

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)
магістра**
(освітньо-кваліфікаційний рівень)
на тему

Апаратно-програмний комплекс для авторизації процесу лідогенерації в
маркетингу

Виконав: студент 6 курсу, групи 601-ТК
спеціальності

123 Комп'ютерна інженерія
(шифр і назва напрямку)

Власенко М.В.
(прізвище та ініціали)

Керівник Фесенко Т.Г.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

**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 МАГІСТРА

спеціальність 123 «Комп'ютерна інженерія»

на тему

**«Апаратно-програмний комплекс для авторизації процесу лідогенерації в
маркетингу»**

Студента групи 601-ТК Власенка Мирослава Валерійовича

Керівник роботи
доктор технічних наук,
професор Фесенко.Т.Г.

В.о. завідувача кафедри

(прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра: 93с., 25 рисунків, 2 таблиць, 1 додатку, 11 джерел.

Об'єкт дослідження: процеси лідогенерації в маркетингу ІТ компанії.

Мета роботи: розробка апаратно-програмного комплексу автоматизованої системи управління даними та процесом їх збору та обробки, що дозволе знизити затрати часу на маркетинговий процес.

Методи: проектування та розробка бази даних для автоматизованої системи управління даними в Microsoft SQL Server, розробка інтерфейсу для користувача з використанням технологій фреймворку ASP.NET Core.

Ключові слова: апаратно-програмний комплекс, ASP.NET Core, C#, модель, база даних, лідогенерація, лід.

SUMMARY

Master's qualification work: 93 page numbers, 25 pictures, 2 tables, 1 appendix, 11 sources.

Research object: lead generation processes in marketing, IT company.

The purpose of the work: development of a hardware and software complex of an automated data management system and the process of their collection and processing, which will reduce time spent on the marketing process.

Methods: design and development of a database for an automated data management system in Microsoft SQL Server, development of a user interface using ASP.NET Core framework technologies.

Keywords: ASP.NET Core, C#, Identity, Entity, model, database, lead generation, lead.

ЗМІСТ

РЕФЕРАТ.....	2
SUMMARY	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	5
ВСТУП.....	6
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	8
1.1 Опис предметної області.....	8
1.2 Переваги та недоліки Web-додатків	12
1.3 Типи веб-розробки	14
1.4 Типи веб-додатків	15
РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ	19
2.1 Структура та технології реалізації	19
2.2 Проектування архітектури додатку	28
2.3 Проектування бази даних.....	31
РОЗДІЛ 3 РОЗРОБКА WEB-ДОДАТКУ	33
3.1 Вибір платформ	33
3.2 Розробка бази даних	37
3.3 Розробка користувацького інтерфейсу	41
РОЗДІЛ 4 ТЕСТУВАННЯ	48
4.1 Вибір виду тестування	48
4.2 Тест план.....	48
ВИСНОВКИ	51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	52
ДОДАТОК А.....	54
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ	54

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

WWW – World Wide Web

HTTP – HyperText Transfer Protocol

HTML – HyperText Markup Language

URL – Uniform Resource Locator

SQL – Structured Query Language

MVC – Model-View-Controller

ПЗ – програмне забезпечення

ОС – операційна система

БД – база даних

СУБД – система управління базами даних

ВСТУП

Інформаційні технології зіграли важливу роль у маркетингових дослідженнях і створили цілий розділ маркетингу – діджитал маркетинг[1]. На сьогоднішній день стан речей в бізнес-середовищі проходить таким чином, що жоден менеджер не буде приймати рішення, якщо не володіє достатньою інформацією. Та слід зазначити що інформація має бути актуальною, тому у маркетингових дослідженнях збір інформації має бути систематичним і об'єктивним. Інформаційна технологія – це технологія, яка підтримує дії, пов'язані зі створенням, зберіганням, маніпулюванням та передачею інформації разом із відповідними методами, управлінням і застосуванням, тому глобалізація інформаційних технологій стала величезною перевагою для маркетингових процесів. Одним із таких процесів є процес лідогенерації[2].

Лідогенерація[2] – це будь-який метод, за допомогою якого ви збираєте клієнтську базу. Головне в цьому процесі – зацікавити людину настільки, щоб вона зробила перший крок до воронки продажів. Наприклад, залишити їх електронну пошту. Відтепер це ваш «лід».

Лід (від англійського lead) — це поняття, яке має два значення. По-перше, лід у продажах — це потенційний клієнт, який може зробити покупку. По-друге, лід у маркетингу — це контакт (наприклад, емейл-адреса), який потім використовують у маркетинговій воронці.

Основними методами лідогенерації є: електронна пошта, LinkedIn, офіційний сайт компанії, Facebook. Всі ці методи потребують системного підходу. Щоб обробляти ліди якомога швидше у багатьох компаній для цього процесу використовується Google Sheets. На перший погляд це зручний та не затратний варіант. Але візьмемо випадок коли потрібно обробити не декілька десятків чи сотень, а декілька тисяч лідів, тоді такий варіант перестає бути зручним і систематизований процес буде більше схожий на хаотичний, що спричинить дезорієнтацію та допускання помилок спеціалістом, що перевіряє лідів. Тому все більше великих компаній шукають способи автоматизувати даний процес, виходячи з актуальності даної проблеми було вирішено

присвятити тему дипломної роботи «Автоматизації процесу лідогенерації для маркетингу».

Об'єктом дослідження є процес лідогенерації в маркетингу. Нам потрібно створити базу даних в якій будуть зберігатися дані про лідів і компанії яких вони працюють та дані про замовника і вимоги які він ставить для підбору лідів. Метою дослідження буде пошук більш ефективного методу генерації ніж Google Sheets, після аналізу було вирішено розробити апаратно-програмний комплекс, який автоматизує[13] та систематизує процес наповнення бази даних.

В роботі створено апаратно-програмний комплекс, за допомогою мови програмування C#, що являє собою веб-додаток який надає змогу обробки інформації про ліда та можливість генерування email адреси для рекламної розсилки. Веб додаток «Lead Generation Application» реалізований на основі фреймвоку ASP.NET. Перевагами розробленого додатку є: додавання даних лідів, списків з вимогами замовників, читання даних про списки та видалення даних з БД.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Прокладаючи шлях до інтернет-революції, яка змінила світ лише за три десятиліття, Всесвітня павутина складається з багатьох компонентів, які дозволяють користувачам отримувати доступ до різноманітних ресурсів, документів і веб-сторінок в Інтернеті. Таким чином, WWW схожа на величезну електронну книгу, сторінки якої зберігаються або розміщені на різних серверах по всьому світу. Ці сторінки є основним компонентом або будівельними блоками WWW і пов'язані за допомогою гіперпосилань, які забезпечують доступ з однієї певної точки в гіпертекстовому або гіпермедійному документі до іншої точки в цьому документі або іншого. Гіперпосилання є ще одним визначальним поняттям WWW і забезпечують його ідентичність як набір взаємопов'язаних документів.

Гіпертекст – це метод миттєвого перехресного посилання на інформацію, який підтримує спілкування в Інтернеті. Гіпертекст дозволяє легко зв'язувати вміст однієї веб-сторінки з вмістом іншої веб-сторінки чи сайту. Гіпертекст і HTTP дозволяють людям отримувати доступ до мільйонів веб-сайтів, активних у WWW.

Протокол передачі гіпертексту (HTTP) є ще одним ключовим компонентом WWW. Він дозволяє користувачам отримувати доступ до веб-сторінок шляхом стандартизації зв'язку та передачі даних між серверами та клієнтами Інтернету. Більшість веб-документів і сторінок створено за допомогою мови гіпертекстової розмітки (HTML), текстового способу опису структури вмісту файлу HTML. HTML описує структуру веб-сторінок за допомогою елементів або тегів і відображає вміст цих сторінок у веб-браузері.

Для того, щоб отримати доступ до однієї з цих сторінок, користувач і його клієнтська машина передають універсальний ідентифікатор на веб-сервер через

браузер. Цей ідентифікатор може бути уніфікованим покажчиком ресурсу (URL) або уніфікованим ідентифікатором ресурсу (URI) і є унікальним для кожної веб-сторінки. Браузер приймає URL або URI, надані користувачем, і передає їх на веб-сервер. Потім сервер отримує веб-сторінку, пов'язану з цією URL-адресою або URI, і представляє її користувачеві у вікні браузера його клієнтської машини.

Оскільки Інтернет — це глобальна мережа комп'ютерів, кожен комп'ютер, підключений до Інтернету, повинен мати унікальну адресу. Інтернет-адреси мають форму `nnn.nnn.nnn.nnn`, де `nnn` має бути числом від 0 до 255. Ця адреса відома як IP-адреса.

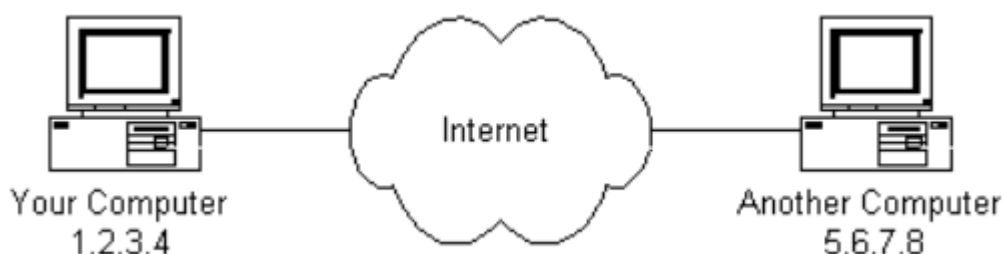


Рисунок 1.1 – Структура інтернет мережі

Магістраль Інтернету складається з багатьох великих мереж, які з'єднуються одна з одною. Ці великі мережі відомі як постачальники мережевих послуг або NSP. Серед великих NSP є UUNet, CerfNet, IBM, BBN Planet, SprintNet, PSINet, а також інші. Ці мережі взаємодіють одна з одною для обміну пакетним трафіком. Кожен NSP повинен підключатися до трьох точок доступу до мережі або NAP. У точках NAP пакетний трафік може переходити від магістралі одного NSP до магістралі іншого NSP. NSP також з'єднуються між міськими біржами або MAEс. MAE служать тій же меті, що й NAP, але є приватною власністю. NAP були оригінальними точками з'єднання Інтернету. І NAP, і MAE називаються точками обміну Інтернетом або IX. NSP також продають пропускну здатність меншим мережам, таким як Інтернет-провайдери та менші постачальники пропускну здатності.

Для передачі даних між комп'ютерами є протокол DNS — це розподілена база даних, яка зберігає імена комп'ютерів та їхні відповідні IP-адреси в

Інтернеті. Багато комп'ютерів, підключених до Інтернету, розміщують частину бази даних DNS і програмне забезпечення, яке дозволяє іншим отримати до неї доступ. Ці комп'ютери називаються DNS-серверами. Жоден DNS-сервер не містить усієї бази даних; вони містять лише його підмножину. Якщо DNS-сервер не містить доменного імені, яке запитує інший комп'ютер, DNS-сервер перенаправляє запитуючий комп'ютер на інший DNS-сервер.

Сучасні програмні додатки та інформаційні системи досягли такого рівня розвитку, що застосовуваний до них термін «архітектура» вже не здається недоречним. Створити з нуля ефективно та надійно функціонуючу інформаційну систему не простіше, ніж побудувати багатоповерховий будинок.

Коротко кажучи, архітектура веб-додатків — це «скелет» або макет, який відображає взаємодію між компонентами додатків, системами проміжного програмного забезпечення, інтерфейсами користувача та базами даних. Така взаємодія дозволяє кільком додаткам працювати разом одночасно.

Як тільки користувач відкриває веб-сторінку, сервер надсилає певні дані браузеру у відповідь на запит користувача. Якщо бути точним, веб-клієнт (або агент користувача) може запитувати веб-ресурси або більш відомі веб-документи (HTML, JSON, PDF тощо) через веб-сервер. Потім, вуаля — за допомогою цих мінімальних маніпуляцій з'являється потрібна інформація. Після цього починається взаємодія між користувачем і сайтом.

Архітектура програмного забезпечення — це скелет і всі високорівневі компоненти системи та те, як вони взаємодіють один з одним[8]. Відповідає на запитання «Де?» і як?". Архітектура програмного забезпечення — це те, де ви зупиняєтеся, щоб вирішити:

- що ти збираєшся робити;
- як ви збираєтеся реалізувати бізнес-вимоги на високому рівні;
- як правильно розташувати сервери;
- де буде зберігатися дата;
- які компоненти будуть потрібні.

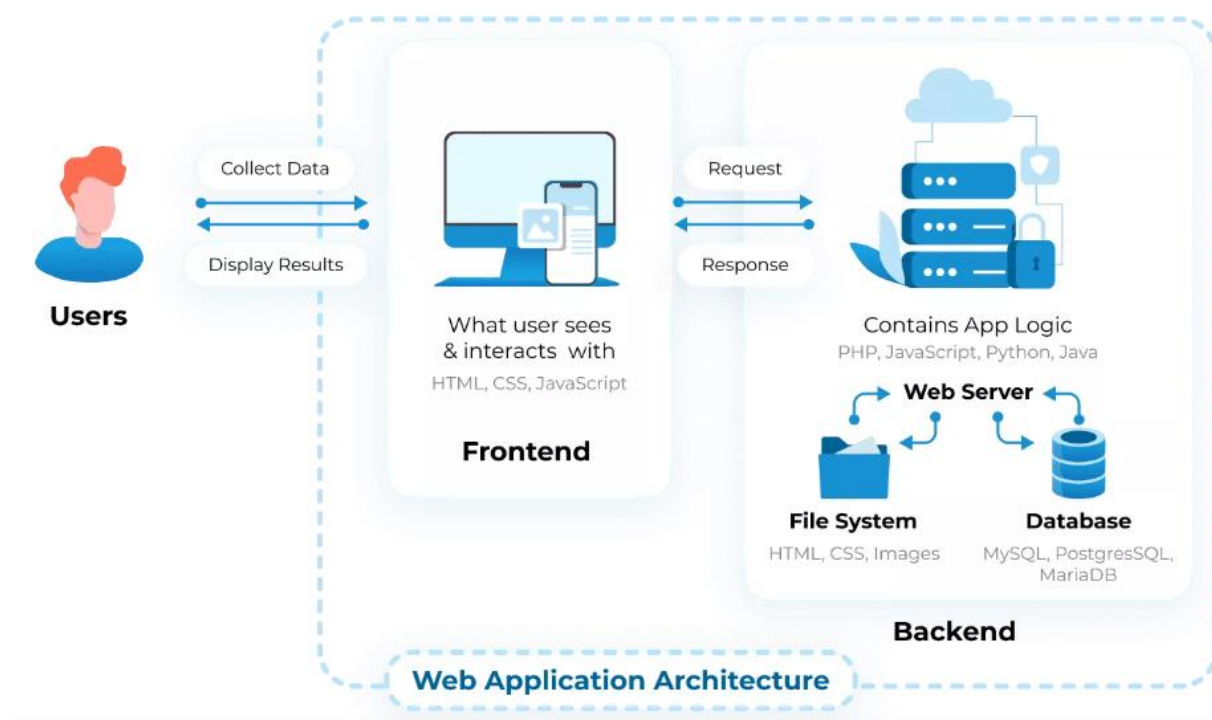


Рисунок 1.2 – Схема взаємодію користувача з веб додатком

З архітектурою програмного забезпечення легше побачити загальну картину. Основна мета цього — визначити функціональні вимоги та вимоги до якості та впоратися з ними для покращення загальної якості програми. Отже, загалом, за допомогою архітектури програмного забезпечення ви можете контролювати продуктивність, масштабованість і надійність.

Дизайн програмного забезпечення стосується дизайну на рівні коду, і він відповідає за функціональність кожного модуля та його цілі. Це «як» процесу розробки програмного забезпечення. Після того, як ви пройшли етап архітектури, настав час для розробника програмного забезпечення подумати про функції, класи, інтерфейси та інші деталі програми. Дизайн програмного забезпечення – це, скажімо, рівень деталей або компонентів. Наприклад, коли відбувається етап кодування, розробники зовнішнього та внутрішнього програмного забезпечення можуть працювати відповідно до цієї специфікації. Таким чином, за допомогою дизайну програмісти мають ефективну спільну мову, щоб знаходити рішення повторюваних проблем і концептуалізувати їх. Це

мінімізує обсяг роботи, яку вони виконують, оскільки немає потреби винаходити колесо.

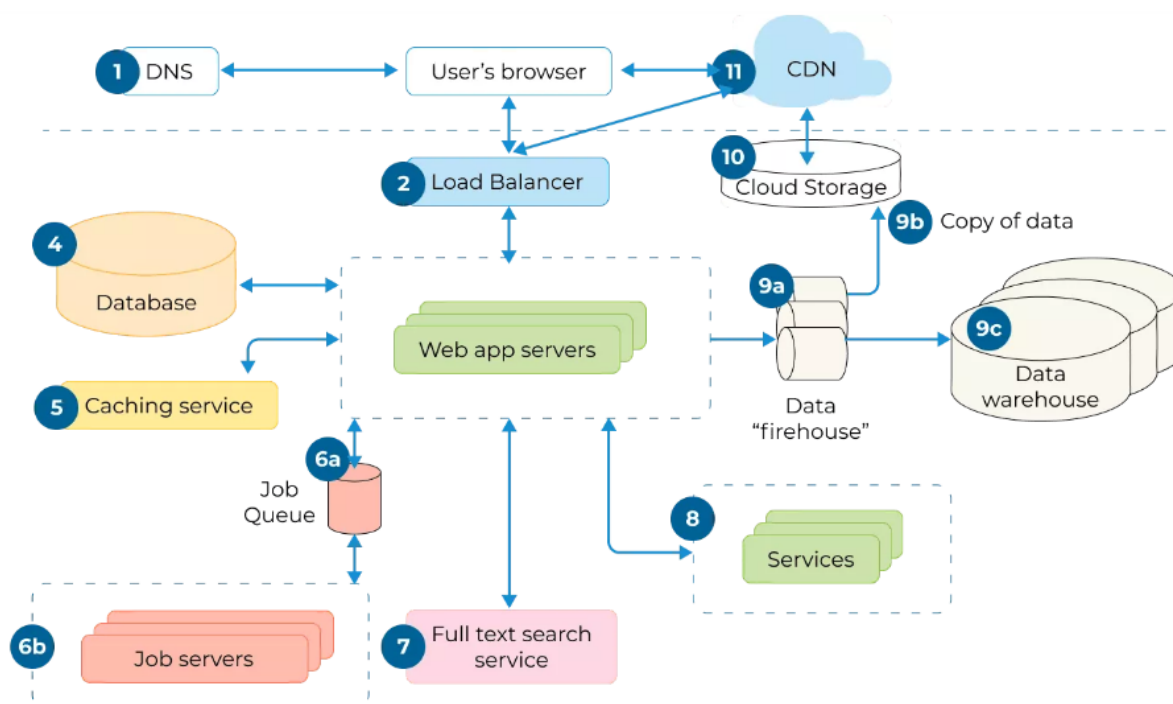


Рисунок 1.3 – Схема архітектури веб додатка

1.2 Переваги та недоліки Web-додатків

Переваги:

1) Низька вартість. Однією з найбільш привабливих переваг створення веб-додатків є фактор вартості. Послуги веб-розробки, необхідні для цього, набагато дешевші, ніж інші види веб-розробок. Це просто створення посилань між URL-адресою та програмою, і оскільки це відносно простіше зробити, розробка займає набагато менше часу. Таким чином, роблячи це загалом рентабельною справою для власника. Тут також допомагає легкість налаштування. Оскільки багатьом розробникам легше налаштувати веб-програми шляхом легкої зміни інтерфейсу програми, операції можна виконувати з меншими затратами часу та зусиль, що призводить до витрачання менше ресурсів.

2) Дані завжди в актуальному стані. Вони не вимагають частого оновлення, як зазвичай роблять звичайні програми. Саме веб-сайт/URL-адреса, на яку безпосередньо пов'язано програму, оновлюється до останньої версії. І оскільки всі мають доступ до однієї версії веб-програми через ту саму URL-адресу, усі користувачі завжди використовують найновішу та однакову її версію.

3) Не потрібно завантажувати та встановлювати додаткове ПЗ. Оскільки за допомогою веб-браузера користувач може безпосередньо взаємодіяти з програмою, її не потрібно встановлювати чи завантажувати окремо з різних платформ, таких як Google Play або Apple App store. Це також призводить до економії грошей, оскільки не потрібно нести жодних витрат за пряме посилання через веб-програму. Крім того, до веб-програм можна отримати доступ через кілька браузерів, а також працювати на кількох платформах, таких як ноутбуки, настільні комп'ютери чи мобільні телефони.

4) Кросплатформеність. Служби веб-дизайну для веб-програми за замовчуванням запрограмовані таким чином, щоб вони могли працювати в будь-якій операційній системі. Поки веб-браузер встановлено, їхній інтерфейс із різними розмірами екрана дозволяє легко адаптуватися до iOS, Android або Windows, серед багатьох інших.

Недоліки:

1) Необхідність стабільного інтернет-з'єднання. Незважаючи на те, що ми живемо в епоху Інтернету, втрата з'єднання з Інтернетом є досить поширеним явищем. І жоден Інтернет безпосередньо не призведе до втрати можливості запускати веб-програму. Таким чином, надійне підключення до Інтернету є обов'язковим у будь-який час для перегляду веб-сайту та запуску програми.

2) Працює більш повільно ніж ПЗ, що встановлено локально. Часто веб-програма працює відносно повільніше, ніж програма, розміщена на локальному сервері, і з цих причин не може повністю замінити мобільні програми. Він також безпосередньо пов'язаний з нашим браузером, через що його розмір програми має тенденцію до збільшення. Таким чином, велика програма значно повільніша за нативну настільну. Крім того, оскільки веб-програма повністю працює в

Інтернеті, вона часто може працювати повільніше через якість інтернет-з'єднання.

3) Більший низький рівень безпеки. Застосування протоколу SSL може допомогти зменшити ризик порушення даних, веб-додатки зазвичай не мають функції контролю якості. Таким чином, рівень безпеки порівняно знижується, що створює загрози для важливих і конфіденційних даних.

4) Обмежена функціональність. Рідна технологія завжди має перевагу над нерідною. Оскільки веб-програми не є нативними, іноді вони не можуть ефективно співпрацювати з усім обладнанням і операційними системами конкретних пристроїв, які ви використовуєте.

Незважаючи на різницю у вимогах, один факт залишається напевно: попит на веб-розробку зростає в різних галузях. Підприємства усвідомили необхідність адаптації до нинішнього часу, щоб успішно працювати та не залишатися позаду в довгостроковій перспективі.

1.3 Типи веб-розробки

Відомо три основні типи веб-розробки.

1. Фронтальна веб-розробка;

Інтерфейсна веб-розробка відома як клієнтська веб-розробка. Це процедура розробки інтерфейсу користувача веб-сайту за допомогою зовнішніх технологій, таких як HTML, CSS і JavaScript.

Мета розробки зовнішньої частини веб-сайту полягає в тому, щоб дозволити відвідувачам веб-сайту легко читати веб-вміст. Щоб мотивувати вас створити чистий і красивий інтерфейс веб-сайту.

2. Back-End веб-розробка;

Бекенд-розробка також відома як веб-розробка на стороні сервера. Обов'язки бекенд-розробників полягають у написанні коду та створенні логічних частин усього веб-сайту. В основному він зосереджений на внутрішній логіці, базах даних, серверах і API.

У веб-проекті логічна частина покладається на бекенд, ви можете не бачити технології, яка використовується для розробки веб-сайту. Однак бекенд веб-розробка є важливою частиною веб-сайту.

3. Повний пакет веб-розробок.

Повноцінна веб-розробка означає розробку передньої та задньої частини. Повноцінний веб-розробник повинен знати веб-дизайн, веб-розробку, базу даних і налагодження веб-сайтів.

Повноцінний розробник має досвід розробки клієнтських і серверних веб-додатків, а також має мати навички керування базами даних. Повноцінні веб-розробники добре знаються на перепроєктуванні, створенні та прискоренні всього етапу дизайну та розробки веб-сайту.

Крім того, обов'язки повнофункціональних веб-розробників полягають у пошуку тенденцій веб-розробки, таких як блокчейн, глибоке навчання та мультимарність залежно від проектів.

1.4 Типи веб-додатків

Веб-додаток[3] — це програма або програмне забезпечення, надане третьою стороною, яке зберігається на віддаленому сервері та доступне з будь-якого веб-браузера з будь-якого пристрою. Веб-сайти мають переважно інформаційний характер. Ви можете отримати доступ до багатьох документів на веб-сайтах через Інтернет за допомогою веб-браузера. Він також складається з веб-додатків, які допомагають користувачам виконувати різні онлайн-завдання, такі як пошук, перегляд і оплата.

Статичний веб-додаток. Найпершим типом веб-програм, доступних в Інтернеті, є статичні веб-програми, створені з використанням HTML і CSS для полегшення показу значного вмісту та інформації. Зазвичай це найпростіша веб-програма, оскільки вона демонструє лише обмежений вміст і не є гнучкою. Зазвичай ці програми не мають персоналізації та вносять зміни після повного завантаження сторінки. Незважаючи на те, що він дозволяє анімовані об'єкти,

такі як GIF-файли, відео тощо, змінити вміст статичної веб-програми нелегко, оскільки це вимагає завантаження, зміни та повернення HTML-коду. Тому ця програма найкраще підходить для компаній, що займаються розробкою програмного забезпечення, і професійних веб-майстрів. Деякі з найкращих прикладів включають цифрові резюме та сторінки для захоплення потенційних клієнтів у маркетингу.

Динамічний веб-додаток надає живі дані на основі запитів користувачів і тому вважається одним із найкращих типів веб-додатків. Порівняно зі статичними веб-додатками вони мають кращу технічну складність. Існує кілька елементів взаємодії та методів привернення уваги клієнта до послуг і продуктів, які надає веб-програма. Такі веб-додатки використовують бази даних для зберігання всіх приватних і публічних даних, які відображаються на Веб-сайті. Зазвичай вони мають панель адміністратора для керування серверною та зовнішньою частинами, а також дозволяють адміністратору змінювати вміст і включати різні інтерактивні компоненти до веб-програми. Динамічна веб-програма створена з використанням різних мов програмування, таких як PHP і ASP.NET.

Односторінкові програми, також відомі як SPA, — це тип динамічних веб-програм, які не потребують перезавантаження веб-переглядача та функціонують як окремий блок веб-програми. Ці веб-програми є швидкими та динамічними, оскільки вони реалізують усі бізнес- та технологічні стратегії в клієнтському браузері. Процес розробки та впровадження SPA простий і швидкий. Оскільки зв'язок відбувається в асинхронній навігації, процес обробки запиту та відповіді користувача відбувається швидше. Крім того, веб-додаток SPA будь-якого типу можна переналаштувати для досягнення бажаних результатів. Однак головна проблема SPA полягає в тому, що вони не відповідають рекомендаціям SEO. Найкращими прикладами односторінкових веб-додатків є Netflix, Twitter і Gmail.

Веб-додаток електронної комерції. Багато основних функцій веб-програми для електронної комерції включають додавання нових продуктів, видалення застарілих і старих продуктів, керування платежами, полегшення електронних

платежів і зручний інтерфейс. Для вирішення всіх цих завдань дуже потрібна ефективна панель управління. Професійні розробники веб-сайтів можуть налаштувати такі програми, щоб зробити їх зручними для користувачів. Деякі з найпоширеніших прикладів веб-додатків для електронної комерції включають Flipkart, Amazon, Ajo, і список можна продовжувати.

Веб-програма порталу пропонує єдину точку доступу до важливих даних для певного типу користувачів. Це веб-програма, яка може отримувати доступ до різних розділів домашньої сторінки. Портالي є найкращим варіантом для організацій і компаній, які віддають перевагу створенню індивідуальних інтерфейсів відповідно до потреб своєї цільової аудиторії. Лише зареєстровані користувачі мають доступ, і постачальник послуг може контролювати дії користувача після входу користувача.

Система керування вмістом (CMS) — це тип веб-додатку, у якому власник може змінювати вміст без допомоги технічної групи. Контент можна змінювати за допомогою панелі адміністратора та без знання мови програмування. Загалом, існує кілька варіацій CMS з різними характеристиками та дизайном, далі наведено деякі з яких.

WordPress – це найбільш широко використовувана платформа великими власниками підприємств, які змінили свій бізнес в Інтернеті. У WordPress ви можете знайти різні функції, такі як теми, плагіни та навчальні посібники, які допоможуть вам зробити ваш веб-сайт привабливим і виразним. Крім того, для цього не потрібна будь-яка технічна допомога ззовні.

Joomla – це після WordPress і вважається другим за популярністю серед користувачів. Хоча у нього небагато активних користувачів, які включили цю платформу у свій бізнес, у нього хороша спільнота та інтуїтивно зрозуміла вартість старої, але вдосконаленої веб-програми.

Drupal – Drupal – це безкоштовна веб-програма з відкритим вихідним кодом, яка є достатньо гнучкою, щоб створити власний веб-сайт. Це спеціально рекомендовано для створення великих порталів спільноти.

Анімовані веб-додатки. Ці програми дозволяють відображати ваш вміст разом із ефектами анімації. Ці типи програм пропонують креативність і дизайн, яких немає в інших типах веб-програм. Проблема полягає в тому, що він не дуже підходить для веб-позиціонування та SEO, оскільки дані, отримані пошуковими системами, неможливо прочитати.

Багатофункціональні веб-програми інтернету (RIA) — це головним чином програми, які мають функціональність багатьох настільних програм. Вони призначені для усунення обмежень браузера та покладаються на плагіни на стороні клієнта. Ця веб-програма була створена з більш ефективними та візуально привабливими ресурсами з більш інтерактивним інтерфейсом користувача та кращим розумінням, ніж старіші настільні програми. Деякі веб-додатки RIA також можуть працювати в автономному режимі. Двома основними проблемами АРВ є ризики та незручності, які вони спричиняють. Наприклад, якщо плагін застарів, багато частин програми або вся програма можуть не працювати належним чином.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Структура та технології реалізації

Маркетинг має свою специфіку, а тому процес до переходу використання додатку може бути складним та довготривалим. При реально застосуванні процес переходу має бути плавним з переходом від тестування окремих модулів до відладки системи.

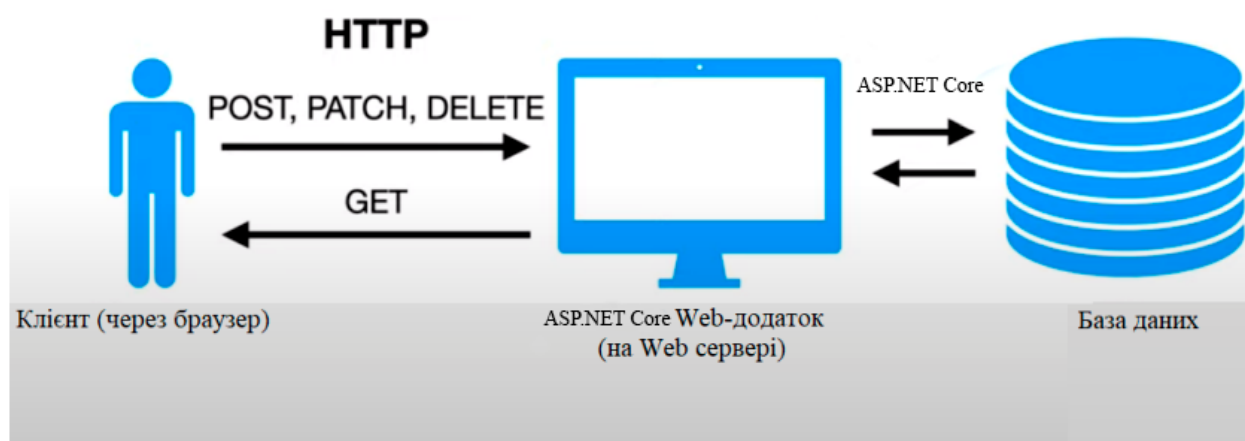


Рисунок 2.1 – Структура додатку, що має бути реалізована

Для реалізації додатку, було обрано мову програмування С#. С# — це сучасна об'єктно-орієнтована мова програмування загального призначення, розроблена Microsoft і схвалена Європейською асоціацією виробників комп'ютерів (ЕСМА) і Міжнародною організацією стандартів (ISO). С# розроблено для спільної мовної інфраструктури (CLI), яка складається з виконуваного коду та середовища виконання, що дозволяє використовувати різні мови високого рівня на різних комп'ютерних платформах і архітектурах. Зокрема привабливою для використання цю мову роблять безліч патернів та фреймворків, одним з таких гігантських фреймворків є ASP.NET Core[12].

ASP.NET Core набагато швидший за ASP.NET MVC. Новий модульний конвеєр HTTP-запитів покращує можливості кешування, а підтримка кількох

архітектур ЦП покращує швидкість роботи фреймворку. Крім того, ASP.NET Core[10] має вбудовану підтримку кешування та стиснення вмісту. Ці функції дозволяють доставляти файли HTML, CSS і JavaScript, які мають менший розмір і швидше завантажуються у браузері. ASP.NET Core MVC вимагає менше коду, ніж ASP.NET MVC. Частково це пов'язано з тим, що багато функцій, які раніше були реалізовані в ASP.NET MVC, наприклад авторизація та стан сеансу, були переміщені до компонентів проміжного програмного забезпечення. Крім того, шаблони проектів за замовчуванням для ASP.NET Core набагато простіші, ніж у MVC[5]. Це зменшує кількість коду, який потрібно написати розробникам NET Core, щоб створити та запустити базову веб-програму.

Безпека є головним пріоритетом під час розробки будь-якого типу веб-додатків. А ASP.NET Core дозволяє вам це робити, дозволяючи підтримувати примусове виконання HTTPS. Основним примусовим засобом HTTP в ASP.NET Core є вбудована система авторизації. Ця система дозволяє детально контролювати, хто має доступ до ресурсів вашого сайту. Ви також можете використовувати цю систему для створення настроюваних ролей і дозволів, які відповідають вашим конкретним потребам. Крім того, структура ASP.NET Core надає кілька способів захисту веб-додатків, включаючи HTTPS/SSL (рівень захищених сокетів) і різні алгоритми шифрування. Крім того, програми ASP.NET Core можуть скористатися багатьма функціями безпеки, що надаються платформою Microsoft .NET, такими як безпека доступу до коду та безпека на основі ролей. До того ж такий фреймворк має безліч додаткових компонентів які розробник може додавати за необхідності, так компонентами при розробці стали Entity та Identity. Для роботи з базою даних використовується Microsoft SQL Client.

Entity Framework (EF) Core[11] — це найновіша технологія даних, створена командою розробників ASP.NET . Ця конкретна структура Entity є рекомендованою структурою для створення програм ASP.NET Core . Entity Framework Core або EF Core класифікується як Object Relational Mapper (ORM); це означає, що можна позбутися складнощів перетворення даних із реляційного

сховища та об'єктно-орієнтованої моделі домену вашої програми, а з використанням патерна CodeFirst не потрібно власноруч створювати таблиці в БД, достатньо правильно описати модель та провести міграцію і таблиці в БД будуть створені автоматично. Крім того в подальшому при роботі з БД в контролері це полегшить підключення до БД та отримання необхідних даних.

Entity Framework представив підхід Code-First у Entity Framework 4.1. Code-First в основному корисний у доменно-орієнтованому проектуванні. У підході Code-First ви зосереджуєтеся на домені вашої програми та починаєте створювати класи для сутності домену, а не спочатку проектуєте базу даних, а потім створюєте класи, які відповідають дизайну вашої бази даних. Наступний малюнок ілюструє підхід «перший код».

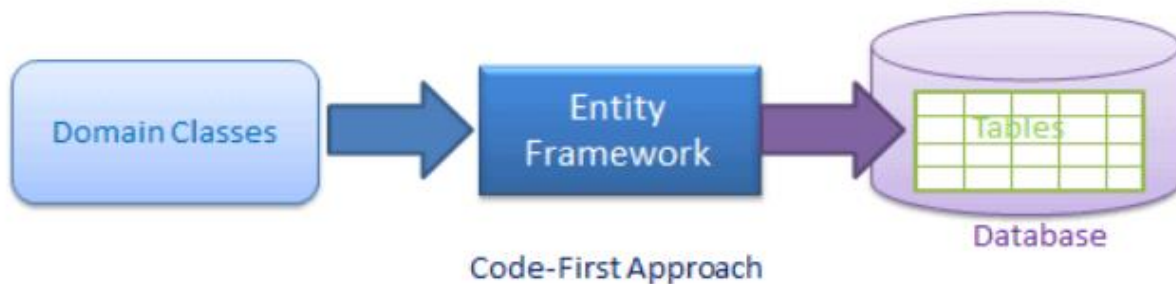


Рисунок 2.2 – Схема роботи Code-First

Як видно на рисунку 2.2, Entity Framework API створить базу даних на основі класів і конфігурації вашого домену. Це означає, що потрібно спочатку почати кодування на C# або VB.NET, а потім EF створить базу даних із коду.



Рисунок 2.3 – Робочий процес Code-First\

Робочий процес розробки в підході, який базується на першому коді, буде таким: створити або змінити класи домену -> налаштувати ці класи домену за

допомогою Fluent-API або атрибутів анотації даних -> створити або оновити схему бази даних за допомогою автоматизованої міграції або міграції на основі коду.

Міграції бази даних, також відомі як міграції схем, міграції схем бази даних або просто міграції, — це контрольовані набори змін, розроблені для зміни структури об'єктів у реляційній базі даних. Міграції допомагають перевести схеми баз даних із поточного стану в новий бажаний стан, незалежно від того, чи це передбачає додавання таблиць і стовпців, видалення елементів, розбиття полів або зміну типів і обмежень.

Міграції керують поступовими, часто оборотними, змінами структур даних програмним способом. Цілі програмного забезпечення для міграції бази даних — зробити зміни бази даних повторюваними, доступними для спільного використання та тестування без втрати даних. Як правило, програмне забезпечення для міграції створює артефакти, які описують точний набір операцій, необхідних для перетворення бази даних із відомого стану в новий. Їх можна перевіряти та керувати за допомогою звичайного програмного забезпечення для контролю версій, щоб відстежувати зміни та ділитися між членами команди.

Хоча запобігання втраті даних, як правило, є однією з цілей програмного забезпечення для міграції, зміни, які руйнують або деструктивно змінюють структури, у яких зараз містяться дані, можуть призвести до видалення. Щоб впоратися з цим, міграція часто є контрольованим процесом, який передбачає перевірку отриманих сценаріїв змін і внесення будь-яких змін, необхідних для збереження важливої інформації.

Міграції корисні, оскільки вони дозволяють схемам бази даних розвиватися відповідно до змін вимог. Вони допомагають розробникам планувати, перевіряти та безпечно застосовувати зміни схеми до свого середовища. Ці розділені зміни визначаються на детальному рівні й описують перетворення, які мають відбутися для переходу між різними «версіями» бази даних.

Загалом, системи міграції створюють артефакти або файли, якими можна спільно користуватися, застосовувати до кількох систем баз даних і зберігати в системі керування версіями. Це допомагає побудувати історію змін до бази даних, яка може бути тісно пов'язана із супутніми змінами коду в клієнтських програмах. Схема бази даних і припущення програми щодо цієї структури можуть розвиватися паралельно.

Деякі інші переваги включають можливість (а іноді й необхідність) вручну налаштовувати процес, відокремлюючи створення списку операцій від їх виконання. Кожну зміну можна перевірити, перевірити та модифікувати, щоб забезпечити отримання правильних результатів, покладаючись на автоматизацію більшої частини процесу. Системи міграції не позбавлені недоліків, але, більшість із них можна пом'якшити за допомогою процесів і нагляду.

Оскільки міграції змінюють існуючі структури бази даних, необхідно бути обережним, щоб уникнути втрати даних. Це може бути викликано неправильними припущеннями, зробленими інструментами, або перетвореннями, які вимагають розуміння значення структур даних. Хоча може виникнути спокуса припустити, що створені файли міграції є правильними, оскільки файли міграцій стосуються в основному структур даних, ви несете відповідальність за те, щоб дані також були збережені або перетворені належним чином.

Міграції також можуть пройти погано, якщо вони застосовані до бази даних, яка перебуває в іншому стані, ніж передбачається. Наприклад, це може статися, коли в структуру бази даних вносяться зміни поза межами системи міграції або коли міграції застосовуються в неправильному порядку. Усі системи міграції покладаються на розуміння поточного стану бази даних, щоб правильно змінити існуючі структури. Якщо фактичний стан відрізняється від припущеного, міграція може завершитися невдачею або може змінити базу даних небажаним чином.

Ще одна потенційна проблема з міграціями полягає в тому, що вони часто залежать від інструментів. Системи міграції створюють артефакти, які

представляють стан баз даних або необхідні зміни. Хоча деякі інструменти видають результати у звичайному SQL, іноді вони кодують додаткові деталі для аналізу інструментом у коментарях або можуть використовувати спеціальні формати імен файлів для вказівки порядку. Перехід між інструментами міграції може бути складним без чіткого розриву між попереднім і поточним поколіннями файлів міграції.

ASP.NET Core Identity[14] використовується для реалізації автентифікації форм. Існує багато варіантів ідентифікації ваших користувачів, включаючи автентифікацію Windows і всі сторонні постачальники ідентифікаційної інформації, такі як Google, Microsoft, Facebook і GitHub тощо. Цей фреймворк дозволяє нам додавати функції, де користувачі можуть реєструватися та входити за допомогою локального пароля. Фреймворк також підтримує двофакторну автентифікацію, сторонніх постачальників ідентифікаційної інформації та інші функції.

До стандартних полів також можна додавати безліч додаткових якщо це необхідно. Безумовно зручністю використання такого фреймворку є те що розробнику не потрібно проектувати та розробляти систему захисту даних, так як вона вже реалізована, а вся відповідальність у разі витоку даних лягає на компанію Microsoft.

Microsoft SQL Server[4] — це система керування реляційною базою даних (RDBMS), яка підтримує широкий спектр програм обробки транзакцій, бізнес-аналітики та аналітики в корпоративних IT-середовищах. Microsoft SQL Server є однією з трьох провідних на ринку технологій баз даних разом із Oracle Database і IBM DB2 .

SQL Server в першу чергу побудовано навколо структури таблиці на основі рядків, яка з'єднує пов'язані елементи даних у різних таблицях один з одним, уникаючи необхідності надмірного зберігання даних у кількох місцях у базі даних. Реляційна модель[6] також забезпечує посилову цілісність та інші обмеження цілісності для підтримки точності даних. Ці перевірки є частиною ширшого дотримання принципів атомарності, узгодженості, ізоляції та

довговічності, спільно відомих як властивості ACID , і призначені для гарантії надійної обробки транзакцій бази даних.

Основним компонентом Microsoft SQL Server є SQL Server Database Engine, який контролює зберігання, обробку та безпеку даних. Він включає реляційний механізм, який обробляє команди та запити, і механізм зберігання, який керує файлами бази даних, таблицями, сторінками, індексами, буферами даних і транзакціями. Збережені процедури, тригери, подання та інші об'єкти бази даних також створюються та виконуються за допомогою Database Engine.

Під обробником баз даних знаходиться операційна система SQL Server або SQLOS. SQLOS обробляє функції нижчого рівня, такі як керування пам'яттю та введенням/виведенням, планування завдань і блокування даних, щоб уникнути конфліктних оновлень. Рівень мережевого інтерфейсу розташований над Database Engine і використовує протокол потоку табличних даних Microsoft для полегшення взаємодії запитів і відповідей із серверами баз даних. А на рівні користувача адміністратори баз даних і розробники SQL Server пишуть оператори T-SQL для створення та модифікації структур бази даних, маніпулювання даними, впровадження засобів захисту та резервного копіювання баз даних, серед інших завдань.

SQL Server 2019 дозволяє користувачам об'єднувати контейнери SQL Server, HDFS і Spark за допомогою нової функції кластера великих даних. SQL Server 2019 також представляє збірки індексів columnstore, перебудови та маскування статичних даних. Також є новинка Accelerated Data Recovery, яка виконує та скасовує етап повторення найстарішої сторінки з порядковим номером журналу. Як приклад, це робиться у випадку, коли користувач закриває програму, яка працювала протягом тривалого періоду часу, тому користувачеві не потрібно довго чекати, поки програма закриється.

Групи доступності Always On, доступні в SQL Server 2012, змінено, щоб спростити адміністрування груп доступності. Це додає підтримку баз даних MSDB і Master system. Інші зміни в функціях включають розширення операцій, які користувачі можуть виконувати з завжди зашифрованими даними; додаткові

конектори Polybase для SQL Server, Oracle, MongoDB і Teradata; додаткові параметри постійної пам'яті для зберігання; і вдосконалення обробки запитів.

Розширені функції безпеки, які підтримуються в усіх версіях Microsoft SQL Server, починаючи з SQL Server 2016 SP1, включають три технології, додані до випуску 2016: Always Encrypted, яка дозволяє користувачеві оновлювати зашифровані дані без необхідності розшифровувати їх, захист на рівні першого рядка, що вмикає дані контроль доступу на рівні рядків у таблицях бази даних; і динамічне маскуванню даних, яке автоматично приховує елементи конфіденційних даних від користувачів без повних привілеїв доступу.

Інші важливі функції безпеки SQL Server включають прозоре шифрування даних, яке шифрує файли даних у базах даних, і детальний аудит, який збирає детальну інформацію про використання бази даних для звітування про відповідність нормативним вимогам. Microsoft також підтримує протокол безпеки транспортного рівня для захисту зв'язку між клієнтами SQL Server і серверами баз даних.

Більшість із цих інструментів та інших функцій Microsoft SQL Server також підтримуються в базі даних Azure SQL, хмарній службі баз даних, створеній на платформі SQL Server Database Engine. Крім того, користувачі можуть запускати SQL Server безпосередньо на Azure за допомогою технології під назвою SQL Server на віртуальних машинах Azure; він налаштовує СУБД у віртуальних машинах Windows Server, що працюють на Azure. Пропозиція віртуальної машини оптимізована для міграції або розширення локальних додатків SQL Server до хмари, тоді як база даних Azure SQL розроблена для використання в нових хмарних програмах.

У хмарі Microsoft також пропонує Azure SQL Data Warehouse, службу сховища даних, засновану на реалізації SQL Server з масовою паралельною обробкою (MPP). Версія MPP, яка спочатку була автономним продуктом під назвою SQL Server Parallel Data Warehouse, також доступна для локального використання як частина платформи Microsoft Analytics Platform System, яка поєднує її з PolyBase та іншими технологіями великих даних.

Як і інше програмне забезпечення RDBMS , Microsoft SQL Server побудовано на основі SQL , стандартизованої мови програмування, яку адміністратори баз даних (DBA) та інші IT-фахівці використовують для керування базами даних і запитів до даних, які вони містять. SQL Server пов'язано з Transact-SQL (T-SQL), реалізацією SQL від Microsoft, яка додає набір власних розширень програмування до стандартної мови.

Набір функцій використання озділені за ролями користувачів. Роль визначає адміністратор. Для доступу до функціоналу користувач має авторизуватися. Нижче наведено основні функції які є в додаткові. Також прямого доступу до реєстрації не має, якщо в компанії з'являється новий співробітник то адміністратор має зареєструвати для нього акаунт.

Таблиця 2.1 – Основні функції додатку

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-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

Функціонал реалізований таким чином що користувач може редагувати наявні дані, а всі глобальні зміни такі як видалення чи додавання нових даних може здійснювати лише адміністратор.

2.2 Проектування архітектури додатку

Будь-яка архітектура програми – це опис основних компонентів та способів їх взаємодії між собою. Або ж це певно структуризація програмної системи, іншими словами це абстракція елементів системи на певному етапі її роботи.

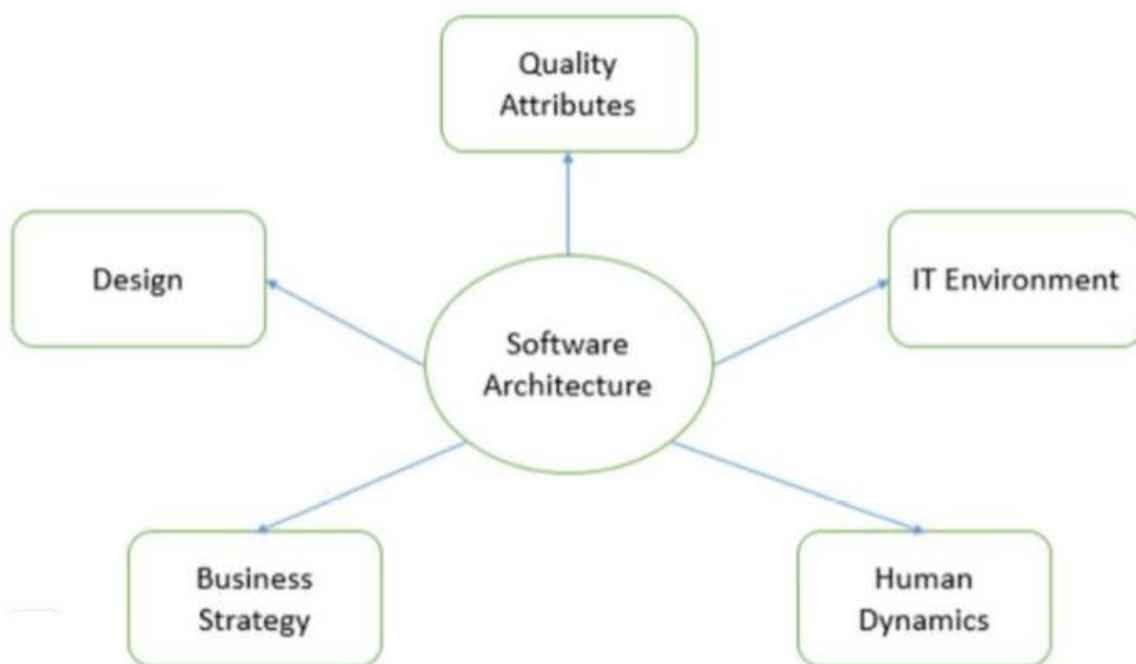


Рисунок 2.4 – Основні складові архітектури ПЗ

Архітектура програми описує шаблони та методи, які використовуються для проектування та створення програми. Архітектура дає вам дорожню карту та найкращі методи, яких слід дотримуватися під час створення програми, щоб ви отримали добре структуровану програму. Шаблони проектування програмного забезпечення можуть допомогти створити програму. Шаблон описує повторюване рішення проблеми.

Патерни можна зв'язувати разом для створення більш загальних архітектур додатків. Замість того, щоб повністю створювати архітектуру самостійно, ви можете використовувати існуючі шаблони проектування, які також гарантують, що все буде працювати так, як вони повинні працювати.

Як частина архітектури програми, існуватимуть як зовнішні, так і внутрішні служби. Фронтальна розробка пов'язана з користувальницьким досвідом роботи програми, тоді як бек-енд розробка зосереджена на наданні доступу до даних, послуг та інших існуючих систем, які забезпечують роботу програми.

Архітектура є відправною точкою або дорожньою картою для створення програми, але вам потрібно буде зробити вибір реалізації, який не враховано в архітектурі. Наприклад, першим кроком є вибір мови програмування, на якій буде написано додаток.

Для розробки програмного забезпечення використовується багато мов програмування. Певні мови можуть використовуватися для створення певних типів програм, наприклад Swift для мобільних програм або JavaScript для інтерфейсної розробки. JavaScript, який використовується з HTML і CSS, наразі є однією з найпопулярніших мов програмування для розробки веб-додатків.

Серед інших популярних мов програмування – Ruby, Python, Swift, TypeScript, Java, PHP і SQL. Мова, яка використовується під час створення програми, залежатиме від типу програми, доступних ресурсів розробки та вимог.

Модель «Клієнт-сервер» — це розподілена структура програми, яка розподіляє завдання або робоче навантаження між постачальниками ресурсів або послуг, які називаються серверами, і запитувачами послуг, які називаються клієнтами (рис. 2.5). В архітектурі клієнт-сервер, коли клієнтський комп'ютер надсилає запит на дані серверу через Інтернет, сервер приймає запитуваний процес і доставляє запитувані пакети даних назад клієнту. Клієнти не діляться своїми ресурсами. Прикладами клієнт-серверної моделі є електронна пошта, Всесвітня павутина тощо. Архітектурний шаблон MVC перетворює розробку складних додатків на набагато легший процес. Це дозволяє кільком розробникам одночасно працювати над програмою.

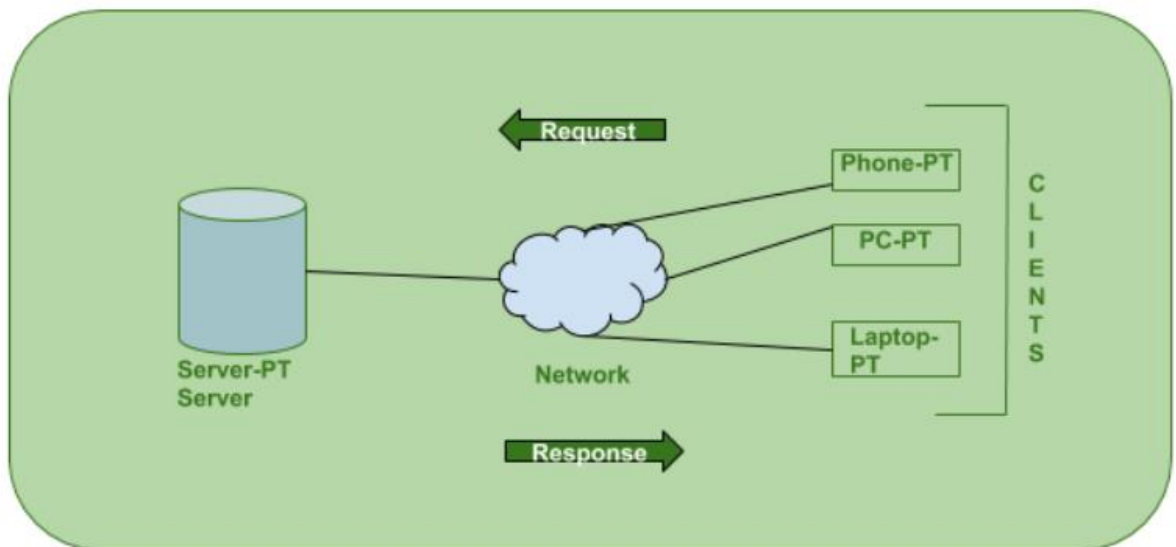


Рисунок 2.5 – Модель архітектури клієнт-сервер

MVC розшифровується як model-view-controller. Ось значення кожного з цих компонентів (рис. 2.6):

- Модель - серверна частина, яка містить усю логіку даних;
- Перегляд - інтерфейс або графічний інтерфейс користувача;
- Контролер – мозок програми, яка контролює відображення даних.

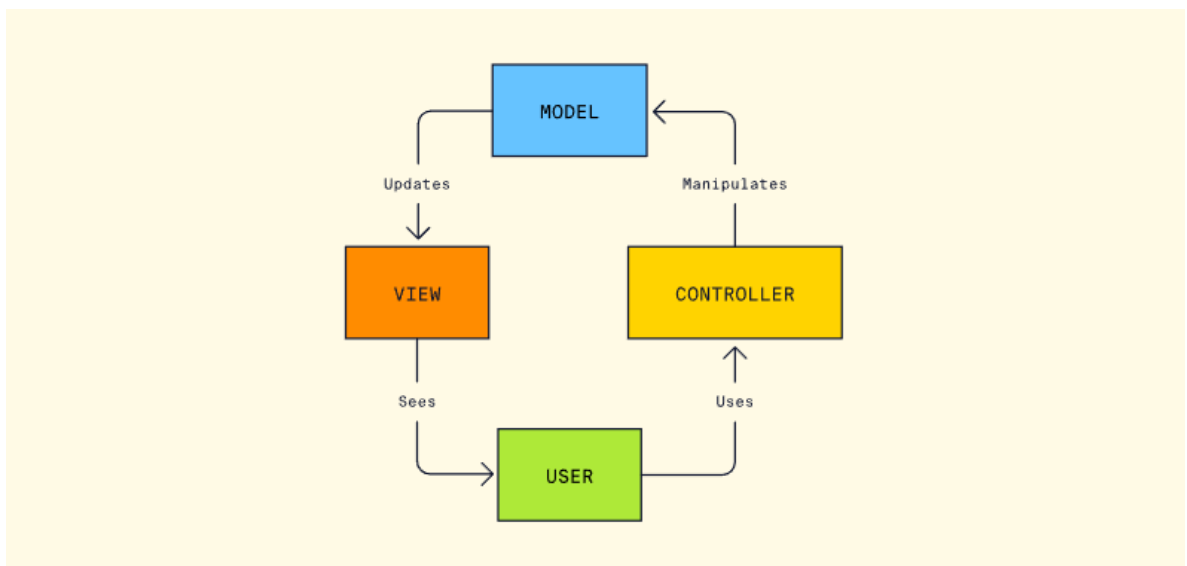


Рисунок 2.6 – Приклад роботи MVC архітектури

2.3 Проектування бази даних

Проектування бази даних – це один з найважливіших етапів створення додатку, так як тут визначаються основні схеми та необхідність цілісності. Існує два підходи до розробки будь-якої бази даних: метод «зверху вниз» і метод «знизу вгору». Хоча ці підходи виглядають кардинально різними, вони мають спільну мету використання системи шляхом опису всіх взаємодій між процесами.

Метод проектування зверху вниз починається від загального і переходить до конкретного. Іншими словами, ви починаєте із загального уявлення про те, що потрібно для системи, а потім переходите до більш конкретних деталей того, як система буде взаємодіяти. Цей процес передбачає ідентифікацію різних типів сутності та визначення атрибутів кожної сутності.

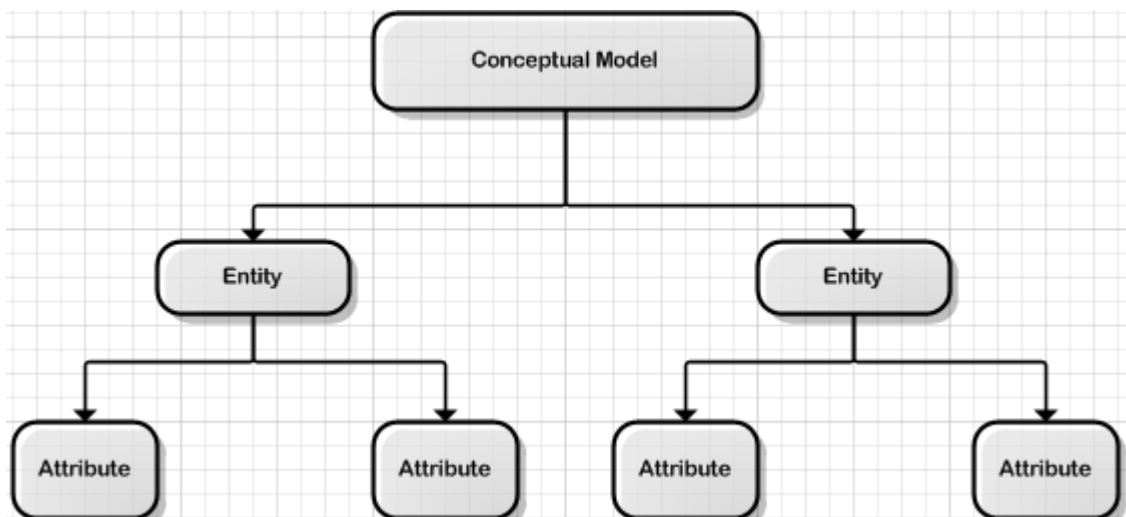


Рисунок 2.7 – Схема методу проектування «зверху в низ»

Підхід «знизу вверху» починається з конкретних деталей і просувається до загального. Це робиться шляхом спочатку ідентифікації елементів даних (елементів), а потім групування їх у набори даних. Іншими словами, цей метод спочатку ідентифікує атрибути, а потім групує їх, щоб сформувати сутності.

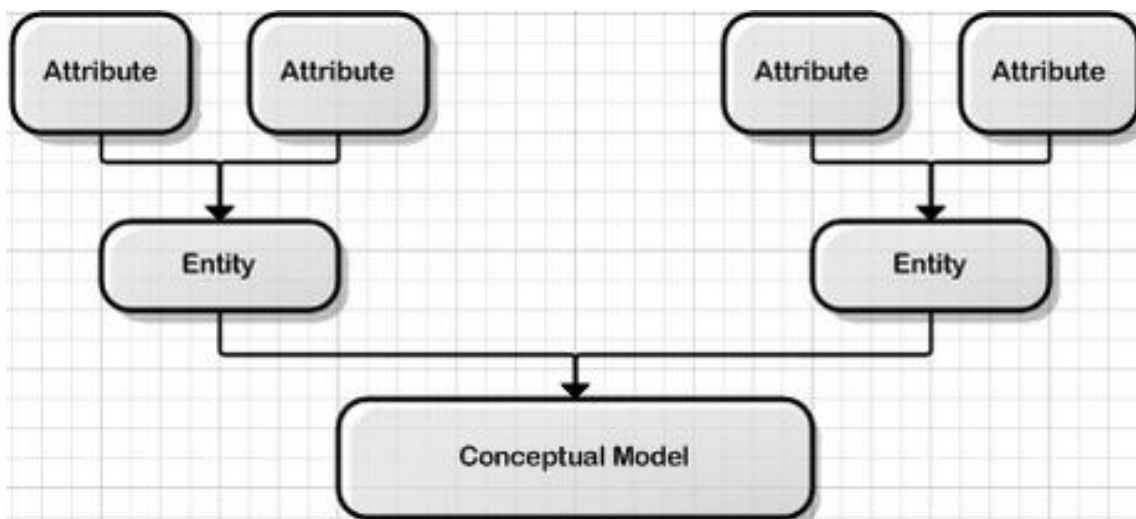


Рисунок 2.8 – Схема методу проектування «знизу в верх»

Нижче буде наведено схему бази даних додатку, як зазначалося вище для її створення було застосовано фреймворк Entity з використанням патерну CodeFirst, що дозволяє автоматизувати цей процес, тобто нам необхідно описати моделі додатку і провести міграцію даних.

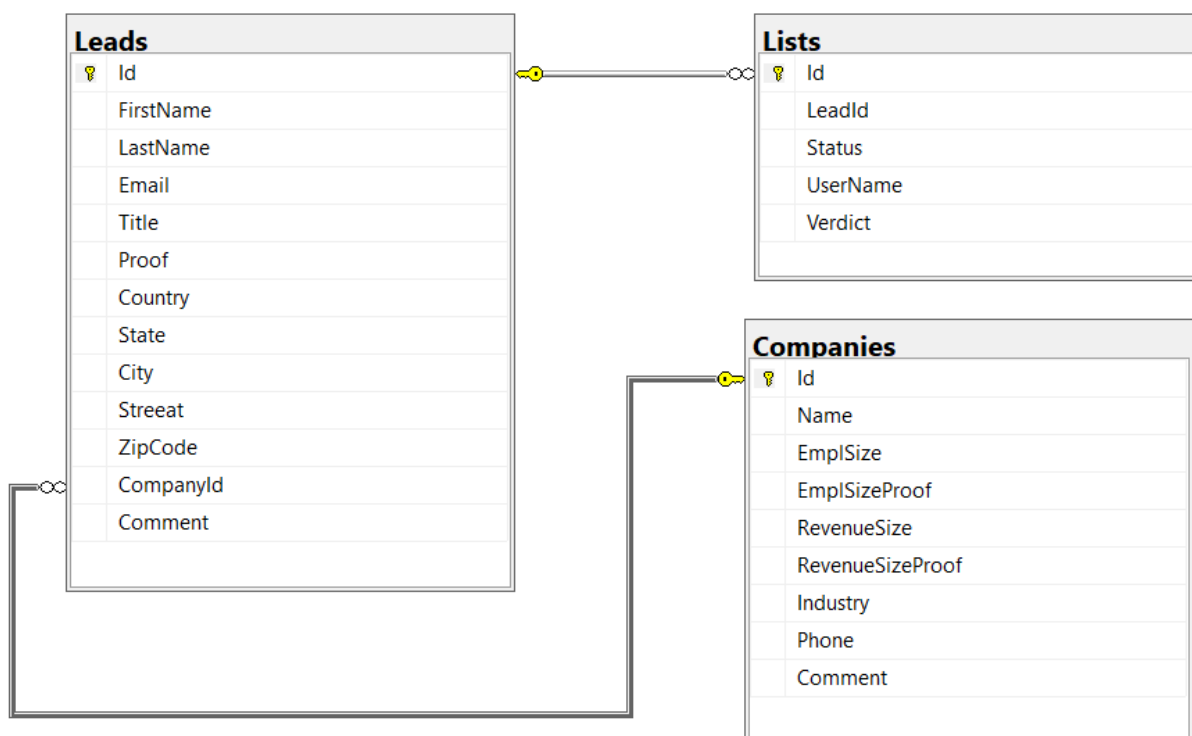


Рисунок 2.9 – Структура БД

РОЗДІЛ 3

РОЗРОБКА WEB-ДОДАТКУ

3.1 Вибір платформ

Система яку необхідно реалізувати – це веб-додаток, з наведеної вище інформації та тенденцій серед сучасних мов програмування було вибрано наступні інструменти: мова програмування C#, Framework ASP.NET Core MVC.

ASP.Net Core MVC — це кросплатформна структура розробки веб-додатків, яка використовує переваги середовища виконання ASP.Net Core і водночас дає змогу розробляти програми, які можна тестувати та підтримувати, а також розробляти та розгортати на кількох платформах. Зауважую, що MVC Core не потребує IIS для розміщення — ми можемо розміщувати програми MVC Core у Kestrel або їх навіть можна розміщувати самостійно. ASP.Net MVC Core є відкритим вихідним кодом, має вбудовану підтримку впровадження залежностей і є розширюваним. Також, структура MVC допомагає вам ізолювати проблеми у ваших програмах і створювати програми, які легше тестувати та підтримувати.

Основні функції в MVC Core включають маршрутизацію, прив'язку моделі, перевірку моделі, впровадження залежностей, фільтри, області, веб-інтерфейси API, суворо типізовані представлення, помічники тегів і компоненти перегляду. Тепер коротко розглянемо кожну з цих функцій.

Механізм маршрутизації ASP.Net Core MVC побудований на основі механізму маршрутизації ASP.Net Core. Тобто ми маємо підтримку маршрутизації двома різними способами: функція маршрутизації на основі умов і функція маршрутизації на основі атрибутів. У першому випадку ми можемо визначити формати URL-адрес для своєї програми глобально.

Також можливо прикрасити свої об'єкти моделі за допомогою атрибутів для виконання перевірки моделі в коді ASP.Net MVC. У наведеному нижче

фрагменті коду показано, як ви можете скористатися перевагами анотацій даних для прикраси своєї моделі.

Оскільки ASP.Net MVC Core побудовано на основі ASP.Net Core, він також успадковує можливості впровадження залежностей ASP.Net Core. В ASP.Net Core вбудовано підтримку ін'єкції залежностей і структуру пошуку служб. Є чотири режими, в яких можна вводити тип. До них належать: Singleton, Scoped, Transient та Instance.

ASP.Net MVC Core дозволяє впроваджувати залежності за допомогою конструкторів у класи контролерів. Ви також можете вставити залежності у файли перегляду за допомогою директиви `@inject`.

ASP.Net Core MVC забезпечує підтримку строго типізованих представлень. Таким чином, наші погляди «бритви» також можуть бути строго типізовані.

Помічники тегів використовуються для створення коду на сервері та відтворення елементів HTML. У вас є багато вбудованих помічників тегів у ASP.Net Core MVC. Ми також можете створити власний помічник тегів. Вбудовані помічники тегів можна використовувати для створення форм, завантаження ресурсів тощо.

ASP.Net MVC Core забезпечує чудову підтримку створення легких служб за допомогою веб-API, які можуть працювати через HTTP. ASP.Net Web API — це структура, яку можна використовувати для створення легких веб-служб, які використовують HTTP як протокол. Веб-API забезпечує вбудовану підтримку узгодження вмісту, засобів форматування та обміну ресурсами між джерелами.

За допомогою MVC Core тепер ми можете створювати та налаштовувати свої додатки для роботи в хмарі. Створення та розгортання програми для хмари тепер безперебійно завдяки чудовій підтримці конфігурації на основі середовища. По суті, тепер у нас є підтримка системи конфігурації на основі хмарного середовища. Це допомагає заощадити час, який нам доведеться витратити через помилки, які виникають під час розгортання.

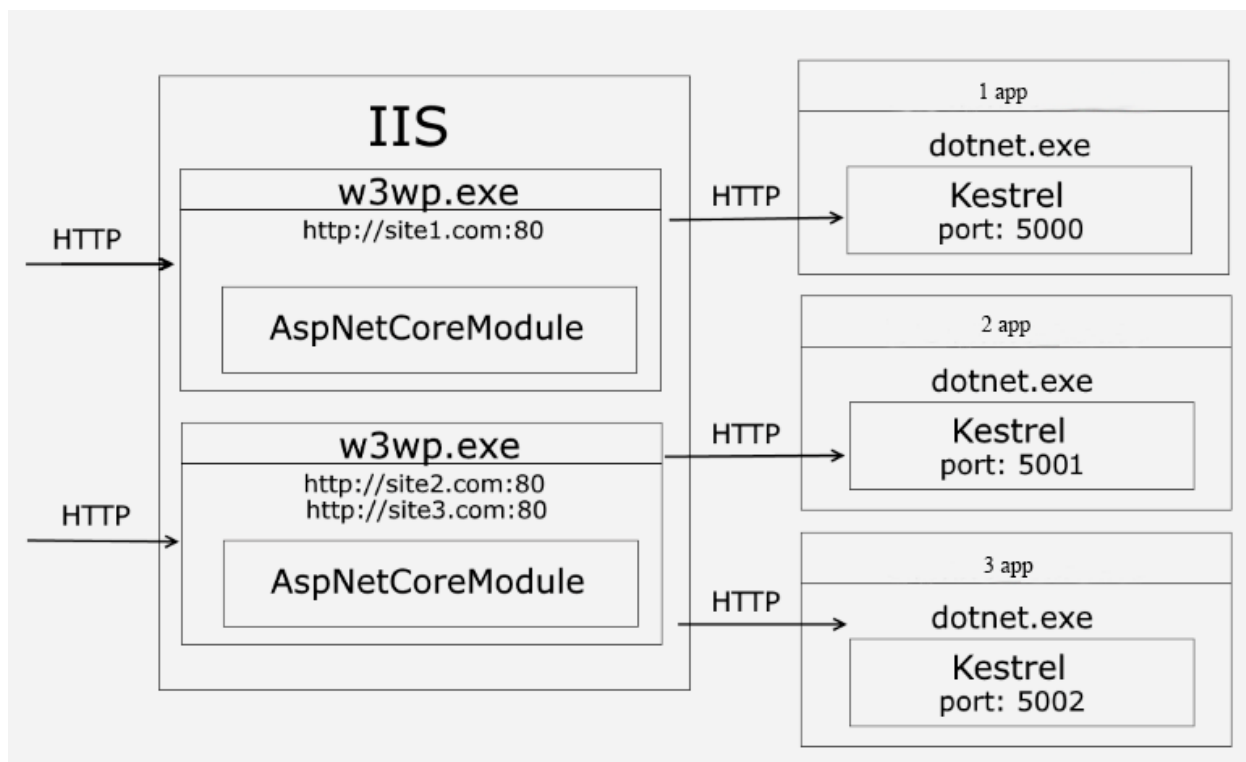


Рисунок 3.1 – Схема роботи MVC

Razor – це синтаксис розмітки, який плавно поєднує мову гіпертекстової розмітки (HTML) із синтаксисом C# і Visual Basic, причому C# є найпоширенішим. Якщо розглядати рішення ASP.NET, файли Razor матимуть розширення .cshtml, .vbhtml або .razor. Зкомпільований артефакт Razor — це виконуваний код, який записує HTML у відповідь HTTP. Компіляція Razor забезпечує покращену продуктивність у сценаріях виробництва. Microsoft розробив Razor для спрощення створення представлень у рамках ASP.NET MVC 3 і є наступником aspx синтаксису традиційних програм ASP.NET WebForms. Метою синтаксису Razor є створення серверних артефактів HTML для веб-клієнтів.

Синтаксис Razor можна впізнати за допомогою кількох зарезервованих символів і ключових слів, які дозволяють розробникам переходити в область C# і повертатися назад до HTML, найпомітнішим є @символ.

```
<span>@DateTime.Now</span>
```

Рисунок 3.2 – Фрагмент синтаксису Razor

Головна філософія Razor полягає в тому, щоб наблизити розробників ASP.NET до веб-екосистеми, використовуючи існуючий синтаксис і технології. Традиційний aspxформат поставив розробників ASP.NET у невідатне становище та ускладнив використання досягнень спільноти HTML. Чим ближче розробники ASP.NET до HTML, тим більше вони можуть покладатися на інших професіоналів, екосистеми та технології для створення веб-рішень. Замість того, щоб бути непрозорою абстракцією, Razor використовує HTML, CSS і JavaScript як необхідність під час створення веб-додатків.

Синтаксис Razor продовжує розвиватися та зазнає додаткових удосконалень. Побудований на основі синтаксису C#, .NET продовжує додавати зарезервовані ключові слова для покращення роботи розробника. Деякі з цих ключових слів є специфічними для контексту, у якому розробники використовують Razor. Передбачуваним сценарієм використання Razor був фреймворк ASP.NET MVC, в першу чергу живлення частин View, які розробники мали створювати. Хоча ASP.NET MVC може підтримувати різні механізми перегляду, за замовчуванням знаходиться RazorViewEngine в Microsoft.AspNetCore.Mvc.Razor просторі імен.

Перегляди ASP.NET Core MVC Razor є надмножиною Razor, яка надає розробникам більше функціональних можливостей під час створення представлень. Ця додаткова функціональність походить від RazorPage типу під Microsoft.AspNetCore.Mvc.Razor простором імен, який додає кілька контекстних властивостей і методів, не властивих vanilla Razor. Деякі властивості включають:

- HttpContext;
- Модель;
- ViewData;
- Допоміжні методи розділу.

RazorViewEngine безпосередньо зареєстрований у своїх програмах ASP.NET, оскільки ASP.NET реєструє компонент за допомогою викликів AddControllersWithViews, який, у свою чергу, викликає AddRazorViewEngine().

Razor Pages — це дуже впевнений підхід до створення веб-додатків, який значною мірою спирається на уроки, засвоєні за роки розробки з інфраструктурою MVC. Razor Pages відмовляється від церемонії контролерів і приймає більш зосереджене мислення на сторінці.

Razor Pages є чудовою відправною точкою для людей, які починають свою подорож до ASP.NET, оскільки вона містить необхідні частини, необхідні для створення інтерактивного Інтернету. Користувачі Razor Pages взаємодітимуть із такими концепціями, як рендеринг HTML, прив'язка моделі та обробка запитів/відповідей.

Сторінки Razor мають майже той самий синтаксис, що й представлення ASP.NET MVC Razor, за кількома винятками. Давайте розглянемо реалізацію сторінки Razor.

Суттєвою відмінністю є наявність `@page` директиви, яка вказує ASP.NET Core розглядати представлення Razor як комбінацію дії MVC і перегляду. Наявність директиви дозволяє сторінці безпосередньо обробляти вхідні запити. Директива `@model` подібна до моделі, знайденої в MVC, з однією відмінністю: визначений тип моделі повинен успадкувати від `PageModel` знайденого в `Microsoft.AspNetCore.Mvc.RazorPages` просторі імен.

Крім інфраструктурних відмінностей, синтаксис Razor, який використовується в ASP.NET Core MVC і Razor Pages, ідентичний. Багато додатків у стилі MVC можна перенести на Razor Pages з невеликими змінами у представленнях, окрім створення посилань, яке тепер використовує посилання на сторінки, а не контролери та дії.

3.2 Розробка бази даних

Визначимо основні сутності.

Сутність «Lead» містить в собі 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;
- Email – електронна пошта, text;
- Title – посада, тип text;
- Proof – посилання на ліда, тип text;
- Country – країна в якій лід проживає, тип text;
- State – регіон в країні, тип text;
- City – місто в країні де проживає лід, тип text;
- Street – вулиця та номер будинку в місті, де знаходиться відділення компанії в якій працює лід, тип text;
- ZipCode – поштовий код міста, тип text;
- CompanyId – ідентифікатор із сутності «Company» який вказує на компанію де працює лід та надає змогу отримати про неї дані, тип integer;
- Comment – поле для коментаря де можна залишити додаткову інформацію про специфіку роботи ліда (не обов'язкове поле), 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;
- EmplSize – кількість працівників компанії, тип text;
- EmplSizeProof – посилання на кількість працівників, тип text;
- RevenueSize – дохід компанії за рік, тип text;
- RevenueSizeProof – посилання на дохід компанії за рік, тип text;
- Industry – індустрія компанії, тип text;
- Phone – номер телефону компанії, тип text;

- **Comment** – поле для коментаря де можна залишити додаткову інформацію про специфіку компанії (не обов'язкове поле), тип `text`.

Варто зазначити що пропустимо опис сутності `AspNetUsers` так як це автоматично створювана сутність `Identity Framework` де додається лише два атрибута `FirstName` та `LastName` для користувача що буде зареєстрований.

Виходячи з того що для роботи з базою даних ми використовуємо `Entity Framework` то нам достатньо описати ці таблиці в моделях, встановити фреймворк через менеджер пакетів `nuGet` (див. рисунок 3.2), програмно реалізувати патерн `CodeFirst` і виконати міграцію через консоль диспетчера пакетів за допомогою наступних команд `Add-Migration` `NameMigration` і `Update-Database`.

`NuGet` — це система керування пакетами для `.NET`. Мета `NuGet` — максимально спростити процес включення сторонніх бібліотек у ваші рішення.

Керування пакетами саме по собі не є новою концепцією. Від `Apt` і «`deity`» до нього на системному рівні на `*nix`, до `Ruby Gems`, `Maven`, `Synaptic`, `portage`, `dpkg`, `rpm` та інших, це добре зрозумілий простір. Існують менеджери пакунків для операційних систем, які встановлюють бібліотеки для всієї машини, і є менеджери пакетів для розробників та їхніх проектів, які керують залежностями та інсталиують бібліотеки. `NuGet` черпає натхнення з `Ruby Gems` і додає деякі особливості `.NET`.

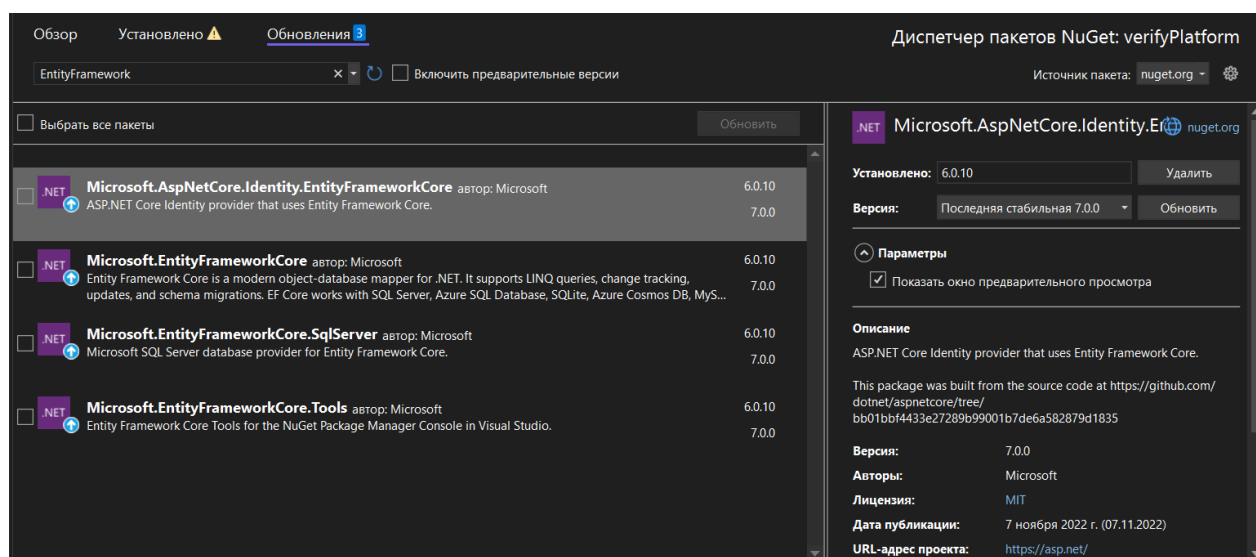


Рисунок 3.3 – Приклад встановлення фреймворку через менеджер пакетів `nuGet`

Виходячи з того що для роботи з БД використовується підхід CodeFirst то переглянемо структури бази даних через графічний інструмент Microsoft SQL Server Management Studio.

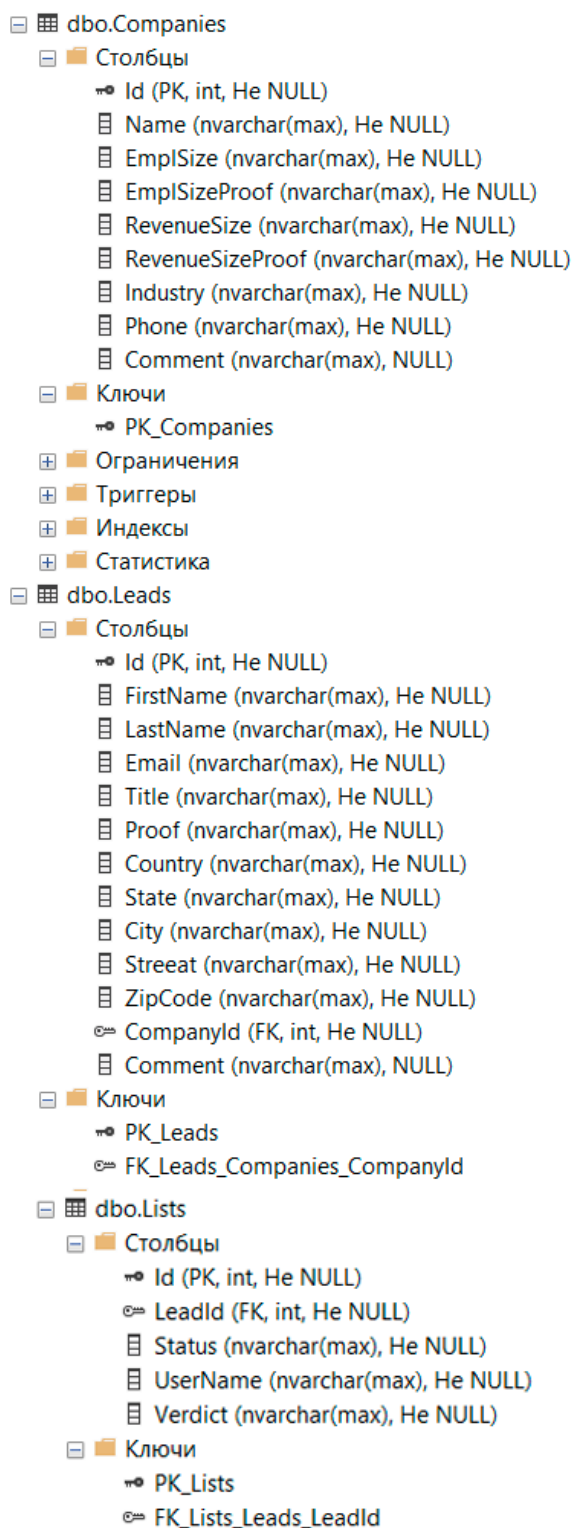


Рисунок 3.4 – Створена база даних

3.3 Розробка користувацького інтерфейсу

Головне меню являє собою представлення в зручному вигляді основних даних та функціонал для роботи з ними.

add new user

add new list



List Name	Edit List	Delete List	Last Changes
ListMicrosoft1670773273			04.12.2022

Рисунок 3.5 – Сторінка відображення всіх наявних списків

На цій сторінці ми маємо бокове меню (рис. 3.5) яке містить пункт з посилання цієї сторінки на саму себе (Dashboard) и пункт виходу з аккаунту (Log Out).

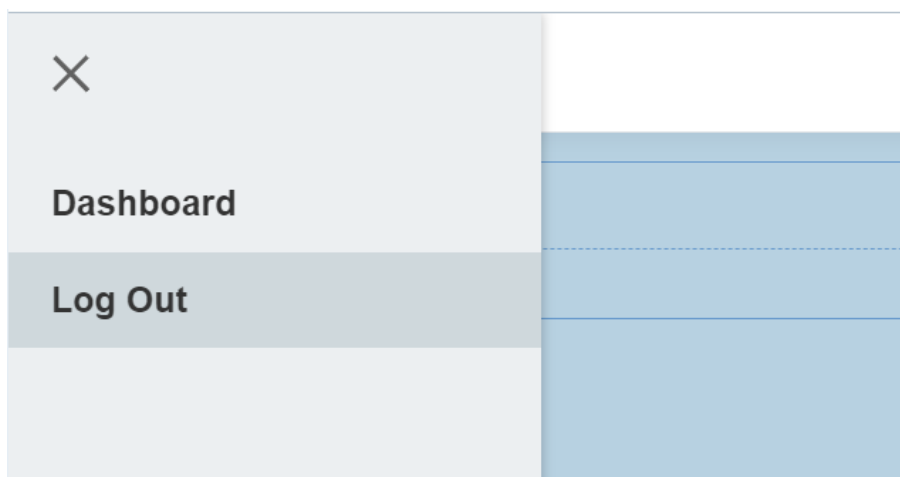


Рисунок 3.6 – Бокове меню

Біля кнопки бокового меню є ще дві додаткові кнопки add new user та add new list. Першою ми можемо додати нового користувача для редагування даних, відповідною другою кнопкою можемо додати новий список який користувачі будуть перевіряти на актуальність даних і в разі необхідності редагувати їх.

При створенні списку ми можемо задати необхідні правила для його верифікації, яких мають дотримуватися користувачі (див рисунок 3.6).

Рисунок 3.7 – Сторінка для створення нового списку

Також в таблиці для кожного списку є дві додаткові кнопки Edit List та Delete List. Edit List дозволяє відредагувати правила верифікації якщо була допущена помилка при їх написанні або замовник оновив вимоги (поки що працює лише в оновленій бета версії). Delete List видаляє список, (в майбутньому потрібно буде реалізувати механізм який спочатку створюватиме cvs копію на випадок якщо дані ще колись знадобляться).

Щоб перейти до генерації лідів користувачеві потрібно натиснути на список і буде відкрито сторінку верифікації.

Рисунок 3.8 – Сторінка для додавання нового ліда або для верифікації наявних

За своєю структурою сторінка ділиться 4 групи (блоки). Перша група – це група кнопок у верху сторінки, друга група це дані про ліда, третя – це дані про компанію і четверта – це геолокація ліда. Також під цими групами є поля для коментарів якщо якась з інформації має певну специфіку і користувачеві важко визначитися чи підходить лід або компанія під задані правила.

Стосовно кнопок, перша реалізована випадаючим списком з можливістю вибору вердикту ліду. Друга кнопка зберігає внесені дані і пропонує почати процес заново, третя не зберігає дані і повертає на початкову сторінку.

Також на другій і третій формах є стероїди, вони прискорюють процес пошуку інформації, поки що можна використовувати лише як патерни.

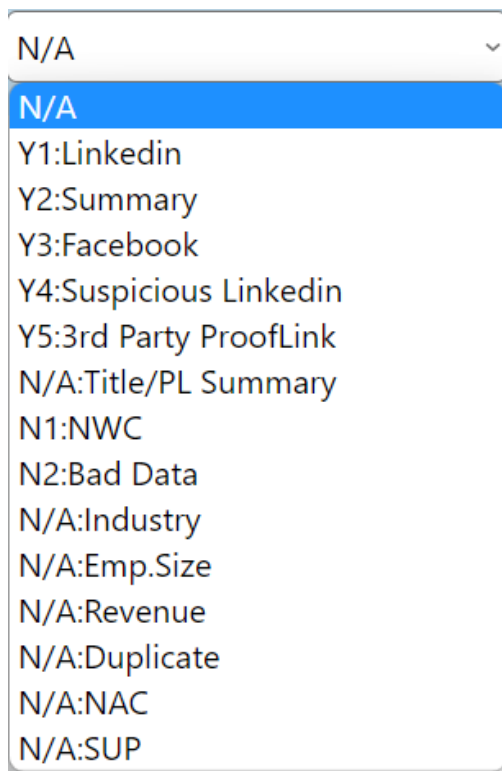


Рисунок 3.9 – Список вибору вердикту

Список для вибору вердикту має наступні значення:

- 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 компанії;
- 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.

Для стандартизації індустрії було проведено аналіз і систематизацію, тому було прийнято рішення зробити список з переліком всіх можливих індустрій (рис. 3.10).

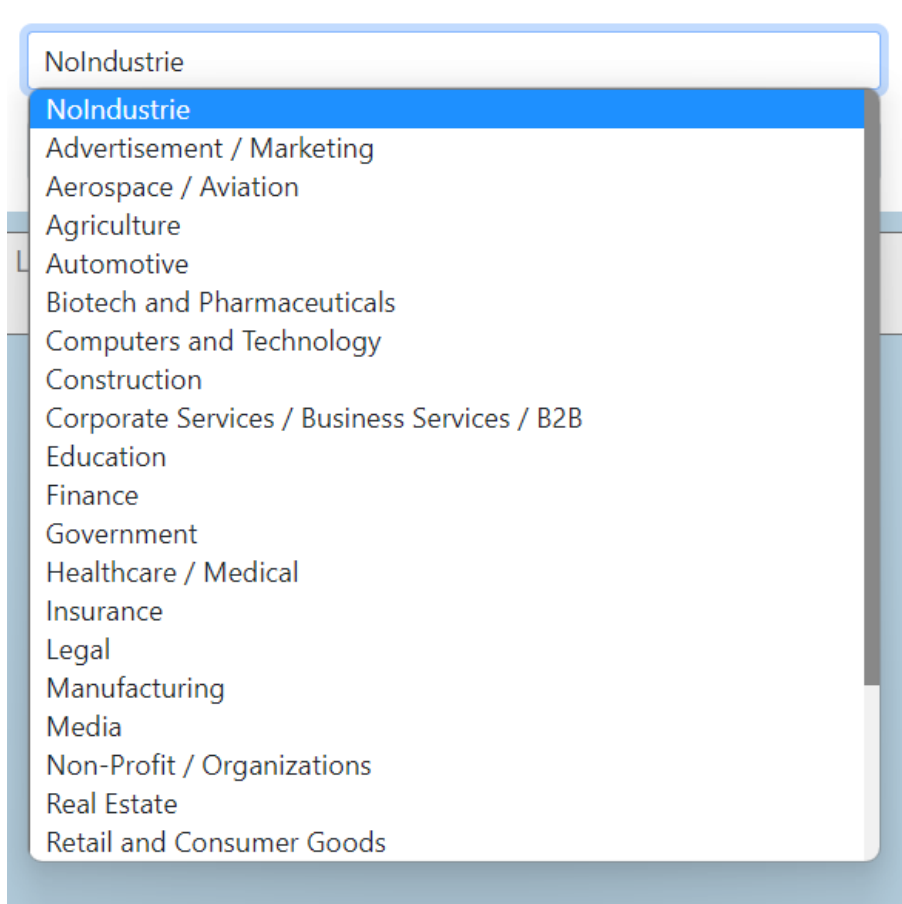
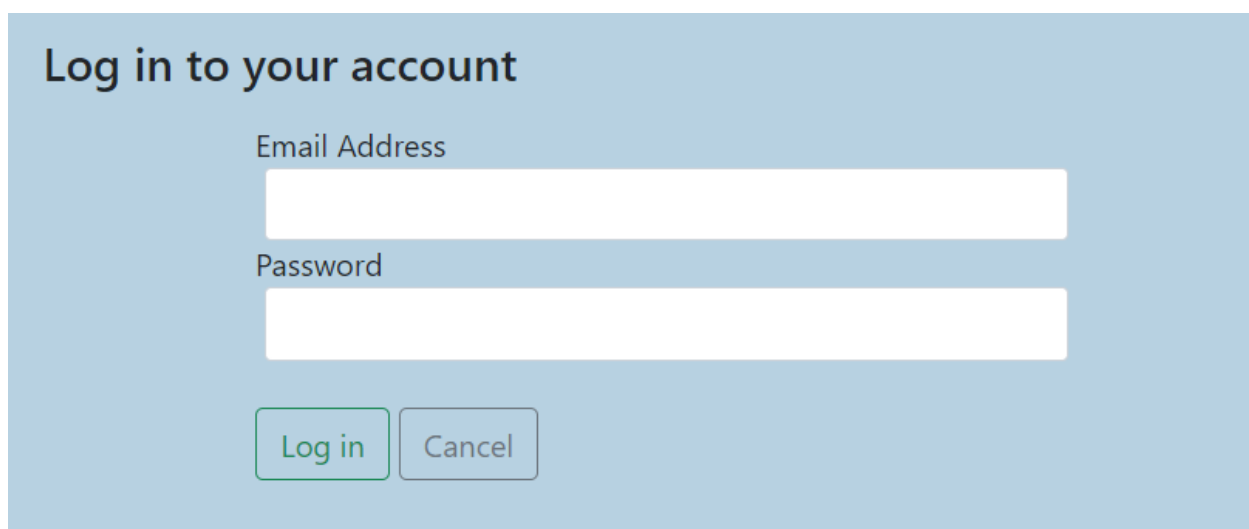


Рисунок 3.10 – Список можливих індустрій

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 – IT 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.



The image shows a login interface with a light blue background. At the top, the text 'Log in to your account' is displayed in a bold, dark font. Below this, there are two input fields: 'Email Address' and 'Password'. Each field has a white rectangular input area. At the bottom of the form, there are two buttons: 'Log in' and 'Cancel'. The 'Log in' button has a green border, while the 'Cancel' button has a grey border.

Рисунок 3.11 – Сторінка авторизації

Sign up for a new account

Email address

Password

Confirm password

FirstName

LastName

Рисунок 3.12 – Сторінка реєстрації нового користувача в системі

РОЗДІЛ 4

ТЕСТУВАННЯ

4.1 Вибір виду тестування

Існують різні типи тестування програмного забезпечення, такі як функціональне тестування, нефункціональне тестування, автоматизоване тестування, гнучке тестування та їх підтипи тощо. Кожен вид тестування має свої особливості, переваги, а також недоліки. На рисунку 4.1 наведено класифікацію типів тестування програмного забезпечення високого рівня.

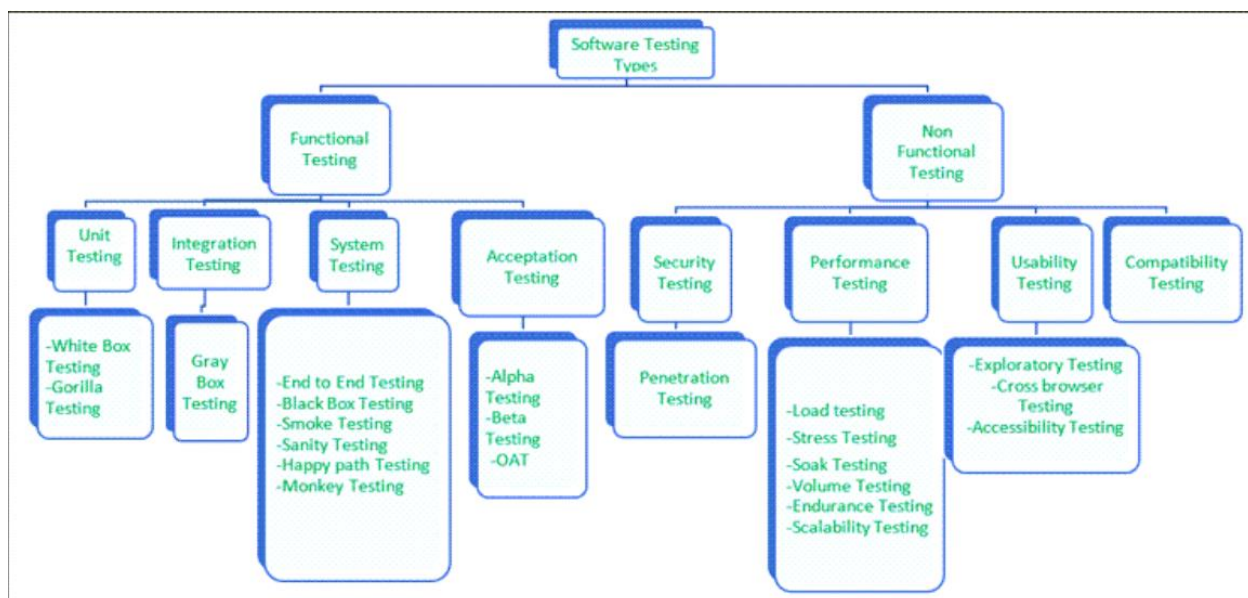


Рисунок 4.1 – Класифікація типів тестування

Для тестування розроблених модулів скористаємося ручним тестуванням, застосовуючи як негативний, так й позитивний підхід.

4.2 Тест план

Щоб провести тестування[9] необхідно скласти тест-план та визначити функції які будуть тестуватися.

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

- Кнопка «add new user»;
- Кнопка «add new list»;
- Кнопка-посилання на форму верифікації по назві листа;
- Кнопка «Logout»

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

- Редагування або внесення інформації у поля;
- Кнопка «Add New Contact»;
- Кнопка «Release»;
- Список з вердиктами;
- Список з індустріями.

Таблиця 4.1 – Тест-кейси для вибраних функцій

Опис	Перевірка функціональності головної сторінки	
Передумови	Завантажена сторінка «Dashboard»	
№	Дія	Очікуваний результат
1	2	3
1	Натиснути кнопку-посилання	Перенаправлення на сторінку з редагуванням даних ліда
2	Натиснути на кнопку «Logout»	Перенаправлення на сторінку для введення логіна і пароля
3	Натиснути кнопку «add new user»	Перенаправлення на сторінку для реєстрації нового користувача
4	Натиснути кнопку «add new list»	Перенаправлення на сторінку для створення нового списку.

Продовження таблиці 4.1

1	2	3
5	Натиснути на кнопку «Release»	Перенаправлення на сторінку «Dashboard»
6	Натиснути на кнопку «Add New Contact»	Дані зберігаються, з'являється повідомлення про успішне збереження

Після того, як веб-сервіс буде готовий до розгортання, його можна розгорнути, завантаживши файли і структуру каталогів системи на віддалений сервер.

ВИСНОВКИ

Мета дипломної роботи полягала в розробці Web-додатку, який прискорив би процес процес лідогенерації в індустрії маркетингу. Створено прототип CRM платформи, тут реалізовано базові функції які показують переваги такого підходу до лідогенерації і це все попри те, що прикладне програмне забезпечення для організацій прийнято використовувати для автоматизації стратегій.

Будь-який програмний продукт має підтримуватися і оновлюватися іншими словами допрацьовуватися і вдосконалюватися, тобто це передбачає, що всі функції які будуть додаватися в майбутньому мають бути реалізовані окремими модулями. Сама головне що такий підхід є реальним, бо ASP.NET Core використовує MVC архітектуру і це дає можливість модульного підходу при реалізації нових функцій.

Вважається що поставленої задачі було досягнуто, програмний продукт автоматизує та прискорює процес збору і обробки даних. Додаток має необхідний інструментарій для додавання, редагування і видалення інформації. Також було освоєно технологію ASP.NET Core та такі фреймворки як Entity та Identity і патерн CodeFirst який значно полегшує роботу з базою даних надаючи можливість писати мінімум коду і взагалі не взаємодіяти з графічним редактором БД.

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. Структура веб додатків [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <http://labaka.ru/likbez/struktura-veb-prilozheniy>
4. Гайдаржи В.І. Базы даних в інформаційних системах: підруч. [для студ. вищ. навч. закл.] / В.І. Гайдаржи, І.В. Ізварін. – Київ: Видавництво “Університету “Україна”, 2018. – 420 с.
5. Конспект лекцій [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://www.dstu.dp.ua/Portal/Data/3/19/3-19-kl43.pdf>
6. Г.Ю. Громов. Введення в реляційні бази даних – 2009. – 254 с. 5.
7. Hanmer R. Pattern-Oriented Software Architecture For Dummies / Robert Hanmer. – USA: For Dummies, 2013. – 384 с. – (1).
8. Архітектура ПЗ [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: <https://www.educative.io/blog/how-to-design-a-web-application-software-architecture-10>
9. Тестування ПЗ [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F
10. ASP.Net Core [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/ASP.NET_Core
11. Entity Framework [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Entity_Framework

12. Адам Фримен — ASP.NET Core MVC с примерами на C# для профессионалов [Электронный ресурс]. – 2017. – URL: <https://bit.ly/2JbSgHe>.
13. Автоматизація [Электронный ресурс]. – Точка доступа: URL: <http://uk.wikipedia.org/wiki/Автоматизація> – Автоматизація.
14. Identity [Электронный ресурс]. – Точка доступа: URL: <https://learn.microsoft.com/en-us/aspnet/identity/overview/>.

ДОДАТОК А

ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

1. Основні контролери проекту;

AccountController.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using verifyPlatform.Data;
using verifyPlatform.Models;
using verifyPlatform.ViewModels;

namespace verifyPlatform.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly ApplicationDbContext _context;

        public AccountController(UserManager<AppUser> userManager,
            SignInManager<AppUser> signInManager,
            ApplicationDbContext context)
        {
            _context = context;
            _signInManager = signInManager;
            _userManager = userManager;
        }

        [HttpGet]
        public IActionResult Login()
```

```

{
    var response = new LoginViewModel();
    return View(response);
}

[HttpPost]
public async Task<IActionResult> Login(LoginViewModel loginViewModel)
{
    if (!ModelState.IsValid) return View(loginViewModel);

    var user = await _userManager.FindByEmailAsync(loginViewModel.EmailAddress);

    if (user != null)
    {
        //User is found, check password
        var passwordCheck = await _userManager.CheckPasswordAsync(user,
loginViewModel.Password);
        if (passwordCheck)
        {
            //Password correct, sign in
            var result = await _signInManager.PasswordSignInAsync(user,
loginViewModel.Password, false, false);
            if (result.Succeeded)
            {
                //IdentityUser User = await
_userManager.FindByNameAsync(HttpContext.User.Identity.Name);

                return RedirectToAction("Dashboard", "List");
            }
        }
    }
}

```



```

    }
}
//Password is incorrect
TempData["Error"] = "Wrong credentials. Please try again";
return View(loginViewModel);
}
//User not found
TempData["Error"] = "Wrong credentials. Please try again";
return View(loginViewModel);
}

```

[HttpGet]

```

public IActionResult Register()
{
    var response = new RegisterViewModel();
    return View(response);
}

```

[HttpPost]

```

public async Task<IActionResult> Register(RegisterViewModel
registerViewModel)
{
    if (!ModelState.IsValid) return View(registerViewModel);

    var user = await
_userManager.FindByEmailAsync(registerViewModel.EmailAddress);
    if (user != null)
    {
        TempData["Error"] = "This email address is already in use";
        return View(registerViewModel);
    }
}

```

```

    }

    var newUser = new AppUser()
    {
        Email = registerViewModel.EmailAddress,
        FirstName = registerViewModel.FirstName,
        LastName = registerViewModel.LastName,
        UserName = registerViewModel.FirstName
    };

    var newUserResponse = await _userManager.CreateAsync(newUser,
registerViewModel.Password);

    if (newUserResponse.Succeeded)
        await _userManager.AddToRoleAsync(newUser, UserRoles.User);

    return RedirectToAction("Login", "Account");
}

[HttpGet]
public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    return RedirectToAction("Login", "Account");
}

}
}

```

LeadsController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;

```

```

using verifyPlatform.Data;
using verifyPlatform.Models;

namespace verifyPlatform.Controllers
{
    public class LeadsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public LeadsController(ApplicationDbContext context)
        {
            _context = context;
        }

        public IActionResult Contact()
        {
            List<string> rules = new List<string>();
            rules.Insert(0, HttpContext.Request.Cookies["RulesTitle"]);
            rules.Insert(1, HttpContext.Request.Cookies["RulesGEO"]);
            rules.Insert(2, HttpContext.Request.Cookies["RulesEmplpyeesSize"]);
            rules.Insert(3, HttpContext.Request.Cookies["RulesRevenueSize"]);
            rules.Insert(4, HttpContext.Request.Cookies["RulesIndustrie"]);
            return View(rules);
        }

        private bool key;

        public JsonResult AddNewContact(string[] choices)
        {
            if (choices == null)
                Console.WriteLine("NULL");
        }
    }
}

```

```

foreach (var company in _context.Companies.ToList())
{
    if (company.Name == choices[10])
    {
        Lead lead1 = new Lead
        {
            FirstName = choices[0],
            LastName = choices[1],
            Email = choices[2],
            Title = choices[3],
            Proof = choices[4],
            Country = choices[5],
            State = choices[6],
            City = choices[7],
            Streeat = choices[8],
            ZipCode = choices[9],
            CompanyId = company.Id
        };
        _context.Leads.AddRange(lead1);
        _context.SaveChanges();
        Console.WriteLine(company.Name);
        key = true;
        break;
    }
}
Console.WriteLine(key);

if (key != true)
{
    Company company1 = new Company

```

```

{
    Name = choices[10],
    EmplSize = choices[11],
    EmplSizeProof = choices[12],
    RevenueSize = choices[13],
    RevenueSizeProof = choices[14],
    Industry = choices[15],
    Phone = choices[16]
};
Lead lead1 = new Lead
{
    FirstName = choices[0],
    LastName = choices[1],
    Email = choices[2],
    Title = choices[3],
    Proof = choices[4],
    Country = choices[5],
    State = choices[6],
    City = choices[7],
    Streeat = choices[8],
    ZipCode = choices[9],
    Company = company1
};
_context.Companies.AddRange(company1);
_context.Leads.AddRange(lead1);
_context.SaveChanges();
}

string NameList = HttpContext.Request.Cookies["ListName"];
string StringConnection = "Data Source=(localdb)\\mssqllocaldb;Initial
Catalog=PlatformOnlineVerefy;Integrated Security=True;Connect

```

```
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False";
```

```
    string seedlist = string.Format("Insert Into " + NameList + " (LeadId, Verdict)  
Values(@LeadId, @Verdict)");
```

```
    List<Lead> lead = _context.Leads.ToList();
```

```
    try
```

```
    {
```

```
        SqlConnection conn = new SqlConnection(StringConnection);
```

```
        conn.Open();
```

```
        foreach (var item in lead)
```

```
        {
```

```
            using (SqlCommand cmdSeed = new SqlCommand(seedlist, conn))
```

```
            {
```

```
                if (item.Email == choices[2])
```

```
                {
```

```
                    //Добавить параметры
```

```
                    cmdSeed.Parameters.AddWithValue("@LeadId", item.Id);
```

```
                    cmdSeed.Parameters.AddWithValue("@Verdict", choices[17]);
```

```
                    cmdSeed.ExecuteNonQuery();
```

```
                }
```

```
            }
```

```
        }
```

```
        Console.WriteLine("Table was Created!");
```

```
        conn.Close();
```

```
    }
```

```
    catch (Exception ex)
```

```
    {
```

```
        Console.WriteLine(ex.Message);
```

```
    }
```

```

        string cook = HttpContext.Request.Cookies["RulesTitle"];
        return Json(cook);
    }
}
}

```

ListController.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using verifyPlatform.Data;
using verifyPlatform.Models;
using Microsoft.Net.Http.Headers;

namespace verifyPlatform.Controllers
{
    public class ListController : Controller
    {
        private readonly ApplicationDbContext _context;

        public ListController(ApplicationDbContext context)
        {
            _context = context;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Dashboard()
        {
            List<ListGroup> listGroups = _context.ListGroups.ToList();

```

```

        return View(listGroups);
    }

    public JsonResult CreateList(string[] choices)
    {

        double unixTime = (DateTime.Now - new System.DateTime(1970, 1, 1, 0, 0,
0, 0)).TotalSeconds;

        int unixTimeInt = Convert.ToInt32(unixTime);
        string utunixTimeString = Convert.ToString(unixTimeInt);
        string NameList = choices[0] + utunixTimeString;

        string StringConnection = "Data Source=(localdb)\\mssqllocaldb;Initial
Catalog=PlatformOnlineVerefy;Integrated                        Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWr
ite;MultiSubnetFailover=False";

        string Command = @"DECLARE @TableName nvarchar(max) = '' +
NameList + ';' +
        "DECLARE @Sql NVARCHAR(max) = 'create table '" +
        " + @TableName + '([Id][int] PRIMARY KEY IDENTITY(1, 1) NOT
NULL," +
        "[LeadId] [int] NOT NULL," +
        "[Status] [nvarchar] (max) NULL," +
        "[UserName][nvarchar] (max) NULL," +
        "[Verdict][nvarchar] (max) NULL)';" +
        "EXECUTE sp_executesql @Sql";
        string sql = string.Format("Insert Into ListGroups" +
        "(ListName, Status, RulesTitle, RulesGEO, RulesEmplpyeesSize,
RulesRevenueSize, RulesIndustrie) Values(@ListName, @Status, @RulesTitle,
@RulesGEO, @RulesEmplpyeesSize, @RulesRevenueSize, @RulesIndustrie)");
    }

```



```

try
{
    SqlConnection conn = new SqlConnection(StringConnection);
    SqlCommand cmd = new SqlCommand(Command, conn);
    conn.Open();
    cmd.ExecuteNonQuery();
    using (SqlCommand cmdInsert = new SqlCommand(sql, conn))
    {
        // add parameters
        cmdInsert.Parameters.AddWithValue("@ListName", NameList);
        cmdInsert.Parameters.AddWithValue("@Status", "No");
        cmdInsert.Parameters.AddWithValue("@RulesTitle", choices[1]);
        cmdInsert.Parameters.AddWithValue("@RulesGEO", choices[2]);
        cmdInsert.Parameters.AddWithValue("@RulesEmplpyeesSize",
choices[3]);
        cmdInsert.Parameters.AddWithValue("@RulesRevenueSize",
choices[4]);
        cmdInsert.Parameters.AddWithValue("@RulesIndustrie", choices[5]);
        cmdInsert.ExecuteNonQuery();
    }
    Console.WriteLine("Rules Insert");

    conn.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

```

        string CommandKey = @"DECLARE @TableName nvarchar(max) = " +
NameList + ";" +
        "DECLARE @Sql1 NVARCHAR(max) = 'ALTER TABLE
' + @TableName + ' WITH CHECK ADD CONSTRAINT
[FK_' + @TableName + '_Leads_Id] FOREIGN KEY([LeadId]);' +
        "EXECUTE sp_executesql @Sql1";
    try
    {
        SqlConnection conn = new SqlConnection(StringConnection);
        SqlCommand cmdkey = new SqlCommand(CommandKey, conn);
        cmdkey.ExecuteNonQuery();
        conn.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return Json(choices);
}

public JsonResult DeleteList(string[] choices)
{
    string NameList = choices[0];
    string StringConnection = "Data Source=(localdb)\\mssqllocaldb;Initial
Catalog=PlatformOnlineVerefy;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWr
ite;MultiSubnetFailover=False";
    string dellist = string.Format("IF EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[ ' + NameList + '']) AND type in (N'U'))
" +

```

```

        "DROP TABLE " + NameList + "");

        string delstr = string.Format("DELETE FROM ListGroups WHERE ListName
= " + NameList + "");

        try
        {
            SqlConnection conn = new SqlConnection(StringConnection);
            conn.Open();
            using (SqlCommand cmdDel = new SqlCommand(dellist, conn))
            {
                //add
                cmdDel.ExecuteNonQuery();
            }
            using (SqlCommand cmdDel = new SqlCommand(delstr, conn))
            {
                //add parameters
                cmdDel.ExecuteNonQuery();
            }
            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        return Json(choices);
    }

    public JsonResult ViewList(string[] choices)
    {
        string NameList = choices[0];

```

```

string StringConnection = "Data Source=(localdb)\\mssqllocaldb;Initial
Catalog=PlatformOnlineVerefy;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWr
ite;MultiSubnetFailover=False";

string viewrules = string.Format("SELECT [RulesTitle], " +
    "[RulesGEO], " +
    "[RulesEmplpyeesSize]," +
    "[RulesRevenueSize]," +
    "[RulesIndustrie] FROM ListGroups WHERE ListName = '" + NameList +
    "'");

try
{
    using (SqlConnection conn = new SqlConnection(StringConnection))
    {
        conn.Open();
        // add parameters
        SqlCommand cmdViewRules = new SqlCommand(viewrules, conn);
        SqlDataReader reader = cmdViewRules.ExecuteReader();
        if (reader.HasRows) // is isset data
        {
            while (reader.Read()) // read in line
            {
                HttpContext.Response.Cookies.Append("RulesTitle",
reader.GetValue(0).ToString());
                HttpContext.Response.Cookies.Append("RulesGEO",
reader.GetValue(1).ToString());
                HttpContext.Response.Cookies.Append("RulesEmplpyeesSize",
reader.GetValue(2).ToString());
                HttpContext.Response.Cookies.Append("RulesRevenueSize",
reader.GetValue(3).ToString());
            }
        }
    }
}

```

```

        HttpContext.Response.Cookies.Append("RulesIndustrie",
reader.GetValue(4).ToString());
        HttpContext.Response.Cookies.Append("ListName", choices[0]);
        Console.WriteLine();
    }
}
reader.Close();
conn.Close();
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
var cookie = new CookieHeaderValue("id", "12345");
Console.WriteLine(HttpContext.Request.Cookies["RulesTitle"]);
return Json("text");
}
}
}

```

2. Основні сторінки проекту.

Login.cshtml

```
@using verifyPlatform.ViewModels
```

```
@model LoginViewModel
```

```
@{
```

```
    ViewData["Title"] = "Login";
```

```
}
```

```
<div class="row">
```

```
    <div class="col-md-6 offset-3">
```

```

<p>
    <h4>Log in to your account</h4>
</p>
@if(TempData["Error"] != null)
{
    <div class="col-md-12 alert alert-danger">
        <span><b>Sorry!</b> - @TempData["Error"]</span>
    </div>
}
<div class="row">
    <div class="col-md-8 offset-2">
        <form asp-action="Login">
            <div      asp-validation-summary="ModelOnly"      class="text-
danger"></div>
            <div class="form-group">
                <label asp-for="EmailAddress" class="control-label"></label>
                <input asp-for="EmailAddress" class="form-control">
                <span      asp-validation-for="EmailAddress"      class="text-
danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                <br/>
                <input class="btn btn-outline-success float right" type="submit"
value="Log in" />

```

```

        <a class="btn btn-outline-secondary" asp-controller="Home" asp-
action="Index">Cancel</a>

```

```

    </div>

```

```

    </form>

```

```

</div>

```

```

</div>

```

```

</div>

```

```

</div>

```

```

    Register.cshtml

```

```

@using verifyPlatform.ViewModels

```

```

@model RegisterViewModel

```

```

@{

```

```

    ViewData["Title"] = "Sign up";

```

```

}

```

```

<div class="row">

```

```

    <div class="col-md-6 offset-3">

```

```

        <p>

```

```

        <h4>Sign up for a new account</h4>

```

```

        </p>

```

```

        @if (TempData["Error"] != null)

```

```

        {

```

```

            <div class="col-md-12 alert alert-danger">

```

```

                <span><b>Sorry!</b> - @TempData["Error"] </span>

```

```

            </div>

```

```

        }

```

```

    <div class="row">

```

```

<div class="col-md-8 offset-2">
  <form asp-action="Register">
    <div      asp-validation-summary="ModelOnly"      class="text-
danger"></div>

    <div class="form-group">
      <label asp-for="EmailAddress" class="control-label"></label>
      <input asp-for="EmailAddress" class="form-control" />
      <span      asp-validation-for="EmailAddress"      class="text-
danger"></span>
    </div>

    <div class="form-group">
      <label asp-for="Password" class="control-label"></label>
      <input asp-for="Password" class="form-control" />
      <span asp-validation-for="Password" class="text-danger"></span>
    </div>

    <div class="form-group">
      <label asp-for="ConfirmPassword" class="control-label"></label>
      <input asp-for="ConfirmPassword" class="form-control" />
      <span      asp-validation-for="ConfirmPassword"      class="text-
danger"></span>
    </div>

    <div class="form-group">
      <label asp-for="FirstName" class="control-label"></label>
      <input asp-for="FirstName" class="form-control" />
      <span asp-validation-for="FirstName" class="text-danger"></span>
    </div>

    <div class="form-group">
      <label asp-for="LastName" class="control-label"></label>
      <input asp-for="LastName" class="form-control" />

```



```

<link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
<link rel="stylesheet" href="~/verifyPlatform.styles.css" asp-append-
version="true" />
</head>
<body>
<div class="background-contact-page">
  <div class="container-fluid containerfluid">
    <div class="input-group input-group-sm mb-3">
      <select name="user_profile_color_2" required="required" class="col-2
verdict" id="verdict">
        <option value="">N/A</option>
        <option value="Y1:Linkedin">Y1:Linkedin</option>
        <option value="Y2:Summary">Y2:Summary</option>
        <option value="Y3:Facebook">Y3:Facebook</option>
        <option value="Y4:Suspicious Lin">Y4:Suspicious
Linkedin</option>
        <option value="Y5:3rd Party ProofLink">Y5:3rd Party
ProofLink</option>
        <option value="N/A:Title/PL Summary">N/A:Title/PL
Summary</option>
        <option value="N1:NWC">N1:NWC</option>
        <option value="N2:Bad Data">N2:Bad Data</option>
        <option value="N/A:Industry">N/A:Industry</option>
        <option value="N/A:Emp.Size">N/A:Emp.Size</option>
        <option value="N/A:Revenue">N/A:Revenue</option>
        <option value="N/A:Duplicate">N/A:Duplicate</option>
        <option value="N/A:NAC">N/A:NAC</option>
        <option value="N/A:SUP">N/A:SUP</option>
      </select>
      @if (User.Identity.IsAuthenticated && User.IsInRole("admin"))

```

```

    {
        <button type="button" id="Add" class="btn btn-success
process">Add New Contact</button>
    }
    @if (User.Identity.IsAuthenticated && User.IsInRole("user"))
    {
        <button type="button" class="btn btn-success
process">Process</button>
    }
    <button type="button" class="btn btn btn-primary process"
id="Release">Release</button>
</div>
<div class="row rowform">
    <div class="col colleft">
        <div class="row rowmarning">
            <div class="col-4 rules">
                <b>JOB LEVEL</b><br />
                <br />
                <b>JOB TITLE</b><br />
                @Html.Raw(Model[0])
            </div>
            <div class="col-8 datacheck">
                <p class="pborder"><b>Name / Email / Company</b></p>
                <div class="input-group input-group-sm mb-3">
                    <div><button type="button" class="btn btn-outline-info btn-sm
steroid">FL+CN</button></div>
                    <div><button type="button" class="btn btn-outline-info btn-sm
steroid">FL+Domain</button></div>
                    <div><button type="button" class="btn btn-outline-info btn-sm
steroid">Email</button></div>

```

```

<div><button type="button" class="btn btn-outline-info btn-sm
steroid">Domain</button></div>

```

```

</div>

```

```

<div class="input-group input-group-sm mb-3">

```

```

    <input type="text" class="form-control" placeholder="First Name"
id="FirstName">

```

```

    <input type="text" class="form-control" placeholder="Last Name"
aria-label="Last Name" aria-describedby="inputGroup-sizing-sm" id="LastName">

```

```

</div>

```

```

<div class="input-group input-group-sm mb-3">

```

```

    <input type="text" class="form-control" placeholder="Email"
id="Email" aria-label="Email" aria-describedby="inputGroup-sizing-sm">

```

```

    <span id="emailGen" class="input-group-text" style="cursor:
pointer">

```

```

        &#9654;

```

```

    </span>

```

```

</div>

```

```

<div class="input-group input-group-sm mb-3">

```

```

    <input type="text" class="form-control" style="width: 130px;"
placeholder="Company Name" id="CompanyName" aria-label="Company Name"
aria-describedby="inputGroup-sizing-sm">

```

```

    <input type="text" class="form-control" placeholder="WebSite"
id="WebSite" aria-label="WebSite" aria-describedby="inputGroup-sizing-sm">

```

```

    <span class="input-group-text" >

```

```

        &#128279;

```

```

    </span>

```

```

</div>

```

```

<p class="pborder"><b>Title / ProofLink</b></p>

```

```

<div class="input-group input-group-sm mb-3">

```

```

        <input type="text" class="form-control" placeholder="Title"
id="Title" aria-label="Title" aria-describedby="Title">
    </div>
    <div class="input-group input-group-sm mb-3">
        <input type="text" class="form-control" placeholder="ProofLink"
id="ProofLink" aria-label="ProofLink" aria-describedby="inputGroup-sizing-sm">
        <span class="input-group-text" >
            &#128279;
        </span>
    </div>
</div>
</div>
</div>

<div class="row rowmarning-2">
    <div class="col-4 rules">
        <b>GEO AND DESCRIPTION </b><br />
        @Html.Raw(Model[1])
    </div>
    <div class="col-8 datacheck">
        <p class="pborder"><b>Address</b></p>
        <div class="input-group input-group-sm mb-3">
            <input type="text" class="form-control" placeholder="Country"
id="Country" aria-label="Country" aria-describedby="inputGroup-sizing-sm">
            <span class="input-group-text" >
                <div class='switcher'>
                    <label
                        class='switcher-label
                        switcher-off
for='offGEO'>off</label>
                    <input id='offGEO' class='switcher-radio-off' type='radio'
name='value1' value='off'>

```

```

        <label                class='switcher-label                switcher-neutral'
for='neutralGEO'>neutral</label>

        <input                id='neutralGEO'                class='switcher-radio-neutral'
type='radio' name='value1' value='neutral' checked>

        <label                class='switcher-label                switcher-on'
for='onGEO'>on</label>

        <input                id='onGEO'                class='switcher-radio-on'                type='radio'
name='value1' value='on'>

        <div class='switcher-slider'></div>

        </div>

    </span>

    <input                type="text"                class="form-control"                placeholder="State"
id="State" aria-label="State" aria-describedby="inputGroup-sizing-sm">

    </div>

    <div class="input-group input-group-sm mb-3">

        <input                type="text"                class="form-control"                placeholder="City"
id="City" aria-label="City" aria-describedby="inputGroup-sizing-sm">

        <input type="text" class="form-control" placeholder="Zip Code"
id="ZipCode" aria-label="Zip Code" aria-describedby="inputGroup-sizing-sm">

    </div>

    <div class="input-group input-group-sm mb-3">

        <input type="text" class="form-control" placeholder="Address"
id="Address" aria-label="Address" aria-describedby="inputGroup-sizing-sm">

    </div>

</div>

</div>

<div class="row">

    <textarea                class="comment"                placeholder="Leave a comment
here"></textarea>

```

```

</div>
</div>
<div class="col colright">
  <div class="row">
    <div class="col-8 datacheck">
      <p class="pborder"><b>Employees / Revenue</b></p>
      <div class="input-group input-group-sm mb-3">
        <div><button type="button" class="btn btn-outline-info btn-sm
steroid">D+YF</button></div>
        <div><button type="button" class="btn btn-outline-info btn-sm
steroid">D+Linkedin</button></div>
        <div><button type="button" class="btn btn-outline-info btn-sm
steroid">D+Zoomonfo</button></div>
        <div><button type="button" class="btn btn-outline-info btn-sm
steroid">D+Wiki</button></div>
      </div>
      <div class="input-group input-group-sm mb-3">
        <input type="text" class="form-control" placeholder="Employees
Size" id="ES" aria-label="Employees Size" aria-describedby="inputGroup-sizing-
sm">
        <input type="text" class="form-control" placeholder="Employees
Size PL" id="ESPL" aria-label="Employees Size PL" aria-describedby="inputGroup-
sizing-sm">
        <span class="input-group-text" >
          &#128279;
        </span>
      </div>
      <div class="input-group input-group-sm mb-3">
        <input type="text" class="form-control" placeholder="Revenue
Size" id="RS" aria-label="Revenue Size" aria-describedby="inputGroup-sizing-sm">

```

```

    <input type="text" class="form-control" placeholder="Revenue
Size PL" id="RSPL" aria-label="Revenue Size PL" aria-describedby="inputGroup-
sizing-sm">

```

```

    <span class="input-group-text">

```

```

        &#128279;

```

```

    </span>

```

```

</div>

```

```

<div class="input-group input-group-sm mb-3">

```

```

    <select      name="user_profile_color_2"      required="required"
id="ind" class="form-control">

```

```

        <option value="NoIndustrie">NoIndustrie</option>

```

```

        <option value="Advertisement / Marketing">Advertisement /
Marketing</option>

```

```

        <option value="Aerospace / Aviation">Aerospace /
Aviation</option>

```

```

        <option value="Agriculture">Agriculture</option>

```

```

        <option value="Automotive">Automotive</option>

```

```

        <option value="Biotech and Pharmaceuticals">Biotech and
Pharmaceuticals</option>

```

```

        <option value="Computers and Technology">Computers and
Technology</option>

```

```

        <option value="Construction">Construction</option>

```

```

        <option value="Corporate Services / Business Services /
B2B">Corporate Services / Business Services / B2B</option>

```

```

        <option value="Education">Education</option>

```

```

        <option value="Finance">Finance</option>

```

```

        <option value="Government">Government</option>

```

```

        <option value="Healthcare / Medical">Healthcare /
Medical</option>

```

```

        <option value="Insurance">Insurance</option>

```



```

        <option value="Legal">Legal</option>
        <option value="Manufacturing">Manufacturing</option>
        <option value="Media">Media</option>
        <option value="Non-Profit / Organizations">Non-Profit /
Organizations</option>
        <option value="Real Estate">Real Estate</option>
        <option value="Retail and Consumer Goods">Retail and
Consumer Goods</option>
        <option value="Service Industry / B2C">Service Industry /
B2C</option>
        <option
value="Telecommunication">Telecommunication</option>
        <option value="Transportation and Logistics">Transportation
and Logistics</option>
        <option value="Travel / Hospitality / Entertainment">Travel /
Hospitality / Entertainment</option>
        <option value="Utility / Energy">Utility / Energy</option>
        <option value="Research">Research</option>
    </select>
</div>
<div class="input-group input-group-sm mb-3">
    <input type="text" class="form-control" placeholder="Phone
number" id="Phone" aria-label="Phone number" aria-describedby="inputGroup-
sizing-sm">
</div>
</div>
<div class="col-4 rules">
    <b>EMPLOYEES</b><br />
    @Html.Raw(Model[2])
    <br />

```

```

        <b>REVENUE</b><br />
        @Html.Raw(Model[3])
        <br />
        <b>INDUSTRIES</b><br />
        @Html.Raw(Model[4])
        <br/>
    </div>
</div>
<div class="row">
    <textarea class="comment" placeholder="Leave a comment
here"></textarea>
    </div>
    <div id="res" style="display: none;">

    </div>
</div>
</div>
</div>
</div>
</div>
</body>

<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script>
    $(document).ready(function(){
        $('#emailGen').click(function(e){
            var FirstName = $('#FirstName').val();
            var LastName = $('#LastName').val();
            var WebSite = $('#WebSite').val();

```

```

        e.preventDefault();

        var email = FirstName + LastName + "@pes" + WebSite;
        document.getElementById('Email').value = email;
    });
});
</script>

<script>
$(document).ready(function(){
    $('#Release').click(function(e){
        window.location.href = 'https://localhost:7081/List/Dashboard';
    });
});
</script>

<script type="text/javascript">
$(document).ready(function(){
    $('#Add').click(function(e){
        var FirstName = $('#FirstName').val();
        var LastName = $('#LastName').val();
        var Email = $('#Email').val();
        var CompanyName = $('#CompanyName').val();
        var Title = $('#Title').val();
        var ProofLink = $('#ProofLink').val();
        var Country = $('#Country').val();
        var State = $('#State').val();
        var City = $('#City').val();
        var ZipCode = $('#ZipCode').val();
        var Address = $('#Address').val();
        var RSPL = $('#RSPL').val();
    });
});

```

```

var RS = $('#RS').val();
var ESPL = $('#ESPL').val();
var ES = $('#ES').val();
var ind = $('#ind').val();
var Phone = $('#Phone').val();
var Verdict = $('#verdict').val();

e.preventDefault();
name = encodeURIComponent(name);
$('#res').load('@Url.Action("AddNewContact")', { 'choices[]':
[FirstName, LastName, Email, Title, ProofLink,
Country, State, City, Address, ZipCode,
CompanyName, ES, ESPL,
RS, RSPL,
ind, Phone, Verdict] });
alert('Contact Was Created');
var FirstName = $('#FirstName').val("");
var LastName = $('#LastName').val("");
var Email = $('#Email').val("");
var Title = $('#Title').val("");
var ProofLink = $('#ProofLink').val("");
var Verdict = $('#verdict').val("");
});
});
</script>

Dashboard.cshtml

@model IEnumerable<ListGroup>

<!-- Modal window -->
<div id="myModal" class="modal">

```

```

<!-- Modal context -->
<div class="modal-content">
  <span class="close">&times;</span>
  <p>By continuing the operation, you agree to comply with the rules of the
company's policy and assume responsibility in case of non-compliance.<br/>
    All the information you will be able to get is absolutely confidential.
  </p>
  <a asp-area="" asp-controller="Leads" asp-action="Contact">
    <button type="button" class="btn btn-outline-primary btn-sm"
id="continue">Continue</button>
  </a>
</div>

</div>

<table>
<tr><th>List Name</th><th style="text-align: center;">Edit List</th><th style="text-
align: center;">Delete List</th><th style="text-align: center;">Last
Changes</th></tr>
  @foreach (var item in Model)
  {
    <tr
      id="Dashboard"><td
      class="view"
id="@item.ListName">@item.ListName</td><td class="td" style="text-align:
center;" id="@item.Id">&#9998;</td><td class="del" id="@item.ListName"
style="text-align: center;">&#10008;</td><td
      style="text-align: center;">04.12.2022</td></tr>
  }
</table>
<div id="result" style="display: none;"></div>

```

```

@section scripts{
    <script type="text/javascript">
        $(document).ready(function(){
            $('.del').click(function() {
                var ListName = $(this).attr('id');
                $('#result').load('@Url.Action("DeleteList")', { 'choices[]': [ListName] });
                alert("List " + ListName + " Was Deleted");
                window.location.href = 'https://localhost:7081/List/Dashboard';
            });
        });
    </script>

    <script>
        $(document).ready(function () {
            $('.view').click(function () {
                var ListName = $(this).attr('id');
                $('#result').load('@Url.Action("ViewList")', { 'choices[]': [ListName] });
                var modal = document.getElementById("myModal");
                // get <span>, because close modal window
                var span = document.getElementsByClassName("close")[0];
                modal.style.display = "block";
                // click <span> (x), user to close modal window
                span.onclick = function () {
                    modal.style.display = "none";
                }
            });
        });
    </script>
}

Index.cshtml(ListController)

```

@using verifyPlatform.ViewModels

```

<div class="background-contact-page">
  <div class="container-fluid containerfluid">
    <div class="row rowform-create-list">
      <div class="col">
        <div class="form-floating">
          <input type="text" class="listname" id="ListName" placeholder="ListName">
        </div>
      </div>
    </div>
    <div class="row rowform-create-list">
      <div class="col">
        <fieldset disabled class="dset">
          <div class="mb-3">
            <input type="text" id="LeadID" class="form-control"
placeholder="LeadID">
          </div>
          <div class="mb-3">
            <input type="text" id="Status" class="form-control" placeholder="Status">
          </div>
        </fieldset>
      </div>
      <div class="col">
        <div class="form-floating">
          <textarea class="comment-create" placeholder="Leave a comment here for
rulesTitle" id="rulesTitle"></textarea>
        </div>
      </div>
      <div class="col">

```

```

    <div class="form-floating">
        <textarea class="comment-create" placeholder="Leave a comment here for
rulesGEO" id="rulesGEO"></textarea>
    </div>
</div>
</div>

<div class="row rowform-create-list">
<div class="col">
    <fieldset disabled class="dset">
        <div class="mb-3">
            <input type="text" id="Verdict" class="form-control"
placeholder="Verdict">
        </div>
        <div class="mb-3">
            <input type="text" id="UserName" class="form-control"
placeholder="UserName">
        </div>
    </fieldset>
</div>
<div class="col">
    <div class="form-floating">
        <textarea class="comment-create" placeholder="Leave a comment here for
rulesES" id="rulesES"></textarea>
    </div>
</div>
<div class="col">
    <div class="form-floating">
        <textarea class="comment-create" placeholder="Leave a comment here for
rulesRS" id="rulesRS"></textarea>

```



```

    </div>
</div>
</div>
<div class="row rowform-create-list">
    <div class="form-floating">
        <textarea class="comment-create" placeholder="Leave a comment here for
rulesIndustrie" id="rulesIndustrie"></textarea>
    </div>
</div>
<div class="row rowform-create-list">
    <a class="nav-link text-dark" asp-area="">
        <button type="button" class="btn btn-outline-success btn-sm"
id="create">Create</button>
    </a>
    <div id="result" style="display: none;"></div>
</div>
</div>
</div>

```

```
@section scripts{
```

```

<script type="text/javascript">
    $(document).ready(function(){
        $('#create').click(function(e){
            var ListName = $('#ListName').val();
            var rulesTitle = $('#rulesTitle').val();
            var rulesGEO = $('#rulesGEO').val();
            var rulesES = $('#rulesES').val();
            var rulesRS = $('#rulesRS').val();
            var rulesIndustrie = $('#rulesIndustrie').val();
            e.preventDefault();

```

```

        name = encodeURIComponent(name);

        $('#result').load('@Url.Action("CreateList")', { 'choices[]': [ListName,
rulesTitle, rulesGEO, rulesES, rulesRS, rulesIndustrie] });

        alert("List Was Created");

        window.location.href = 'https://localhost:7081/List/Dashboard';

    });

});

```

```

</script>

```

```

}

```

```

    _Layout.cshtml

```

```

@using Microsoft.AspNetCore.Identity

```

```

@inject SignInManager<AppUser> SignInManager

```

```

@inject UserManager<AppUser> UserManager

```

```

<!DOCTYPE html>

```

```

<html lang="en">

```

```

<head>

```

```

    <meta charset="utf-8" />

```

```

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

```

```

    <title>@ViewData["Title"] - verifyPlatform</title>

```

```

    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

```

```

    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />

```

```

    <link rel="stylesheet" href="~/verifyPlatform.styles.css" asp-append-
version="true" />

```

```

</head>

```

```

<body>

```

```

<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-
white border-bottom box-shadow mb-3">
        <div class="container-fluid">
            <div class="burger-menu">
                <input id="menu-toggle" type="checkbox" />
                <label class="menu-btn" for="menu-toggle">
                    <span></span>
                </label>
                <ul class="menubox">
                    @if (User.Identity.IsAuthenticated)
                    {
                        <li><a class="menu-item" asp-area="" asp-controller="List"
asp-action="Dashboard">Dashboard</a></li>
                        <li><a class="menu-item" asp-area="" asp-
controller="Account" asp-action="Logout">Log Out</a></li>
                    }
                </ul>
            </div>
            @*<p class="navbar-brand">@User.Identity.Name</p>*@
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
                <ul class="navbar-nav flex-grow-1">
                    @if (!SignInManager.IsSignedIn(User))
                    {

```

```

        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-
controller="Account" asp-action="login">Login</a>
        </li>
    }
    @if (User.Identity.IsAuthenticated && User.IsInRole("admin"))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-
controller="Account" asp-action="Register">
                <button type="button" class="btn btn-outline-primary btn-
sm">add new user</button>
            </a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-
controller="List" asp-action="Index">
                <button type="button" class="btn btn-outline-primary btn-
sm">add new list</button>
            </a>
        </li>
    }
</ul>
</div>
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    
```

```
</main>
</div>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```