску додатків з мінімальними зусиллями і налаштуваннями [9]. Цей модуль ділиться на два стеки: заснований на API сервлетах Spring MVC і реактивний Spring WebFlux.

Spring WebFlux – веб-платформа, створена, щоб по максимуму використовувати переваги сучасних багатоядерних процесорів і обробляти величезну кількість одночасних підключень.

Spring MVC побудований на API сервлетах і використовує архітектуру синхронного вводу-виводу з моделлю «один запит на потік». У Spring Boot також можна опціонально підключити бібліотеку Reactor для створення реактивних систем на JVM.

Основні характеристики:

* вбудовані контейнери Tomcat, Jetty або Undertow, що працюють безпосередньо без розгортання War-файлів;
* готові стартові залежності, що спрощують конфігурацію збірки;
* можливість конфігурувати проект прямо в браузері за допомогою Spring Initializr;
* автоматичне налаштування сторонніх бібліотек (по можливості);
* готові до роботи функції, такі як збір метрик, перевірка працездатності і використання зовнішньої конфігурації;
* немає кодо генерації та не потрібна конфігурація XML – все конфігурується через анотації;

Разом з Spring Boot в проектах зазвичай використовуються Spring Security і Cloud. За допомогою Spring Boot можна створювати:

* мікросервіси;
* реактивні системи;
* веб-додатки.

Spring Data – модуль забезпечує додаткам доступ до даних через реляційні та нереляційні бази даних, map-reduce фреймворки і хмарні сервіси. Spring Data містить безліч підпроектів, призначених для певних СУБД. Серед них є, наприклад, MySQL, MongoDB, Redis і багато інших. Також можна використовувати підмодуль, розроблений спільнотою Spring для більш специфічних баз даних на кшталт ArangoDB, Google Datastore, Microsoft Azure Cosmos DB і других.

Основний механізм, що реалізується в Spring Data – репозиторій. Це набір інтерфейсів, що використовують JPA Entity для взаємодії з даними.

Основні характеристики:

* налаштування відображення сутностей в БД на Java-об'єкти;
* створення динамічних запитів в базу даних;
* базові класи для різних завдань;
* прозорий аудит об'єктів;
* можливість інтегрувати власний код репозиторію;
* проста інтеграція з Spring через JavaConfig, а також кастомних просторів імен XML;
* розширена інтеграція з контролерами Spring MVC.

Spring Data використовується скрізь, де потрібен доступ до даних, і легко інтегрується з іншими модулями Spring.

Spring Cloud – з Spring Cloud можна легко і швидко створювати шаблони в розподілених системах. З прикладів таких шаблонів: управління конфігурацією, виявлення сервісів, інтелектуальна маршрутизація, мікропроксі, одноразові токени і багато іншого. Шаблони, створені за допомогою Spring Cloud, будуть добре працювати в будь-якому розподіленому середовищі, включаючи ноутбук, центри обробки даних і PaaS-платформи, такі як Cloud Foundry.

Spring Cloud також складається з безлічі підпроектів для різних цілей. Так, Spring Cloud Azure інтегрує Spring зі службами Azure. Spring Cloud Stream використовується для створення керованих подіями мікросервісів (event-driven microservices) і так далі.

Основні характеристики:

* розподілена конфігурація;
* реєстрація та виявлення сервісів;
* маршрутизація;
* зв'язок між сервісами (service-to-service calls);
* балансування навантаження;
* вибір лідера і стан кластера;
* розподілений обмін повідомленнями;

В загальному Spring Cloud містить багато корисних інструментів для мікросервісів і розподілених систем.

Spring Integration дозволяє полегшити обмін повідомленнями в додатках на основі Spring, підтримує інтеграцію з зовнішніми системами і дає інструменти для обробки даних з різних джерел. Один з підпроектів Spring Cloud, Spring Cloud Stream, використовує Spring Integration як движок для мікросервісів, керованих подіями.

Основні характеристики:

* багато шаблонів для інтеграції додатків підприємства;
* інтеграція з зовнішніми системами;
* веб-сервіси (SOAP і REST);
* велика підтримка JMX;
* Mbeans-компоненти.

Spring Integration підключається до проекту, якщо потрібно зв'язати POJO (Plain Old Java Object) за допомогою парадигми обміну повідомленнями без впровадження залежностей. Також Integration дозволяє взаємодіяти із зовнішніми системами за допомогою адаптерів, каналів і шлюзів. Адаптери каналів використовуються для односторонньої інтеграції (відправлення або отримання), а шлюзи – для сценаріїв запиту / відповіді (вхідного або вихідного).

Spring Batch – платформа для розробки пакетних додатків. Spring Batch підійде як для простих, так і для складніших проектів – платформа легко масштабується і може обробляти великі обсяги інформації.

Основні характеристики:

* управління транзакціями;
* обробка на основі фрагментів даних;
* декларативне введення;
* веб-інтерфейс адміністрування (Spring Cloud Data Flow).

Spring Batch підійде для додатків з багаторазово використовуваними функціями, щоб обробляти великі обсяги записів. Серед таких функцій – ведення логів і трасування, управління транзакціями, статистика обробки завдань, перезапуск і пропуск завдань, управління ресурсами та інші.

Окремо можна виділити технологію Thymeleaf. Це серверний механізм Java-шаблонів автономних середовищ, здатний обробляти HTML, XML, JavaScript, CSS і навіть простий текст. Основною метою Thymeleaf є створення елегантного і зручного способу шаблонізації. Щоб досягти цього, Thymeleaf грунтується на концепції Natural Templates, щоб впровадити свою логіку в файли шаблонів таким чином, щоб цей шаблон не впливав на відображення прототипу дизайну. Це покращує комунікацію в команді і зменшує розрив між дизайнерсько-програмістськими групами. Thymeleaf також був розроблений з самого початку з урахуванням стандартів Web, особливо HTML5, що дозволяє створювати повністю відповідні стандарту шаблони.

Також не можливо не згадати систему автоматичного збирання проекту Maven, адже не можливо сьогодні уявити проект java без цього програмного модуля. Maven – це інструмент для збирання Java проекту: компіляції, створення jar, створення дистрибутива програми, генерації документація.

Переваги Maven:

* незалежність від ОС. Збірка проекту відбувається в будь-який операційній системі;
* управління залежностями. Рідко які проекти пишуться без використання сторонніх бібліотек (залежностей). Ці сторонні бібліотеки часто теж в свою чергу використовують бібліотеки різних версій. Мавен дозволяє управляти такими складними залежностями. Саме це дозволяє вирішувати конфлікти версій і в разі необхідності легко переходити на нові версії бібліотек;
* можлива збірка з командного рядка. Таке часто необхідно для автоматичного складання проекту на сервері (Continuous Integration);
* хороша інтеграція з середовищами розробки. Основні середовища розробки на java легко відкривають проекти, які збираються за допомогою maven. При цьому найчастіше проект налаштовувати не потрібно – він відразу готовий до подальшої розробки. Як наслідок – якщо з проектом працюють в різних середовищах розробки, то maven зручний спосіб зберігання налаштувань;
* файл налаштування середовища розробки і для збірки один і той же, а це значить, що буде менше дублювання даних і відповідно помилок;
* декларативний опис проекту.

Вище були розглянуті основні модулі Spring. Насправді їх набагато більше. Є ще Spring for Android для створення Android-додатків, Spring CredHub для взаємодії з CredHub-сервером, Spring LDAP і багато інших.

Аналізуючи наведену інформацію, було прийнято рішення використати для реалізації веб-дадатку наступні технології:

* Java EE;
* Apache Tomcat Server;
* PostgreSQL;
* Spring Boot;
* Spring MVC;
* Spring Security;
* Spring Data;
* Thymeleaf;
* Maven;
* JavaScript(Ajax).

Вибір правильних технологій та фреймворків є дуже важливим при розробці програмного забезпечення. Тому технології були обрані таким чином, щоб унеможливити конфлікти несумісності, не перевантажити код не потрібними технологіями, зробити розробку швидкою (допоможе добре написана документація Spring). Отже, виходячи з цього потужність мови програмування java відкриє доступ до необхідних можливостей розробки веб-дадатків, система автоматичного збирання проекту Maven сильно спростить розробку підключивши всі необхідні залежності та конфігурації. Spring MCV дозволить створити основну логіку роботи дадатку, Spring Security надасть технології безпеки, Spring Data дозволить взаємодіяти з базою даних PostgreSQL, а Spring Boot об’єднає в собі всі модулі і використовуючи контейнер для сервлетів Apache Tomcat Server запустить додаток на локальному сервері. Не можна не згадати оптимальність вибору технології Thymeleaf, яка дуже просто дозволить зв’язати високорівневий код java зі звичайним шаблоном HTML, що дозволить вкрай просто сворити front-end частину.

**2.2.1 Функціональні вимоги**. Для того, щоб визначити функціонал додатку необхідно розглянути функції, які можуть виконувати користувачі. Функції будуть частково розділені між роллю адміністратора і звичайного користувача (табл. 2.1). Це забезпечить захист від не узгоджених змін та дозволить контролювати хто саме і на якому етапі вніс зміни в базу даних. Функціонал Spring Security вже “з-під капота” надає готове рішення, автоматично генеруючи HTML сторінку з формою для вводу логіна та пароля. Вже після цього можна буде самостійно додати ролі для доступу.

Таблиця 2.1 – Функції адміністратора та користувача

|  |  |
| --- | --- |
| **Функція** | **Роль** |
| РЕДАГУВАННЯ. Можливість створення та редагування даних лідів і їхніх компаній. | Адміністратор |
| ПЕРЕГЛЯД. Перегляд всіх даних, що надійшли з бази даних. | Адміністратор |
| ВИДАЛЕННЯ. Можливість видалити будь будь якого ліда з бази даних. | Адміністратор |
| РЕДАГУВАННЯ. Можливість лише редагувати дані лідів і їхніх компаній. | Користувач |
| ПЕРЕГЛЯД. Перегляд всіх даних, що надійшли з бази даних. | Користувач |
| ВИДАЛЕННЯ. Відсутня можливість видаляти будь-які елементи з бази даних. | Користувач |

Тобто, функціонал повинен бути реалізований таким чином, щоб користувач міг лише редагувати дані та не видаляти, а адміністратор мав змогу користуватися повним набором функцій.

## 2.2 Проектування архітектури додатку

Програмна архітектура будь-якої програми описує її основні компоненти, їх відношення і те, як вони взаємодіють один з одним. По суті, він служить кресленням. Забезпечує абстракцію для управління складністю системи та встановлення зв'язку і координації між компонентами.

Ось деякі ключові моменти:

* архітектура допомагає визначити рішення, що відповідає всім технічним і експлуатаційним вимогам, із загальною метою оптимізації продуктивності і безпеки;
* проектування архітектури передбачає перетин потреб організації та потреб команди розробників. Кожне рішення може мати значний вплив на якість, швидкодійність та багато іншого.

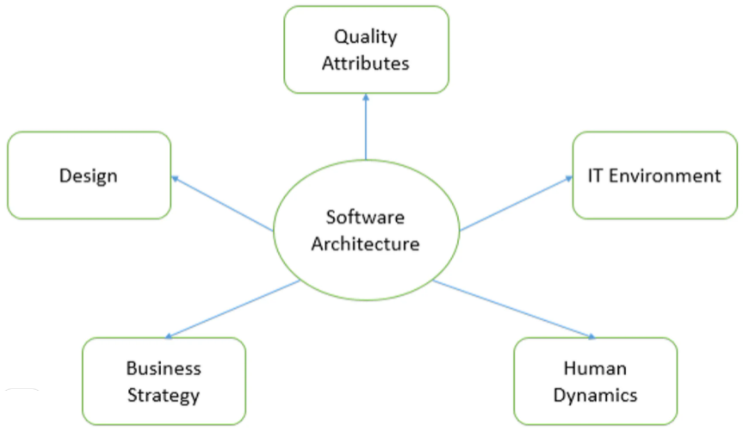
Отже, з урахуванням сказаного, перейдемо до того, чому архітектура програмного забезпечення важлива (рис 2.3).

Рисунок 2.3 – Основні складові архітектури програмного забезпечення

Ключовим елементом успішного створення чого-небудь є правильна база. Будь то будівництво будівлі чи чогось іншого. Якщо не отримаємо правильну базу, доведеться почати все спочатку, іншого шляху немає.

Створення веб-додатку нічим не відрізняється. Архітектура є його основою і повинна бути ретельно продумана, щоб уникнути будь-яких серйозних змін дизайну і рефакторинга коду в більш пізній момент часу.

Багато інженерів скажуть, що не кожен програміст захочете заглиблюватися в перепроектування. Вона пожирає час, як чорна діра. Може здвинути дату здачі проекту далі за календарем на місяці, а то й більше.

Це також залежить від того, на якому етапі процесу розробки ми зайшли в глухий кут через поспішні рішення, прийняті на початкових етапах проектування. Тому, перш ніж ми навіть доторкнутися до коду потрібно зробити базову архітектуру правильною.

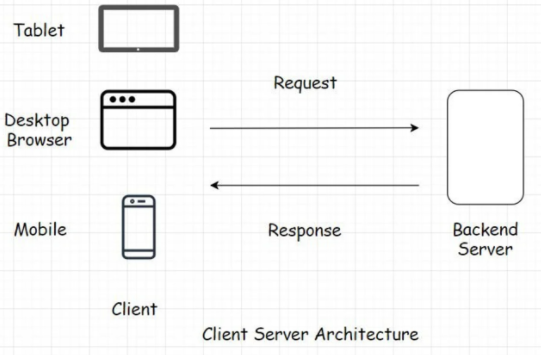
Хоча розробка програмного забезпечення – це ітеративний і еволюційний процес, ми не завжди досягаємо досконалості з першого разу. Однак, це не може бути виправданням. Часто виникає плутанина між дизайном програмного забезпечення та архітектурою, тому це питання буде розглянуто.

Архітектура програмного забезпечення використовується для визначення скелета і високорівневих компонентів системи, а також того, як всі вони будуть працювати разом. Наприклад, чи потрібна нам безсерверна архітектура, яка розділяє додаток на два компоненти: BaaS (backend-as-a-service) і FaaS (functions-as-a-service)?

Або нам потрібно щось на зразок архітектури мікросервісу, в якій різні функції чи завдання розділені на окремі відповідні модулі або кодові бази?

Вибір архітектури визначатиме, як ми справляємося з продуктивністю, відмовостійкістю, масштабованістю і надійністю. Розробка програмного забезпечення відповідає за розробку рівня коду і за те, що робить кожен модуль, область дії класів, призначення функцій та інше. При стратегічному використанні вони можуть зробити програміста значно ефективнішим, дозволивши йому уникнути винаходу колеса, замість цього використовуючи методи, вже вдосконалені іншими.

2.2.1 Шаблони архітектури програмного забезпечення. Клієнт-серверна архітектура працює за моделлю "запит-відповідь" [10]. Клієнт відправляє запит на сервер для отримання інформації, і сервер відповідає на нього (рис. 2.4). Кожен веб-сайт, який ми переглядаємо, будь то блог на Wordpress або веб-додаток, такий як Facebook, Twitter або наш банківський додаток, побудований на архітектурі клієнт-сервер.

 Ще однією наявною архітектурою є мікросервіси (рис. 2.5). В архітектурі мікросервісів різні функції та завдання поділяються на окремі відповідні модулі (кодові бази), які працюють спільно один з одним, утворюючи велику службу в цілому. Ця архітектура полегшує і спрощує обслуговування додатків, розробку функцій, тестування і розгортання в порівнянні з монолітною архітектурою.

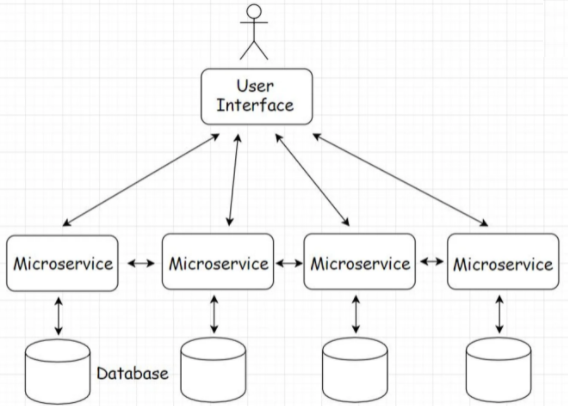
Рисунок 2.4 – Схема клієнт-серверної архітектури

Рисунок 2.5 – Схема мікросервісної архітектур

Model-View-Controller(MVC) – це архітектурний шаблон програмного забезпечення, в якому логіка програми розділена на три компоненти на основі функціональності (рис. 2.5). Ці компоненти називаються:

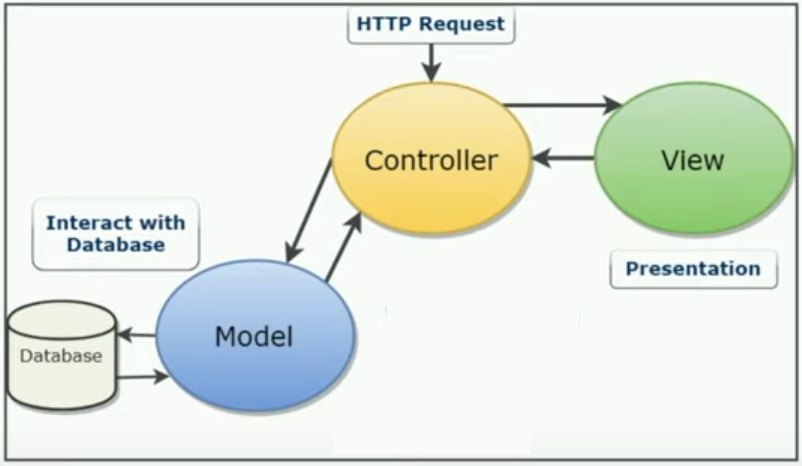
* моделі-представлення, які показують, як дані зберігаються в базі даних;
* представлення-компоненти, видимі користувачеві (графічний інтерфейс);
* контролери-компоненти, які виступають в якості інтерфейсу між моделями і уявленнями.

Рисунок 2.5 – Схема MVC архітектури

Архітектура MVC використовується не тільки для веб-додатків, але і для мобільних і веб-додатків. Основною SpringMVC Framework є підтримка в Spring архітектури модель-уявлення-контролер (model-view-controller). Spring забезпечує готові компоненти, які можуть бути використані (і використовуються) для розробки веб-додатків. Застосовування цієї бібліотеки у поєднанні з бібліотекою Spring Data надає потужні можливості масштабування та розгортання проекту на будь-якій базі даних та на будь-якій платформі без труднощів.

Отже, аналізуючи наявні архітектурні рішення, був отриманий висновок, що архітектура MVC, як найкраще задовільнить вимогу проекта. Будуть отримані гнучкі різні рівні взаємодії між шарами архітектури, що дозволить спростити розробку та в подальшому легко додавати Data та Service класи, які зможуть в свою чергу розширити функціонал і при цьому не рефакторити написаний код.

## 2.3 Проектування бази даних

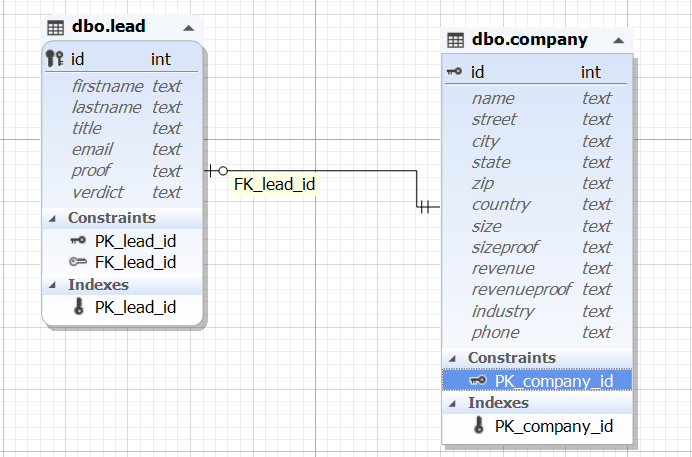
Проектування бази даних – це процес створення проекту, який буде підтримувати формулювання місії та цілей бізнесу для необхідної системи баз даних. Дотримуються двох основних підходів до проектування бази даних. Це:

* знизу вгору;
* зверху донизу.

Підхід знизу вгору починається на фундаментальному рівні атрибутів (тобто властивостей сутностей та відношень), які через аналіз асоціацій між атрибутами об’єднуються у відношення.

Більш доцільною стратегією для проектування складних баз даних є використання підходу зверху донизу, який починається з розробки моделей даних, які містять кілька сутностей і відношень високого рівня, а потім застосовувати послідовні уточнення зверху вниз для ідентифікації сутностей нижчого рівня, відносини та пов'язані з ними атрибути.

Наступним етапом буде проектування бази даних. Визначимо структуру бази, тобто поля таблиць та логічні зв'язки і перенесемо це все в додаток DbForgeStudio, який дозволить автоматично створити схему структури бази даних (рис. 2.6).

Рисунок 2.6 – Структура БД

# РОЗДІЛ 3 РОЗРОБКА CRUD WEB-ДОДАТКУ

## 3.1 Вибір платформи

Система, яку необхідно реалізувати являє собою веб-додаток, тому для реалізації було обрано мову програмування Java та фреймворк Spring MVC.

Метою SpringMVC Framework є підтримка в Spring архітектури модель- уявлення-контролер (model-view-controller). Spring забезпечує готові компоненти, які можуть бути використані (і використовуються) для розробки веб-додатків [5].

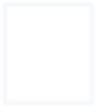
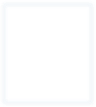
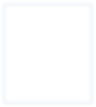
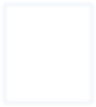
Головною метою MVC є поділ об'єктів, бізнес-логіки й зовнішнього вигляду програми. Всі ці компоненти слабо пов'язані між собою і при бажанні ми можемо змінити, наприклад, зовнішній вигляд програми, не вносячи суттєві зміни в інші два компонента.

Модель (Model). Цей блок інкапсулює дані програми. На практиці це POJO-класи.

Вид (View). Модуль вид відповідає за виведення даних користувачеві. Зазвичай це JSP файл, який може бути розпізнаний та інтерпретований браузером на стороні користувача.

Контролер (Controller). Відповідає за обробку запитів користувачів й передачу даних модулю View для обробки.

В основі Spring MVC Framework лежить DispatcherServlet, завдання якого – обробка всіх HTTP запитів і відповідей. У розумінні DispatcherServlet нам допоможе наступний рисунок (рис. 3.1).



HTTP Request

HTTP Response

4

View

3

View Resolver

2

Controller

1

Handler Mapping

DispatcherServlet

Рисунок 3.1 – Схема роботи DispatcherServlet

DS виконує наступні дії:

* + 1. після отримання HTTP-запиту DS дає вказівку об'єкту Handling Mapping (обробка зв'язування), який викликає наступний об'єкт;
    2. DS надсилає запит контролеру і викликає відповідні методи, в основі яких лежать методи GET і POST. Ці методи повертають об'єкт, відповідно до бізнес логікою методу і передають назву (назву посилання) назад в DS;
    3. за допомогою View Resolver, DS підбирає необхідний вид для запиту;
    4. після формування зовнішнього вигляду DS передає ці дані в модуль View, який обробляється браузером користувача.

У якості СУБД було обрано реляційну систему управління базами даних PosgreSQL. Це не просто реляційна, а об'єктно-реляційна СУБД [11]. Вона надає деякі переваги над іншими базами даних SQL з відкритим вихідним кодом, такими як MySQL, MariaDB та Firebird.

Фундаментальна характеристика об'єктно-реляційної бази даних – це підтримка користувацьких об'єктів та їх подання, зокрема типів даних, функцій, операцій, доменів та індексів. Це робить Постгрес неймовірно гнучким і надійним. Також є можливість створювати, зберігати та виводити складні структури даних.

Існує загальний список типових даних, які підтримує Постгрес. Крім числових, з плаваючою точкою, текстових, булевих та інших очікуваних типів даних, PostgreSQL може похизуватися підтримкою uuid, грошового, геометричного, бінарного типів, мережевих адрес, бітових стрічок, текстового пошуку, xml, json, масивів, композитних типів та діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів та розташування логів. Варто сказати, що MySQL, MariaDB та Firebird також мають деякі з цих даних, але лише Постгрес підтримує їх усі.

Для спрощення роботи з базою даних використаємо JDBCTemplate технологію. Spring надає шаблонний клас, що полегшує роботу з реляційними базами даних SQL та JDBC. У випадках, коли ми використовуємо стандартний JDBC API, то завжди стикаємося з наступними проблемами:

* нам потрібно написати багато коду до і після виконання запиту. Створити з'єднання, вираз, закрити результуючий набір, з'єднання і так далі;
* нам потрібно виконати обробку виключень в логіці бази даних;
* нам потрібно обробити транзакції;
* повторення цього коду з однієї логіки бази даних в іншу – трудомістка задача, яка потребує також не малої кількості часу.

JDBCTemplate – це центральний клас в підтримці Spring JDBC. Він забезпечує створення та вивільнення ресурсів, таких як створення і закриття об'єкта з'єднання і, так далі. Тому, це не призведе до жодних проблем, якщо ми забудемо закрити з'єднання. Він обробляє виключення і надає інформативні повідомлення за допомогою класів excepion, визначених у пакеті org.springframework.dao.

Ми можемо виконувати всі операції з базою даних за допомогою класу JDBCTemplate. Тобто, оновлення, видалення та пошук в базі даних. Це ті основні методи, які необхідно реалізувати для веб-додатку і саме ці методи нам і надає JDBCTemplate.

Тож узагальнюючи вище написане. Основне завдання цієї технології, хоча і не повністю, але розв'язувати основні проблеми звичайного JDBC. Для створення інформаційної системи буде використовуватись: JavaEE, Spring MVC, Spring Boot, Spring Security, JDBCTemplate, PostgreSQL.

## 3.2 Розробка бази даних

Визначимо основні сутності ІС:

Сутність «Lead» містить інформацію про ліда та має наступні атрибути:

* id – ідентифікатор ліда, має тип integer, є первинним ключем, не може бути NULL;
* firstname – ім’я, тип text;
* lastname – прізвище, тип text;
* title – посада, тип text;
* email – електронна пошта, text;
* proof – посилання на ліда, тип text;
* verdict – вердикт по контакту, тип text;

Сутність «Company» містить інформацію про компанію та має наступні атрибути:

* + id – ідентифікатор компанії, має тип integer, є первинним ключем, не може бути NULL;
  + name – назва компанії, тип text;
  + street – вулиця, тип text;
  + city – місто, тип text;
  + state – штат, тип text;
  + zip – поштовий індекс, тип text;
  + country – країна, тип text;
  + size – кількість працівників, тип text;
  + sizeproof – посилання на кількість працівників, тип text;
  + revenue – дохід за рік, тип text;
  + revenueproof – посилання на дохід за рік, тип text;
  + industry – індустрія, тип text;
  + phone – телефон, тип text;

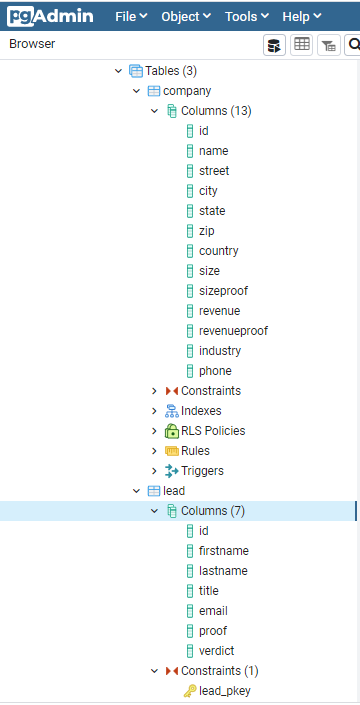
Виходячи з того, що для роботи з базою даних використовується JDBCTempalte, то перед написанням коду моделей потрібно створити базу даних за допомогою зручною графічного інтерфейсу PostgreSQL. Використуючи його, нам навіть не доведеться писати навіть жодного запиту. Переглянемо структуру бази за допомогою інтерфейсу PostgreSQL (Рисунок 3.2).

Рисунок 3.2 – Створена база даних в PostgreSQL

## 3.3 Розробка користувацького інтерфейсу

Головне меню являє собою представлення в зручному вигляді всіх даних на лідів та має три важливі кнопки, а саме (рис. 3.3) :

* «verify» – перенаправляє на сторінку для редагування даних;
* «Create new Lead» – функція доступна лише адміністратору, дозволяє додати до БД дані нового ліда;
* «LogOut» – виходить з облікового запису та перенаправляє на сторінку для введення логіна та паролю.

Після натискання кнопки “Verify”, нас перенаправить на сторінку для редагування даних ліда (рис 3.4). На цьому етапі вносити зміни до даних може як і користувач, так і адмін. Відмінність лише в тому, що адмін може також і видаляти ці дані.

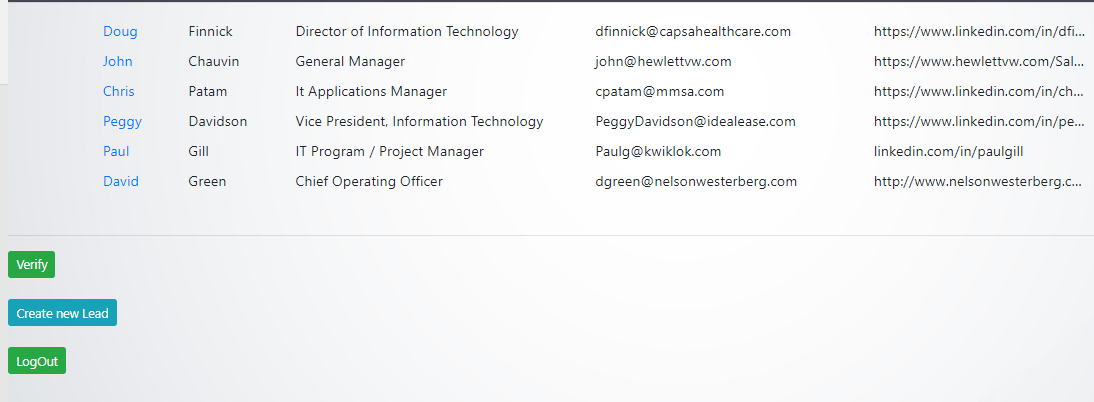


Рисунок 3.3 – Сторінка з відображенням всіх даних з БД

Дана сторінка є основною, яка повністю реалізує саму суть програми. Тому звернемо особливу увагу на особливості та доступні можливості при обробці даних на цій сторінці веб-додатку.

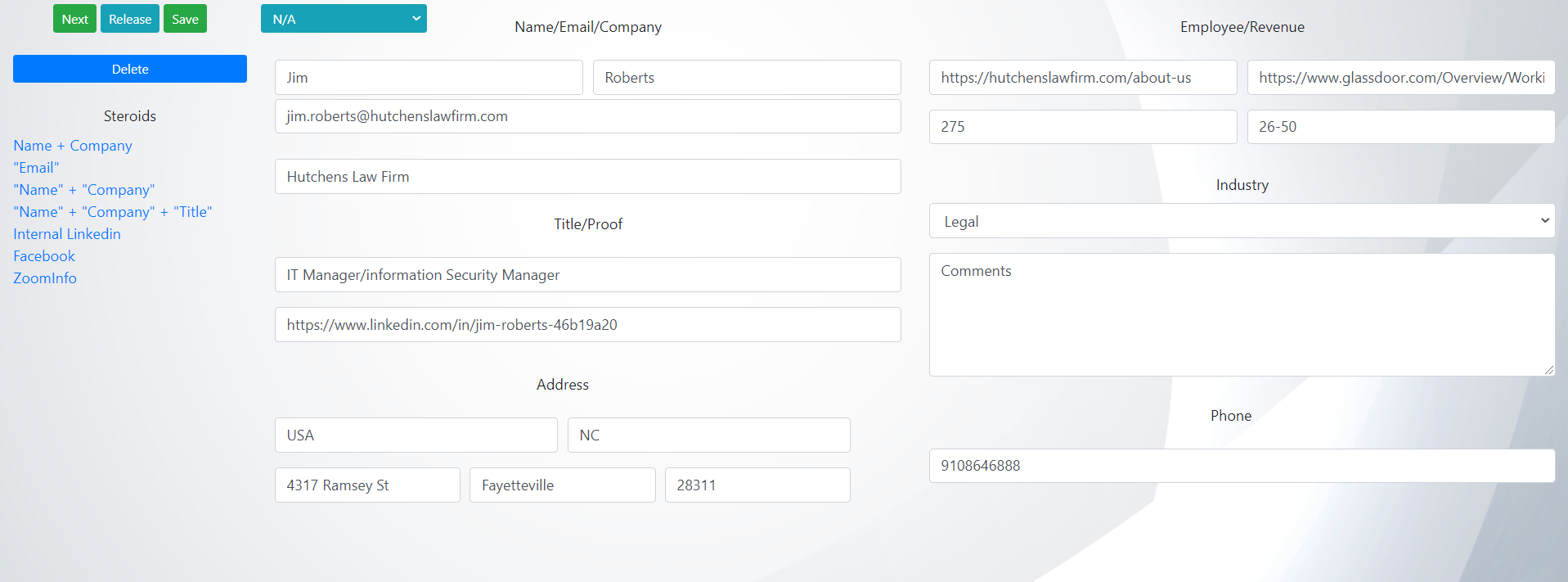


Рисунок 3.4 – Сторінка з даними ліда і його компанії

По своїй структурі сторінка розбита на 3 частини. Ліва частина містить в собі блок “Steroids”, який являє собою гіперпосилання на пошукову сторінку Google. В майбутньому потрібно буде освоїти java script та створити клас який би копіював необхідні поля і вставляв в пошуковий запит, що досить сильно прискорило б процес пошуку інформації. Та не зважаючи на це, наразі блок стероїдів можна використовувати, як патерн при самостійному пошуку в Google. Тому поки що, залишимо цю ідею і реалізуємо її в наступній версії додатку. Вище блоку стероїдів, розміщуються такі кнопки, як: next, release, save, delete.

* Next – при натисканні перенаправляє на наступного ліда, не зберігаючи дані;
* Release – при натисканні перенаправляє на головну сторінку, не зберігаючи дані;
* Save – при натисканні відправляє всі форми та зберігає дані;
* Delete – при настисканні (за умови адміністраторської ролі) видаляє всі дані ліда.

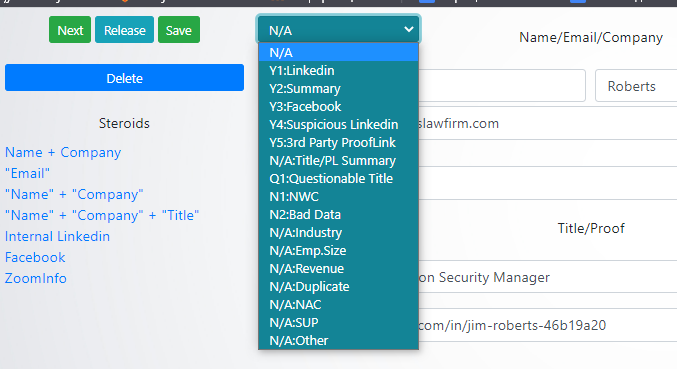
Середня частина містить всю інформацію на ліда з бази даних. Поля відповідають своїм представленням в базі даних, що була спроектована і побудована раніше. Всі поля, окрім поля “email”, можна редагувати. Це зроблено для того, щоб користувач випадково, не вніс зміни до електронної адреси і не зробив її не валідною. Адже, як ми вже говорили, вся маркетингова кампанія будується навколо електронної адреси, на яку буде відправлено рекламну пропозицію. І виходить, що не валідна електронна адреса означає, що далі працювати з даним лідом немає сенсу. В ході розробки стало зрозуміло, що необхідно створити набір критеріїв за якими ми могли б оцінювати на скільки лід відповідає критеріям клієнта. Тому було прийнято рішення створити список з вердиктами, що дозволяє розбити лідів на категорії за яким вони або відповідають або не відповідають вимогам клієнта (рис. 3.5).

Рисунок 3.5 – Список вердиктів, які можуть бути присвоєні ліду

Даний список містить наступні поля:

* N/A – значення за замовчування (відсутність вердикту);
* Y1:Linkedin – лід повністю відповідає вимогам і знайдено його профіль Linkedin або підтвердження його посади на офіційному веб-сайті компінії;
* Y2:Summary – лід відповідає вимогам, якщо проаналізувати його профіль Linkedin або опис на веб-сайті;
* Y3:Facebook – лід відповідає вимогам, в якості підтвердження використано Facebook або іншу соціальну мережу;
* Y4:Suspicious Linkedin – лід відповідає вимогам, використано Linkedin без часових рамок (відсутня інформація про тривалість роботи на даній посаді);
* Y5:3rd Party ProofLink – лід відповідає вимогам, використано всі інші сторонні, не офіційні ресурси для його підтвердження;
* N/A:Title/PL Summary – посада, яку на даний момент займає лід не відповідає вимогам клієнта;
* Q1:Questionable Title – посада, яку на даний момент займає лід не відповідає вимогам клієнта, але є сумнівною і потребує розгляду людини, яка відповідає за маркетингову кампанію і може остаточно вирішити питання відповідності критеріям;
* N1:NWC – лід більше не працює в даній компанії (його імейл деактивовано і рекламна розсилка не можлива);
* N2:Bad Data – отриманий імейл має випадковий набір символів або інші проблеми, за яким одразу помітно, що він не валідний;
* N/A:Industry – індустрія компанії в якій працює лід не відповідає вимогам;
* N/A:Emp.Size – кількість працівників компанії в якій працює лід не відповідає вимогам;
* N/A:Revenue – дохід компанії в якій працює лід не відповідає вимогам;
* N/A:Duplicate – використовується, коли до БД випадково потрапив дублікат “email”, але з іншим патерном;
* N/A:NAC – означає, що дана компанія відсутня в списку обов’язкових компаній, які хоче бачити клієнт;
* N/A:SUP – означає, що дана компанія знаходиться в списку компаній з якими клієнт не хоче працювати;
* N/A:Other – всі інші варіанти, за якими лід не відповідає вимогам.

Тобто, перевіривши актуальність всієї інформації, перевіряльник повинен проаналізувати вимоги і винести вердикт по кожному контакту.

Права частина містить всю інформацію про компанію з бази даних. Поля відповідають своїм представленням в базі даних. Для стандартизації індустрій компаній також було прийнято рішення зробити список з переліком всіх можливих індустрій (рис. 3.6).

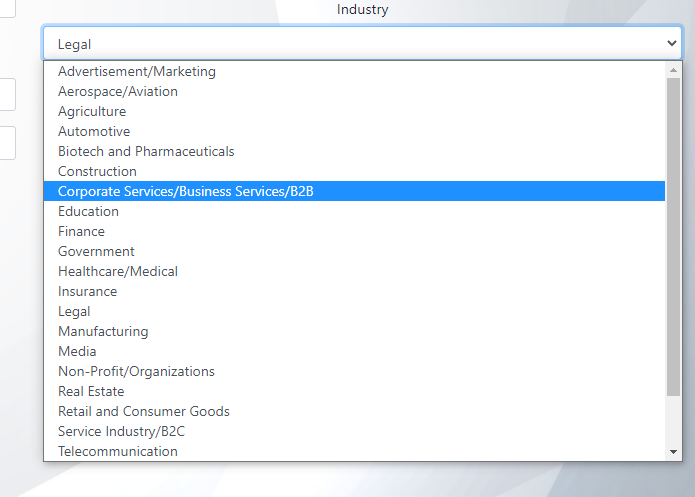
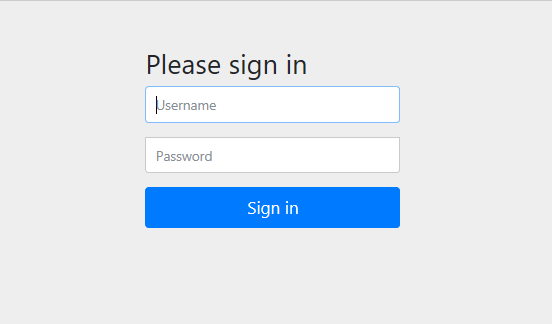


Рисунок 3.5 – Список можливих індустрій компанії

Даний список містить наступні поля:

* Advertising / Marketing – маркетингова індустрія;
* Aerospace / Aviation – авіаційна індустрія;
* Agriculture – аграрна індустрія;
* Automotive – автомобільна індустрія;
* Biotech and Pharmaceuticals – фармакологічна та біотехнологічна індустрія;
* Computers and Technology – ІТ індустрія;
* Construction – будівельна індустрія;
* Corporate Services/Business Services/B2B – сфера послуг для бізнесу;
* Education – навчальна індустрія;
* Finance – фінансова індустрія;
* Government – урядова індустрія;
* Healthcare / Medical – медична індустрія;
* Insurance – страхова індустрія;
* Legal – юридична індустрія;
* Manufacturing – виробнича індустрія;
* Media – медійна індустрія;
* Non-Profit / Organizations – неприбуткові організації;
* Other – всі інші;
* Real Estate – індустрія нерухомості;
* Retail and Consumer Goods – індустрія роздрібної торгівлі;
* Service Industry/B2C – сфера послуг;
* Telecommunications – телекомунікаційна індустрія;
* Transportation and Logistics – логістична індустрія;
* Travel / Hospitality / Entertainment – туристична/ готельна/ розважальна індустрія;
* Utility / Energy – енергетична індустрія.

І останні реалізовані сторінки які теж необхідно описати – це login та logout (рис. 3.6). Як вже було сказано, Spring Security надає вже готове рішення зі сторінкою для введення логіна і пароля “з-під капоту”. Тобто, достатньо лише підключити до pom.xml файлу необхідну залежність. Але готового рішення не завжди буває достатньо. Тому було прийнято рішення. Створити Enum з ролями для прав доступу. Створити окремий клас, який наслідується від WebSecurityConfigurerAdapter та який перевизначить такі методи, як: configure, userDetailsService. В свою чергу – це дозволить нам обмежити доступ до сторінки з додаванням нового ліда до бази даних, заборонить видалення даних звичайним користувачам без відповідних прав, а також налаштувати пароль і логін для кожної ролі.

Рисунок 3.6 – Сторінка Login

# РОЗДІЛ 4 ТЕСТУВАННЯ

## 4.1 Вибір виду тестування

Тестування – завершальний етап розробки програмних модулів, що відіграє важливу роль у процесі створення якісного програмного забезпечення. Техніка тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Тестування ПЗ поділяється на велику кількість видів та класифікується за багатьма ознаками. Для тестування розроблених модулів скористаємося ручним тестуванням, застосовуючи як негативний, так й позитивний підхід.

## 4.2 Тест план

Для того, щоб провести тестування, складемо тест-плани та визначимо функції, що будуть протестовані.

Функції, що повинні бути протестовані за тест-планом тестування головної сторінки:

* + кнопка “Create new Lead” для створення нового ліда (відкривається лише з доступом адміна);
  + кнопка “Verify”, що повинна перенаправити на будь-якого ліда для редагування даних;
  + кнопка “LogOut”, що повинна вийти з облікового запису і перенаправити на сторінку для введення логіна і пароля.

Функції, що повинні бути протестовані за тест-планом сторінки з представленням ліда:

* + редагування інформації у всіх полях за виключенням поля email;
  + кнопка “Delete”, що видалить ліда з БД (можливо лише з доступом адміна);
  + кнопка “Next”, що перенаправить на наступного ліда; заповнення шаблону документа даними;
  + кнопка “Release”, що перенаправить нас на головну сторінку;
  + кнопка “Save”, що збереже відредаговані дані;
  + список з вердиктами;
  + список з індустріями.

Розробимо тест-кейси, які необхідно перевірити. Спочатку перевіримо функції адміністративного модуля.

Таблиця 4.1 – Тест-кейси для головної сторінки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Опис | | Перевірка фунцкіоналу головної сторінки | | |
| Передумови | | Завантажена сторінка «leads» | | |
| № | Дія | | Очікуваний результат | |
| 1 | Натиснути на кнопку «Verify» | | Перенаправлення на сторінку з редагуванням даних ліда |  |
| 2 | Натиснути на кнопку «LogOut» | | Перенаправлення на сторінку для введення логіна і пароля | |
| 3 | Натиснути на кнопку «Create new Lead», залогінившись, як адмін | | Перенаправлення на сторінку для створення нового ліда | |

Продовження таблиці 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Натиснути на кнопку «Create new Lead», залогінившись, як користувач | | | Лід не створюється, помилка доступу |
| Опис | | Перевірка фунціоналу сторінки з даними ліда | | |
| Передумови | | Завантажена сторінка «leads/id» | | |
| № | Дія | | | Очікуваний результат |
| 1 | Натиснути на кнопку «Delete», залогінившись, як адмін | |  | Лід видаляється, перенаправлення на головну сторінку |
| 2 | Натиснути на кнопку «Delete», залогінившись, як користувач | | | Лід не видаляється, помилка доступу |
| 3 | Натиснути на кнопку «Release» | | | Перенаправлення на сторінку /leads |
| 4 | Натиснути на кнопку «Save» | | | Дані зберігаються, з'являється повідомлення про успішне збереження |
| 5 | Натиснути на список та обрати один з вердиктів і потім натиснути кнопку «Save» | | | Дані зберігаються |
| 6 | Відредагувати інформацію в кожному полі та натиснути кнопку «Save», оновити сторінку | | | Всі поля після оновлення збережені |
| Опис | Перевірка роботи сторінки входу | | | |
| Передумови | Завантажена сторінка /login | | | |
| № | Дія | | | Очікуваний результат |
| 1 | Введення логіна та пароля | | | Перевірка введених даних та перенаправлення на сторінку /lead |

Після того, як веб-сервіс буде готовий до розгортання, його можна розгорнути, завантаживши файли і структуру каталогів системи на віддалений сервер. Крім того, є можливість упакувати програму в файл WAR/JAR і розгорнути веб-додаток як один файл.

# ВИСНОВКИ

Мета дипломної роботи полягала в розробці CRUD веб-додатку для автоматизації процесу лідогенерації в маркетинговій індустрії. Говорячи іншими словами, було створено прототип CRM платформи. Хоча, такі платформи – це перш за все прикладне програмне забезпечення для організацій, призначене для автоматизації стратегій взаємодії з замовниками (клієнтами), зокрема для підвищення рівня продажів, оптимізації маркетингу і поліпшення обслуговування клієнтів шляхом збереження інформації про клієнтів і історію взаємин з ними, встановлення і поліпшення бізнес-процесів і подальшого аналізу результатів, але реалізованого функціоналу достатньо для розв'язання основних задач в процесі лідогенерації. Будь-який програмний продукт повинен підтримуватись, оновлюватись, доопрацьовуватися. Тому, всі інші функції, які повинна мати повноцінна CRM платформа повинні бути розроблені та внесені окремим модулем в майбутньому. Це можливо завдяки продуманим архітектурним рішенням зробленим на етапі проектування та вибору технологій. Адже, саме Java в поєднанні з системою автоматичного збирання проекту Maven до якої в вигляді залежностей підключені технології Spring Framework, дозволяє досягти модульності при розробці програмного забезпечення.

Розроблений програмний веб-додаток автоматизує процес збору та обробки інформації. Основними функціями є додавання, редагування, видалення та перегляд інформації на лідів з БД. А отже, можна з впевненістю сказати, що поставлену задачу реалізовано. В процесі розробки були розглянуті та проаналізовані різноманітні архітектурні та проектні рішення. В процесі розробки були покращені навички програмування. Також було освоєно досить не просту технологію Spring Framework, яку вдалося, реалізувати на практиці. Знову ж таки, говорячи про цей фреймворк, не можливо не згадати чималу кількість технологій, які були не тільки розглянуті, а й використані та поєднанні між собою для реалізації бізнес-логіки додатку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Digital marketing [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://ru.wikipedia.org/wiki/Цифровой_маркетинг>
2. Лідогенерація [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://en.wikipedia.org/wiki/Lead\_generation
3. Events Successfully Target C-Suite Decision-Makers [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.emarketer.com/Article/Events-Successfully-Target-C-Suite-Decision-Makers/1013372
4. Структура веб додатків [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: http://labaka.ru/likbez/struktura-veb-prilozheniya
5. Огляд засобів створення інтерфейсів веб-додатків на мові Java [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: http://masters.donntu.org/2013/fknt/riabinin/library/article1.htm
6. Бази даних [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.wwwmaster.ru/bazy-dannyh-dlya-web
7. Web-API [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <http://www.joelonsoftware.com/articles/APIWar.html>
8. Що таке CRUD ? [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://stackify.com/what-are-crud-operations/>
9. Огляд модулів Spring [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://tproger.ru/articles/spring-modules-overview/>
10. Архітектура ПЗ [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://www.educative.io/blog/how-to-design-a-web-application-software-architecture-101>
11. Переваги PostgreSQl [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://habr.com/ru/post/282764/

# ДОДАТОК А ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

1. Клас MySpringMvcDispatcherSerlvetIntitializer

public class MySpringMvcDispatcherSerlvetIntitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return null;

}

@Override

protected Class<?>[] getServletConfigClasses() {

return new Class[]{SpringConfig.class};

}

@Override

protected String[] getServletMappings() {

return new String[]{"/"};

}

@Override

public void onStartup(ServletContext aServletContext) throws ServletException {

super.onStartup(aServletContext);

registerHiddenFieldFilter(aServletContext);

}

private void registerHiddenFieldFilter(ServletContext aContext) {

aContext.addFilter("hiddenHttpMethodFilter",

new HiddenHttpMethodFilter()).addMappingForUrlPatterns(null ,true, "/\*");

}

@Override

protected void customizeRegistration(Dynamic registration) {

registration.setInitParameter("enableLoggingRequestDetails", "true");

}}

1. Клас SecurityConfig

@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf()  
 .disable()  
 .authorizeRequests()  
 .antMatchers("/leads/new")  
 .hasRole(String.valueOf(Role.ADMIN))  
 .and()  
 .formLogin()  
 .and()  
 .logout()  
 .and()  
 .httpBasic();  
 }  
 @Bean  
 @Override  
 protected UserDetailsService userDetailsService() {  
 return new InMemoryUserDetailsManager(  
 User.builder().username("admin")  
 .password(passwordEncoder().encode("admin"))  
 .roles(Role.ADMIN.name())  
 .build(),  
 User.builder().username("user")  
 // Use without encode first  
 .password(passwordEncoder().encode("user"))  
 .roles(Role.USER.name())  
 .build()); }

1. Клас SpringConfig

@Configuration

@ComponentScan("ua.edu.nupp")

@EnableWebMvc

public class SpringConfig implements WebMvcConfigurer {

private final ApplicationContext applicationContext;

@Autowired

public SpringConfig(ApplicationContext applicationContext) {

this.applicationContext = applicationContext;

}

@Bean

public SpringResourceTemplateResolver templateResolver() {

SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();

templateResolver.setApplicationContext(applicationContext);

templateResolver.setPrefix("/WEB-INF/views/");

templateResolver.setSuffix(".html");

return templateResolver;

}

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/\*\*")

.addResourceLocations("classpath:/static/");

registry.addResourceHandler("/webjars/\*\*").addResourceLocations("/webjars/").resourceChain(false);

}

@Bean

public SpringTemplateEngine templateEngine() {

SpringTemplateEngine templateEngine = new SpringTemplateEngine();

templateEngine.setTemplateResolver(templateResolver());

templateEngine.setEnableSpringELCompiler(true);

return templateEngine;

}

@Override

public void configureViewResolvers(ViewResolverRegistry registry) {

ThymeleafViewResolver resolver = new ThymeleafViewResolver();

resolver.setTemplateEngine(templateEngine());

registry.viewResolver(resolver);

}

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName("org.postgresql.Driver");

dataSource.setUrl("jdbc:postgresql://localhost:5432/DiplomaDataBase");

dataSource.setUsername("postgres");

dataSource.setPassword("1234");

return dataSource;

}

@Bean

public JdbcTemplate jdbcTemplate() {

return new JdbcTemplate(dataSource());

}

}

1. Клас CompaniesController

@Controller

@RequestMapping("/companies")

public class CompaniesController {

private final CompanyDAO companyDAO;

@Autowired

public CompaniesController(CompanyDAO companyDAO) {

this.companyDAO = companyDAO;

}

@GetMapping()

public String index(Model model) {

model.addAttribute("companies", companyDAO.index());

return "companies/index";

}

@GetMapping("/companies/{id}")

public String show(@PathVariable("id") int id, Model model) {

model.addAttribute("company", companyDAO.show(id));

return "companies/show";

}

@PostMapping("/update")

public String update(@RequestBody Company company) {

companyDAO.update(company);

return "redirect:/leads/"+company.getId();

}

}

1. Клас LeadsController

@Controller

@RequestMapping("/leads")

public class LeadsController {

private final LeadDAO leadDAO;

private final CompanyDAO companyDAO;

@Autowired

public LeadsController(LeadDAO leadDAO, CompanyDAO companyDAO) {

this.companyDAO = companyDAO;

this.leadDAO = leadDAO;

}

@GetMapping()

public String index(Model model) {

model.addAttribute("leads", leadDAO.index());

model.addAttribute("companies", companyDAO.index());

return "leads/index";

}

@GetMapping("/{id}")

public String show(@PathVariable("id") int id, Model model) {

model.addAttribute("lead", leadDAO.show(id));

model.addAttribute("company", companyDAO.show(id));

return "leads/show";

}

@GetMapping("/new")

public String newLead(@ModelAttribute("lead") Lead lead, @ModelAttribute("company") Company company) {

return "leads/new";

}

@PostMapping("/new")

public String create(@ModelAttribute("lead") Lead lead, @ModelAttribute("company") Company company) {

leadDAO.save(lead);

companyDAO.save(company);

return "redirect:/leads/";

}

@PostMapping("/update")

public String update(@RequestBody Lead lead) {

System.out.println(lead);

leadDAO.update(lead);

return "redirect:/leads/"+lead.getId();

}

@PostMapping("/{id}")

public String delete(@PathVariable("id") int id) {

leadDAO.delete(id);

return "redirect:/leads";

}

}

1. Клас CompanyDAO

@Component

public class CompanyDAO {

private final JdbcTemplate jdbcTemplate;

@Autowired

public CompanyDAO(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public List<Company> index() {

return jdbcTemplate.query("SELECT \* FROM company", new BeanPropertyRowMapper<>(Company.class));

}

@SuppressWarnings("deprecation")

public Company show(int id) {

return jdbcTemplate.query("SELECT \* FROM company WHERE id=?", new Object[]{id}, new BeanPropertyRowMapper<>(Company.class))

.stream().findAny().orElse(null);

}

public void save(Company company) {

jdbcTemplate.update("INSERT INTO company VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", company.getId(),

company.getName(), company.getSizeproof(),

company.getSize(), company.getRevenueproof(), company.getRevenue(), company.getIndustry(),

company.getPhone(), company.getCountry(), company.getState(), company.getStreet(),

company.getCity(), company.getZip());

}

public void update(Company updatedcompany){

jdbcTemplate.update("UPDATE company SET name=?, sizeproof=?, size=?, revenueproof=?, "

+ "revenue=?, industry=?, phone=?, country=?, state=?, street=?, city=?, zip=? WHERE id=?",

updatedcompany.getName(), updatedcompany.getSizeproof(),

updatedcompany.getSize(), updatedcompany.getRevenueproof(), updatedcompany.getRevenue(), updatedcompany.getIndustry(),

updatedcompany.getPhone(), updatedcompany.getCountry(), updatedcompany.getState(), updatedcompany.getStreet(),

updatedcompany.getCity(), updatedcompany.getZip() , updatedcompany.getId());

}

public void delete(int id) {

jdbcTemplate.update("DELETE FROM company WHERE id=?", id);

}

}

1. Клас LeadDAO

@Component

public class LeadDAO {

private final JdbcTemplate jdbcTemplate;

@Autowired

public LeadDAO(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public List<Lead> index() {

return jdbcTemplate.query("SELECT \* FROM lead", new BeanPropertyRowMapper<>(Lead.class));

}

@SuppressWarnings("deprecation")

public Lead show(int id) {

return jdbcTemplate.query("SELECT \* FROM lead WHERE id=?", new Object[]{id}, new BeanPropertyRowMapper<>(Lead.class))

.stream().findAny().orElse(null);

}

public void save(Lead lead) {

jdbcTemplate.update("INSERT INTO lead VALUES(?, ?, ?, ?, ?, ?, ?)", lead.getId(), lead.getFirstName(), lead.getLastName(),

lead.getTitle(), lead.getEmail(), lead.getProof(), lead.getVerdict());

}

public void update(Lead updatedLead){

jdbcTemplate.update("UPDATE lead SET firstname=?, lastname=?, title=?, email=?, proof=?, verdict=? WHERE id=?",

updatedLead.getFirstName(),

updatedLead.getLastName(),

updatedLead.getTitle(),

updatedLead.getEmail(),

updatedLead.getProof(),

updatedLead.getVerdict(),

updatedLead.getId());

}

public void delete(int id) {

jdbcTemplate.update("DELETE FROM lead WHERE id=?", id);

}

}

1. Клас Company

public class Company {

private int id;

private String name;

private String street;

private String city;

private String state;

private String zip;

private String country;

private String size;

private String sizeproof;

private String revenue;

private String revenueproof;

private String industry;

private String phone;

public Company() {

}

public Company(int id, String name, String street, String city, String state, String zip, String country,

String size, String sizeproof, String revenue, String revenueproof, String industry, String phone) {

this.id = id;

this.name = name;

this.street = street;

this.city = city;

this.state = state;

this.zip = zip;

this.country = country;

this.size = size;

this.sizeproof = sizeproof;

this.revenue = revenue;

this.revenueproof = revenueproof;

this.industry = industry;

this.phone = phone;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getZip() {

return zip;

}

public void setZip(String zip) {

this.zip = zip;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

public String getSize() {

return size;

}

public void setSize(String size) {

this.size = size;

}

public String getSizeproof() {

return sizeproof;

}

public void setSizeproof(String sizeproof) {

this.sizeproof = sizeproof;

}

public String getRevenue() {

return revenue;

}

public void setRevenue(String revenue) {

this.revenue = revenue;

}

public String getRevenueproof() {

return revenueproof;

}

public void setRevenueproof(String revenueproof) {

this.revenueproof = revenueproof;

}

public String getIndustry() {

return industry;

}

public void setIndustry(String industry) {

this.industry = industry;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

@Override

public String toString() {

return "Company [id=" + id + ", name=" + name + ", street=" + street + ", city=" + city + ", state=" + state

+ ", zip=" + zip + ", country=" + country + ", size=" + size + ", sizeproof=" + sizeproof + ", revenue="

+ revenue + ", revenueproof=" + revenueproof + ", industry=" + industry + ", phone=" + phone + "]";

}}

1. Клас Lead

public class Lead {

private int id;

private String firstName;

private String lastName;

private String title;

private String email;

private String proof;

private String verdict;

public Lead() {

}

public Lead(int id, String firstName, String lastName, String title, String email, String proof, String verdict) {

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

this.title = title;

this.email = email;

this.proof = proof;

this.verdict = verdict;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getProof() {

return proof;

}

public void setProof(String proof) {

this.proof = proof;

}

public String getVerdict() {

return verdict;}

public void setVerdict(String verdict) {

this.verdict = verdict;}

@Override

public String toString() {

return "Lead{" +

"id=" + id +

", firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

", title='" + title + '\'' +

", email='" + email + '\'' +

", proof='" + proof + '\'' +

", verdict='" + verdict + '\'' +

'}';}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Lead lead = (Lead) o;

return id == lead.id && Objects.equals(firstName, lead.firstName) && Objects.equals(lastName, lead.lastName) && Objects.equals(title, lead.title) && Objects.equals(email, lead.email) && Objects.equals(proof, lead.proof) && Objects.equals(verdict, lead.verdict);

}

@Override

public int hashCode() {

return Objects.hash(id, firstName, lastName, title, email, proof, verdict);

}}

1. Enum Role

public enum Role {

USER,

ADMIN;

}

1. Клас SecuringWebApplication

@SpringBootApplication

public class SecuringWebApplication {

public static void main(String[] args) throws Throwable {

SpringApplication.run(SecuringWebApplication.class, args);

}}