РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
Факультет математики та інформатики

Кафедра інформаційно-комунікаційних технологій та
методики викладання інформатики

«До захисту допущено»
Завідувач кафедри
_____ Войтович І. С.
«__»_____20 ____ р
протокол №_____

КВАЛІФІКАЦІЙНА РОБОТА

ІМПЛЕМЕНТАЦІЯ ПОШУКОВОЇ СИСТЕМИ ДИСТАНЦІЙНИХ КУРСІВ

здобувача першого (бакалаврського) рівня вищої освіти спеціальності 015 Професійна освіта (комп'ютерні технології)

Остапчука Віталія Олеговича _____

Керівник:_____Петренко С. В., доцент кафедри Інформаційних технологій та моделювання, канд. пед. наук

Рецензент:_____Шроль Т. С., доцент кафедри Інформаційно-комунікаційних технологій та методики викладання інформатики, канд. пед. наук

Рецензент:_____Крайчук С. О., доцент кафедри Економіки та управління бізнесом, канд. техн. наук

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Рівне – 2022року

ЗМІСТ

ВСТУП

У сучасних умовах життя та розвитку суспільства, освітня діяльність масово здійснюється із використанням інтенет-ресурсів. Освітні платформи пропонують величезну кількість навчальних курсів за різноманітними тематиками. Використання масових відкритих дистанційних курсів ускладнюється тим, що серед величезної кількості представлених курсів важко знайти саме той, який здатний максимально задовольнити потреби студента. **Актуальність** проблеми пошуку потрібного навчального курсу серед безлічі проєктів представлених на освітніх платформах і зумовила вибір теми нашого дослідження.

Мета дослідження: розробка веб-додатку для оптимізації пошуку потрібних користувачу навчальних курсів із представлених на існуючих освітніх платформах.

Завдання дослідження:

- Проаналізувати можливості та переваги використання дистанційних курсів;
- Розглянути існуючі дистанційні курси та платформи для пошуку навчальних курсів
- Обрати оптимальні технології розробки веб-додатку
- Спланувати архітектуру і дизайн веб-додатку
- Розробити прикладний програмний інтерфейс для збору даних
- Розробити інтерфейс веб-додатку
- Розгорнути додаток на веб-сервер

Об’єкт дослідження: процес розробки веб-застосунків з можливістю використання зовнішніх інформаційних ресурсів.

Предмет дослідження: технології оптимізації пошукових систем дистанційних курсів.

Методи дослідження: теоретичні; практичні; узагальнення; аналіз; порівняння; синтез; емпіричні.

Наукова новизна дослідження полягає в тому, що вперше було розроблено веб-ресурс з можливістю здійснення пошуку навчальних курсів з конкурентних освітніх платформ.

Практичне значення. Розроблений прикладний програмний інтерфейс та односторінковий веб-застосунок може бути використаний студентами для оптимізації пошуку навчальних курсів.

Апробація результатів роботи. Результати кваліфікаційного дослідження були представлені та обговорені на XIV Всеукраїнській науково-практичній конференції «Інформаційні технології в професійній діяльності» (м. Рівне, 1 листопада 2021 р.) та опубліковані в роботі [2].

Структура роботи. Кваліфікаційна робота складається зі вступу, двох основних розділів з підрозділами, висновків та списку використаних джерел. Загальний обсяг роботи становить 55 сторінок.

РОЗДІЛ І. АНАЛІЗ ОСВІТНІХ ПЛАТФОРМ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

1.1 Можливості та переваги використання дистанційних курсів

За останні роки різко зросла популярність дистанційної освіти. Необхідність запровадження такого виду освіти як в Україні так і в усьому світі продиктована існуючими викликами пандемії та військовими діями. Така форма навчання - найбільш гнучка. Вона доступна для багатьох бажаючих отримати знання. На даний момент вже було багато сказано про переваги та недоліки дистанційної форми навчання. Таким чином виникає необхідність проаналізувати, вагомість причин впровадження дистанційного навчання в наше повсякденне життя.

Дослідженню можливостей використання та вивченню переваг дистанційних курсів присвячено ряд наукових праць.

Наприклад, у роботі [3] було проведено аналіз організації системи дистанційної освіти за допомогою українських масових відкритих онлайн-курсів в умовах карантину. Охарактеризовано найбільш популярні українські освітні платформи онлайн-освіти: Університет онлайн, ВУМ online, Освіта-онлайн, EDUGET, Prometheus, LINGVA SKILLS, Освітній Хаб міста Києва, EdEra, Brainbasket й ін.

Питання, пов’язані з використанням можливостей певних хмарних сервісів у процесі навчання здобувачів вищої освіти вивчаються в [2]. У роботі [4] проаналізовано міжнародний досвід організації дистанційного навчання в окремих європейських країнах в умовах пандемії на основі даних, розміщених на офіційних освітніх сайтах. Охарактеризовано досвід тих європейських країн, які: демонструють високий рівень розвитку цифровізації (Австрія, Велика Британія); мають дуже високі показники захворюваності і смертності в період пандемії (Італія, Іспанія, Франція); мають порівняно низький рівень цифровізації (Болгарія, Хорватія, Чехія). У статті [5] обґрунтовано теоретичні

засади трьох типів навчання: е-навчання, дистанційне та змішане; проаналізовано досвід використання змішаного стилю викладання курсу «Візуальне програмування», де за платформу було використано LMS Moodle. Також було з’ясовано як позитивні, так і негативні аспекти реалізації курсу за допомогою веб-аналітики.

Суттєвими перевагами дистанційної форми навчання можна відзначити наступні:

- **Можливість навчатися у будь-який час.** Завдяки тому, що студент може вільно будувати для себе графік навчання, то він самостійно вирішує коли і скільки часу упродовж семестру йому приділяти на вивчення матеріалу. Це може значно підвищити рівень опановування матеріалу.
- **Можливість навчатися в будь-якому місці.** Для того, щоб приступити до навчання, необхідний лише комп'ютер з доступом в Інтернет, а отже, будь-який студент може навчатися вдома, на свіжому повітрі чи навіть на іншому боці планети. Також великою перевагою даного типу навчання є відсутність необхідності щодня відвідувати навчальний заклад, наприклад, для людей з обмеженими можливостями здоров'я чи студентів, які проживають у віддалених населених пунктах.
- **Навчання без відриву від основної діяльності.** Тепер зовсім не обов'язково брати відпустку на основному місці роботи, адже завдяки дистанційному навчанню можна отримувати нові знання на декількох курсах чи навіть у декількох навчальних закладах одночасно не покидаючи офіс.
- **Можливість навчатися у своєму темпі.** Студент може самостійно обрати темп вивчення нового матеріалу, оскільки немає потреби навчатися в тому ж темпі, що і решта учнів. Тепер можна самостійно повертатися до тих запитань, які викликали труднощі та присвятити їм більше часу на вивчення та засвоєння. З легкістю можна віднайти пройдений матеріал та повторити його у будь-який час.

- **Доступність навчальних матеріалів.** У наш час студенту потрібно лише зареєструватись в системі дистанційного навчання або мати електронну скриньку. Вся необхідна література та матеріали надаватимуться студенту в онлайн режимі. Така проблема як нестача чи відсутність підручників, навчальних посібників, методичок – зникає.
- **Мобільність.** Зв'язок з викладачами та репетиторами можна підтимувати як on-line, так і off-line. Тепер для того, щоб проконсультуватися з викладачем можна написати йому на електронну пошту і це буде ефективніше та швидше, ніж призначити особисту зустріч при очному або заочному навчанні.
- **Навчання в спокійній обстановці.** У студентів менше причин відчувати хвилювання на проміжних атестаціях дистанційних курсів, тому що вони проходять у формі on-line тестів. Виключається можливість суб'єктивної оцінки. На систему, яка перевіряє правильність відповідей на питання тесту, не впливає ні успішність студента з інших предметів, ні його соціальний статус та і будь-які інші чинники.
- **Індивідуальний підхід.** Оскільки студент сам обирає темп власного навчання, він може оперативно отримати від викладача відповідь на конкретне запитання і ефективно продовжити опановувати знання. В університеті викладачеві досить важко приділити необхідну кількість уваги всім студентам групи через брак часу.
- **Дистанційна освіта дешевша.** У порівнянні вартості форм заочного та дистанційного навчання, то друга, скоріш за все, буде дешевшою. Студенту потрібно платити лише за самий матеріал навчання та власні матеріали для навчання. Зникає потреба оплачувати дорогу чи проживання у гуртожитку. Якщо брати до уваги зарубіжні вузи, то тепер не потрібно витрачатися на візу та закордонний паспорт.
- **Зручність для викладача.** Тепер усі, хто займається педагогічною діяльністю можуть приділяти увагу більшій кількості студентів

дистанційно. Також можна працювати у сфері освіти, навіть перебуваючи у відрядженні чи на конференції за кордоном. [6].

1.1.1 Аналіз існуючих платформ для пошуку навчальних курсів

Всесвітня онлайн платформа для дистанційного навчання Coursera, заснована в 2012 році Стенфордськими професорами інформатики Ендрю Ін і Дафною Коллер. Вона працює з університетами та іншими організаціями. Пропонує власні онлайн-курси, спеціалізації та ступені з різних предметів, наприклад інженерія, машинне навчання, гуманітарні науки, бізнес, інформатика, математика, медицина, цифровий маркетинг, біологія, соціальні науки, та інші.

Coursera надає можливість отримання знань та досвіду від найкращих викладачів усвоїй сфері діяльності. Більш того, багато хто з них також мають власний бізнес крім викладання.

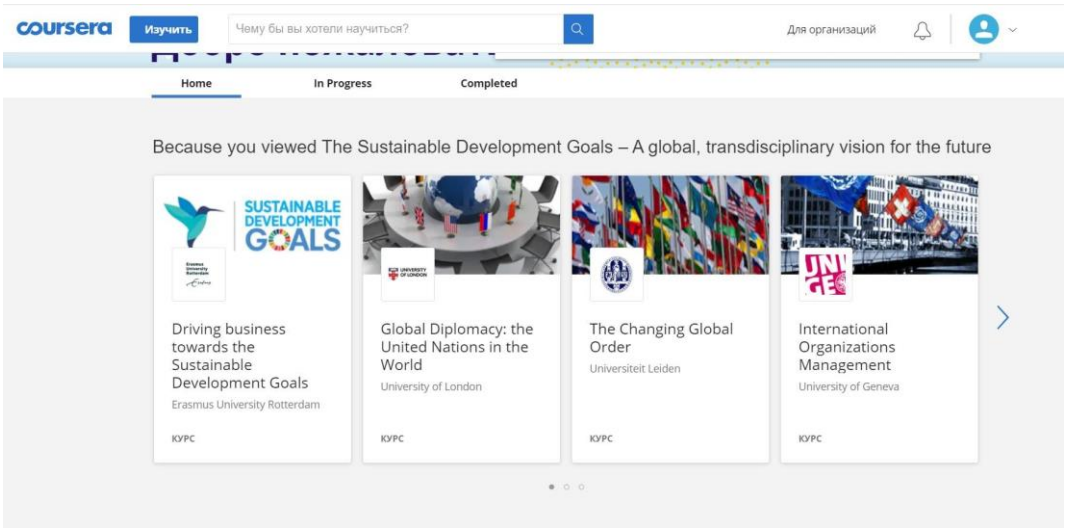

Рисунок 1.1 Інтерфейс веб системи Coursera.

Проте обмеження цього формату очевидні. Дива не вийшло, і замінити одним першокласним викладачем сто звичайних не вийде. Coursera не підходить для отримання потрібного обсягу знань, у порівнянні зі стандартним бакалавром, тому вона не може замінити отримання звичного диплому. А

також один із недоліків є те, що дана веб система не надає точного та повноцінного моніторингу отриманих знань.[7].

Udemy - американська платформа онлайн-навчання заснована в травні 2010 року, призначена як для дорослих, так і для студентів. Статистика на січень 2020 року вказує на те, що на платформі навчалися понад 50 мільйонів студентів та 57 000 викладачів на більш ніж 65 мовах. За весь час було зареєстровано понад 295 мільйонів студентів та викладачів з більш ніж 190 країн. До того ж 2/3 студентів за межами США.

Зазвичай студенти беруть курси як засіб підвищення професійних навичок. Деякі курси наголошують увагу технічної сертифікації. «Удемо» доклала неймовірних зусиль для залучення корпоративних тренерів, які не лише прагнуть, а і створюють курсову роботу для співробітників своєї ж компанії. Станом на 2020 рік на сайті налічується понад 150 000 різних курсів.

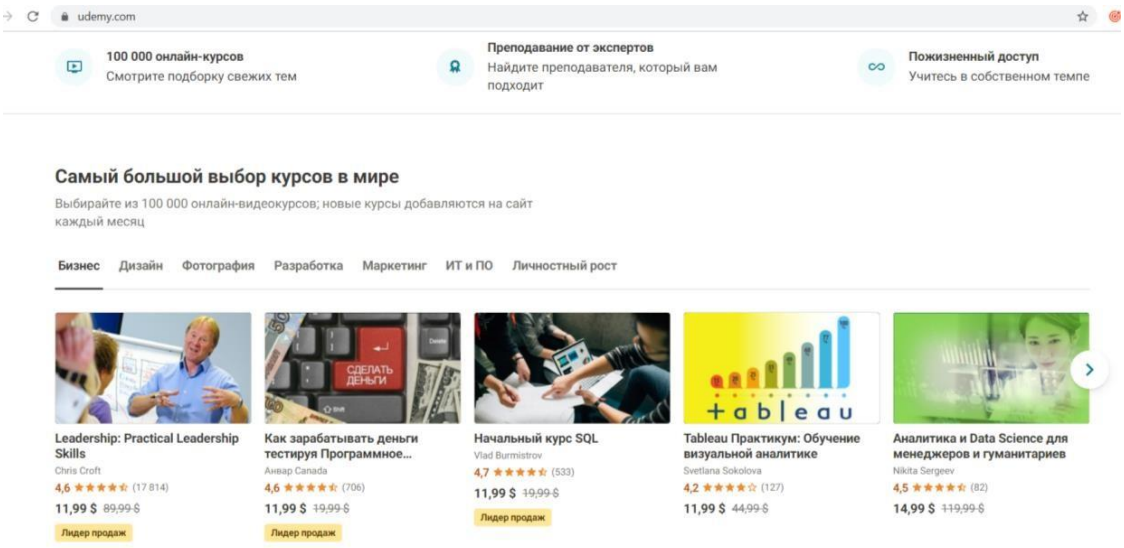

Рисунок 1.2 Інтерфейс веб системи Udemy

Серйозною проблемою є те, що будь-який користувач може стати викладачем, а отже буде багато неякісних та непрофесійних курсів. Більшість авторів мають добрі наміри, але оскільки вони не завжди є експертами в цих галузях, то і навчати інших вони можуть неякісно [8].

Prometheus – волонтерський проект масових безкоштовних онлайн-курсів. На жаль, він неприбутковий. За досить маленький період існування платформа курсів зібрала 100 тисяч слухачів. І це навчання заради власного майбутнього, а не заради папірця чи галочки в дипломі.

Проєкт Prometheus називають волонтерським, народним та революційним. Він дуже швидко набуває популярності, не дивлячись на те, що створений ентузіастами і волонтерами, не лише серед студентів, а й серед тих, хто давно й успішно працює у певній сфері. Дану ідею таких масових безкоштовних онлайн-курсів помітили. Засновник Prometheus, Іван Примаченко аспірант КНУ ім. Шевченка увійшов до списку 30-ти успішних українців віком до 30 років, укладеного журналом Forbes.

Рисунок 1.3 Платформа Прометеус

Prometheus узяв формат не лише курсів Coursera, edX, а й інших провідних онлайн-платформ західних університетів. Проте мова наших курсів українська. Це надзвичайно важливо не тільки тому, що не так багато українців володіє англійською, а й тому, що саме рідною мовою подаються деякі специфічні курси, такі як історія України чи українське право. Проте також планується запуск бізнес-англійської для українців.

Також є ще одна достатньо велика відмінність Prometheus від Coursera та інших. Coursera не створює курсів, оскільки університети просто надають готові лекції, а платформа їх розміщує. Тих, хто адмініструє платформу, не

цікавить, ні те як готують, ні те як знімають та монтують лекції. В свою чергу ми майже повністю виробляємо матеріали для навчання самі – від пошуку викладачів, створення планів лекцій, зйомок, ілюстрування, монтажу та дизайну і до розміщення. Українські університети, на жаль, поки що не можуть ні взяти на себе весь цикл даного виробництва, ні підтримувати його у необхідних масштабах.[9]

1.2 Добір оптимальних технологій розробки веб-додатку

Для добору оптимальних технологій розробки веб-додатку варто окремо і детально розглянути технології Front-end та Bback-end.

1.2.1 Front-end

Даний напрям займається питаннями, які стосуються клієнтського інтерфейсу. Абсолютно все, що користувач бачить (шрифт, оформлення тощо) на моніторі та з чим він може взаємодіяти, відноситься до області впливу front-end. Організація роботи сервера, реалізація логіки веб-продукту, рішення інших завдань, які приховані від очей юзера, - сфера компетенції back-end.

Щоб додаток функціонував ефективно, важливо забезпечити грамотний розподіл функцій усередині команди з урахуванням специфіки кожного напряму. Розглянемо аспекти які включать в себе взаємодію з користувачем веб-застосунку:

- створення естетично привабливої картинки, де кожен елемент знаходиться на своєму місці;
- розробка простого, зрозумілого кінцевому користувачеві функціоналу для реалізації призначення додатка;
- отримання клієнтських запитів з відправкою у back-end і формуванням відповідного відгуку.

У розробці веб-застосунків front-end і back-end визначаються як розділення відповідності між рівнем презентації (front-end) та бізнес логікою

(back-end) веб-застосунку або фізичною інфраструктурою та обладнанням. Зазвичай, моделлю клієнт-сервер клієнтом вважають front-end, а сервер вважається back-end.

UX [10] (User Experience) являється важливим фактором при виборі продукту користувачами. Саме для кросс-платформеної розробки стек JavaScript технологій використовується (один і той же веб-застосунок без перешкод запускається як на Android, так на iOS і в браузері).

У ході роботи front-end- програміст стикається з необхідністю співпраці з фахівцями різних галузей. Текстовий, графічний контент, верстання і фронтенд нерозривно пов'язані. Тому він повинен взаємодіяти з авторами публікацій, дизайнерами, маркетологами, прагнучи грамотно об'єднати усі блоки в єдине ціле і змусити їх злагоджено працювати. Від цього значною мірою залежить комерційний успіх проєкту.

Проте головні інструменти, які потрібні йому в роботі, зводяться до трьох позицій:

- HTML;
- CSS;
- JavaScript.

Інструмент, що дозволяє зробити взаємодію з користувачем живим, динамічним. Ця мова програмування обробляє інформацію, що поступає від юзера :

- кліки мишею;
- натиснення клавіш;
- переміщення курсора;
- передає запити на сервер.

Front-end це у тому числі дизайн сайту, тому потрібний універсальний код, використовуваний для зовнішнього відображення контент, він відповідає за формування розміру, кольору, стилю шрифту, створення фону, розміщення блоків на сторінці, переформатування документів для друкарської версії або для читання. Для цього застосовується мова розмітки, яка дозволяє

сформувати правильне верстання на сайті. З її допомогою текстовий контент розбивається на розділи, абзаци, списки, в структуру сторінки впроваджуються таблиці, малюнки, графіки, заголовки і так далі. Правильне застосування HTML дасть можливість кожному елементу знаходитися в потрібному місці для органічного сприйняття користувачем.

У сучасному підході до створення інтерфейсу для кінцевого користувача – front-end, як і в цілому кожна структура майже будь-якого забезпечення, представляється у вигляді ієрархічної структури. На верхньому рівні завжди знаходяться блоки, кожен з яких має якусь власну реалізацію, яка дозволяє створювати взаємодію з іншими блоками. У кожному з них виділяються модулі, які, в свою чергу, складаються з методів. Кожна частина даної ієрархії має свій спосіб комунікації та взаємодії, представлені як вхідні і вихідні параметри веб-застосунку.

Сучасна архітектура Front-end базується на декількох принципах

- потік даних займає центральне місце в архітектурі;
- компонентний підхід;
- архітектура на основі компонентів.

Масштаби та складність середовища, в якому виконується front-end, бізнес логіка визначається як ключовий. За спостереженнями, більша частина розробки витрачається на пошук необхідної інформації та рефакторинг коду. В даному випадку, важливою умовою для будь-якої архітектури є знаходження необхідного нам коду, та безпосередньо, швидкість його знаходження. Найнеобхідніше – це визначити відповідальну частину бізнес логіки. Доволі важливо розуміти, що вносячи якусь нову функціональністьв систему, слід прогнозувати як вона вплине на майбутню працездатність та безпосередню ефективність програмного забезпечення. Розуміння на кожному етапі механізму переміщення даних по архітектурі програмного забезпечення, являється єдиним способом для здійснення такого роду завдання. Тому, побудування любого Front-end фреймворку сьогодні базується на цьому принципі, на чіткому розподілі на модуль State (зберігання і маніпуляція з

даними) і на модуль View (відображення даних).

Як приклад можна навести такий фреймворк як Angular, який використовує принцип двонаправленого потоку даних, що створює неконтрольовані процеси при передачі даних. Поява патерну проектування Flux, зумовила використання більш надійного принципу односпрямованого потоку даних. По таким причинам, був створений Angular II, який коригувався як раз принципом однонаправленістю, як Flux.

Компонентний підхід є чітким наслідком першого пункту про потік даних. Традиційні архітектури front-end застосунків спрямовані на горизонтальний розподіл функціональних забов'язань. Будь-який блок коду, що інкапсулює якусь одну «pure responsibility» логіку, може бути як причиною, що призвела врахування першого принципу.

Елементи тіла документу призначені для управління відображенням інформації в програмі інтерфейсу користувача. У даному випадку, компонентний підхід дозволяє пере використовувати і тестувати окремі блоки веб-застосунку, не призводивши до видозмін його архітектури.

Беручи до уваги сучасні підходи, компоненти повинні бути:

- незалежні;
- слабкозв'язані;
- перевикористовані.

Елементи–контейнери ніякої дії браузеру не викликають, а використовують тільки для логічного ділення HTML документу на частини.

Компоненти можуть бути досить складні в середині, але мають бути прості зовні. Тут мається на увазі, що інтерфейс будь-якої компоненти повинен бути настільки простим, щоб процес підключення компоненти до батьківського блоку проходив без побічних ефектів.

У контейнери (між початковим і закриваючим тегами) розміщують інші елементи, якими потрібно керувати спільно. Такі елементи створюються за допомогою тегів div і span. За допомогою тегу div створюється блоковий контейнер, який розпочинається з нового рядка, і елемент, що слідує за нього,

також буде розпочатий з нового рядка. А за допомогою тегу створюється рядковий елемент, який розміщується в тому ж самому рядку, що і текст, що оточує його. Основними атрибутами
контейнерів є Id і Class. Слід пам'ятати, що рядкові контейнери можуть включати тільки інші рядкові контейнери, а блокові контейнери можуть включати як рядкові, так і блокові контейнери. Елементи-контейнери дозволяють застосовувати правила каскадних таблиць стилів(CSS) до вмісту HTML документу.

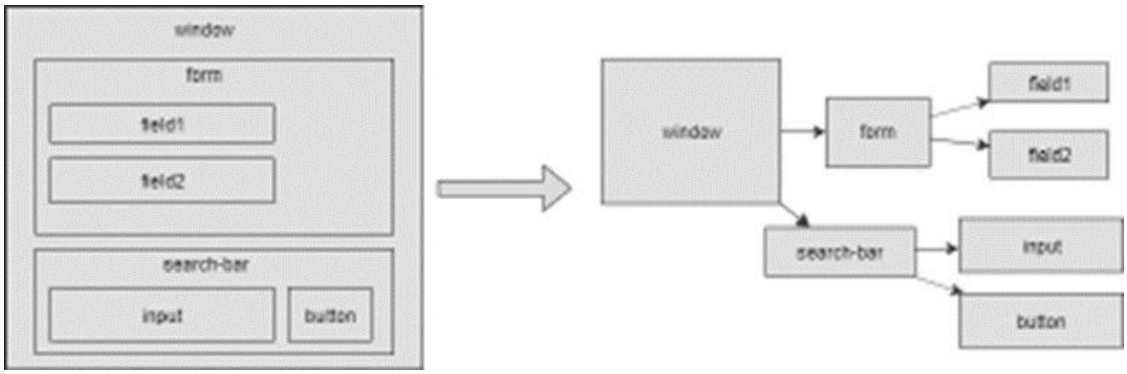

Рисунок 1.4 Приклад декомпозиції View

Певне оновлення DOM є досить складним за участі ресурсів та часу. Тому в такий момент front-end спільнота дійшла до ідеї створення певної технології, яка б виконувала мінімальну для DOM кількість дій по переутворенні певних конкретних DOM елементів при зміні даних веб-застосунку, в певний State. Результатом таких змін стала поява певних підходів, які буде розглянуто далі.

Проблеми взаємозв'язку даних (Model) і уявлення (View), являються основними для DOM структури. Тому, Data-binding між Model і View у поєднанні з компонентним підходом використання є ідеальним вирішенням проблеми взаємозв'язку даних (Model) і уявлення (View). View можна уявити функцією моделі, яка являється саме View та поля даних, які потрібні їй для

відтворення необхідних їй компонентів. Як приклад, фреймворк Angular використовує низку шаблонів, в React JSX – мова, що є сумішшю HTML і JavaScript мов.

Одна з вимог це можливість архітектурного рішень будь яким чином стежити за тим, які поля стану інтерфейсу були оновлені, видалені чи додані, саме це включає в себе оповіщення про зміну даних (Change detection). Перевірка в нескінченному циклі через певний період часу, які поля були змінені, представлена як можливість в Angular і реалізована через метод брудної перевірки (dirty checking). Тому, оновлення необхідних полів стану інтерфейсу у фреймворку React відбувається через незмінні структури даних і за допомогою подвійної шини відбувається оновлення. У майбутньому Vue3 планується використання об'єкта певного шару (proxy) для реалізації data checking (перевірка даних на зміну).

Перемальовування DOM можливе після виявлення змін стану, на основі яких необхідно виконати певні дії. Саме для цього у React використовується технологія VirtualDOM [11].

Отже, ключовими вимогами для розробки front-end веб-застосунку є можливість додавання нових функції з мінімальними впливами або зовсім без впливів на його внутрішню структуру інтерфейсу та потік даних.

Модифікованість, ремонтопридатність та масштабованість – основні якості розширюваної програмної системи.

Поділ функціональної архітектури відбувається таким чином, щоб зміна системи передбачала би найменшу кількість змін в найменшій кількості можливих елементів визначається як модифікованість.

Ремонтопридатність, що визначено як термін «Соммервіллом», полягає в створенні системи, якій необхідно враховувати введення нових додаткових вимог без ризику додавання новостворених помилок.

Масштабованість, що являється здатністю системи до розширення у вибраному режимі, без великих змін в її архітектурі. Масштабована система може обробляти більше даних з мінімальним впливом на продуктивність.

Модульність, тобто поділ програмної системи на кілька незалежних модулів, які можуть самостійно виконувати одне чи багато завдань. Ці модулі можуть функціонувати як основні елементи для всього ПЗ. Модулі створюються з окремих задач. Переваги модуляції включають покращену ремонтопридатність та функціональність програмного забезпечення модулів, бажаний рівень абстракції, високу згуртованість, багаторазове використання та підвищену безпеку.

Архітектура на основі компонентів – це функціонал в якому кожен компонент має необхідність створення для взаємодії з іншими компонентами та їх повторне використання у веб - застосунку.

Формування даних вимог створювались саме з точки зору розробників та внутрішнього влаштування веб-застосунку. І що важливо, front-end представляє собою частину ПЗ, яку бачить користувач. Саме через неї він отримує уявлення про роботу даного застосунку, тому до цієї частини ще висуваються окремі вимоги.

Правильна структуризація веб-сторінки. HTML є корсетом вебсайтів, на основі якого здійснюється пошукова оптимізація, тому життєва важливо встановити правильну структуру документа для класів та ідентифікаторів, які забезпечать стиль та взаємодію.

Саме погана семантична структура веб-сторінки досить часто є джерелом багатьох помилок в роботі front-end.

Унікальний візуальний стиль веб-сторінок, який також впливає на структурованість є важливим аспектом стилю є перевірка в декількох браузерах та написання стислого, лаконічного коду, який є специфічним, але загальним і одночасно відображає якомога більше рендерів.

Кросплатформеність. Необхідно передбачати можливість того, що користувачі будуть використовувати різні браузери та різні комп’ютери, що може суттєво вплинути на відображення веб-сторінок.

Юзабіліті – характеристика, яка оцінює взаємодію програмного забезпечення з користувачем. Взаємодія застосунка з людиною повинна бути

зрозумілою, зручною, достатньо простою для сприйняття.

Продуктивність. Для створення веб-застосунків, що швидко завантажуються, розмітка, стилі та JavaScript повинні бути масштабованими. Необхідно використовувати скорочувати розміри сторінок там, же цеможливо.

Виходячи з цього front-end застосунок повинен відповідати таким критеріям:

- у веб-застосунку має використовуватися лише попередньо визначена кількість загальних макетів;
- для кожного випадку використання, вже використовувані елементи взаємодії з користувачем повинні бути однаковими;
- у кожному випадку використання, розміщення елементів взаємодії з користувачем має бути однаковим;
- розміри елементів інтерфейсу користувача повинні бути однаковими для кожного випадку використання;
- стан та відгуки про взаємодію користувачів повинні відображатися однаково для кожного випадку використання;
- реалізація необхідна для усіх використовуваних елементів користувача, використовуючи лише попередньо визначений колір схем.

JavaScript - це динамічна мова комп'ютерного програмування. Вона легка, тому найчастіше використовується як частина веб-сторінок, реалізація яких надає дозвіл клієнтському сценарію взаємодіяти з користувачем і створювати динамічні сторінки. Це інтерпретована мова програмування з об'єктно-орієнтованими можливостями.

JavaScript [12] вперше була відома як LiveScript, але Netscape змінив назву. Вперше JavaScript з'явилася в Netscape 2.0 у 1995 році під назвою LiveScript. Було вбудовано в Netscape, Internet Explorer та інші веб-браузери ядро мови загального призначення.

Специфікація ECMA-262 визначила основної мови JavaScript стандартну версію: JavaScript є легкою та інтерпретованою мовою програмування;

- призначений для створення мережевих орієнтованих додатків;
- доповнює та інтегрується з Java;
- доповнює і інтегрується з HTML;
- відкрита і крос-платформна.

На стороні клієнта JavaScript є найпоширенішою формою мови. Сценарій повинен бути включений або посилатися на HTML документ, для того щоб код був інтерпретований браузером.

Це означає, що веб-сторінка може включати програми, які взаємодіють з користувачем, керують браузером і динамічно створюють вміст HTML, але не повинна бути статичним.

На стороні клієнта JavaScript механізм надає багато переваг перед традиційними сценаріями на стороні сервера CGI. Наприклад, ви можете скористатися JavaScript, для перевірки, чи користувач впише дійсну адресу електронної пошти у полі форми.

Код JavaScript виконується, коли користувач подає форму і якщо всі записи дійсні, то вони подаватимуться на веб-сервер.

Кліки кнопок, навігація посилань та інші дії, які користувач ініціює явно або неявно – саме для цього JavaScript може використовуватися - для уловлювання ініційованих користувачем подій.

Переваги використання JavaScript:

- менша взаємодія з сервером – перед відправкою сторінки на сервер, що економить навантаження на сервер ви можете перевірити вхід користувача;
- зворотний зв’язок з користувачами, яким не потрібно чекати, щоб побачити, що вони мають змогу щось ввести поки сторінка перезавантажиться;
- підвищена інтерактивність – ви можете створювати інтерфейси, коли користувач зависає над ними за допомогою миші або активує їх за допомогою клавіатури, які реагують;
- більш чудові інтерфейси –щоб включити такі елементи, як компоненти

перетягування і повзунки, ви можете використовувати JavaScript, щоб надати відвідувачам свого сайту багатий інтерфейс.

Як повноцінну мову програмування ми не можемо розглядати JavaScript. Вона не має таких важливих функцій:

- JavaScript на стороні клієнта не дозволяє читати або писати файли. З міркувань безпеки це зберігається;
- для мережевих програм JavaScript не можна використовувати, оскільки такої підтримки немає;
- відсутні багато-поточні або багатопроцесорні можливості у JavaScript;
- знову ж таки, JavaScript є легкою, інтерпретованою мовою програмування,яка дозволяє у статичних HTML-сторінках створювати інтерактивність.

Бібліотека React.js [13] – це бібліотека JavaScript, яка використовується для створення користувацьких інтерфейсів спеціально для односторінкових додатків з відкритим вихідним кодом. Використовується для того щоб обробляти шар уявлення для веб-додатків і мобільних додатків. Також React дозволяє нам створювати використовувані повторно компоненти користувальницького інтерфейсу. React був створений Джорданом Уолке, інженером-програмістом, який працював в Facebook. Вперше React був розміщений на новинній стрічці Facebook в 2011 році і в 2012 році на Instagram.com.

Розробникам React надає можливість створювати великі веб-додатки, які можуть змінювати дані без перезавантаження сторінки. Основна мета React – бути простим, швидким, та таким, що масштабується. Він працює лише на призначених для користувача інтерфейсів в додатку. Це відповідає уявленню в шаблоні MVC. Він може використовуватися з комбінацією середовищ, таких як Angular JS в MVC або інших бібліотек JavaScript.

Тепер, чому слід використовувати ReactJS це головне питання, яке виникає перед нами. Існує дуже багато платформ, таких як Angular, з відкритим кодом для полегшення розробки веб-додатків. Давайте розглянемо

переваги React у порівнянні з іншими конкурентними структурами або технологіями. На щоденній основі зі зміною front-end-у важко присвятити час вивченню нових рамок, особливо коли ці рамки в кінцевому підсумку можуть зайти в глухий кут. Отже, якщо ви шукаєте наступну кращу річ, проте відчуваєте себе трохи загубленим у джунглях, я пропоную перевірити React.

React.js простіше зрозуміти відразу. Чітко визначений життєвий цикл, компонентний підхід і використання простого JavaScript дозволяють навчитися, реагувати дуже просто, створювати професійні веб-сайти (та мобільні програми) і підтримувати його. React використовує спеціальний синтаксис під назвою JSX, який дозволяє змішувати HTML з JavaScript. Це не є обов'язковим. Розробник все ще може писати на звичайному JavaScript, але JSX набагато простіше у використанні.

Той, хто володіє в області програмування базовими знаннями, може легко зрозуміти React, в той час як Angular і Ember називаються "предметно-орієнтованою мовою", маючи на увазі, що їх важко вивчати. Вам просто необхідні базові знання CSS і HTML для React.js.

React можна використовувати для створення мобільних додатків (React Native). А React – відданий прихильник можливості повторного використання, що означає те, що в ньому підтримується велике використання коду повторно. Так що в той самий час ми можемо зробити Android, IOS і веб-додаток.

Архітектура додатку, так звана Flux, контролює потік даних до компонентів через диспетчер – одну контрольну точку, а React використовує однобічну прив'язку даних. Автономні компоненти великих додатків ReactJS простіше налагоджувати.

Жодної концепції вбудованого контейнера для залежності React не пропонує. Ви можете використовувати модулі EcmaScript 6, Browserify, Require JS, які ми звісно ж можемо використовувати щоб автоматично вводити залежності через Babel, ReactJS-di.

Дуже легко перевірити програми React.js. Ми можемо маніпулювати станом, оскільки реактивні види можуть розглядатися як функції держави, тому, ми переходимо до перегляду React.js та подивимося на вивід і

спрацьовування дії, події та функції.

Віртуальний DOM – це така концепція програмування, у якій ідеальне чи так зване «віртуальне» подання UI зберігається в пам'яті та синхронізується з так званим «реальним», наприклад DOM бібліотекою, такою як ReactDOM. Такий процес називається примиренням.

Віртуальний DOM – це відображення в пам'яті Real DOM. Це легкий об'єкт JavaScript, який є копією Real DOM. ReactJS безпосередньо оновлює не Real DOM, а Virtual DOM. Це і спричиняє велику користь для ReactJS.

Маніпуляція DOM – це серце інтерактивної, сучасної мережі. Це також, на жаль, набагато повільніше, ніж більшість операцій JavaScript. Дана повільність ускладнюється ще й тим фактом, що більшість фреймворків JavaScript набагато більше, ніж вони повинні оновлюють DOM.

Наприклад, у вас є список з десяти елементів. Ви перший елемент відмічаєте. Більшість JavaScript фреймворків відновить весь список. Це буде навіть в десять разів більше, ніж потрібно було! Зміниться лише один елемент, проте решта дев'ять перебудовуються саме так, як раніше.

Відновлення списку для веб-браузера не має великого значення, але величезну кількість маніпуляцій DOM можуть використовувати сучасні веб-сайти. Серйозною проблемою стало неефективне оновлення.

Люди в React, щоб вирішити цю проблему, популяризували те, що має назву віртуальний DOM. Для кожного об'єкта DOM, у React існує відповідний "віртуальний об'єкт DOM". Як легка копія, віртуальний об'єкт DOM є поданням об'єкта DOM.

Ті ж властивості, що й реальний об'єкт DOM має й віртуальний об'єкт DOM, але йому не вистачає реальної можливості змінювати безпосередньо те, що знаходиться на екрані. Відбувається повільно маніпулювання DOM.

Маніпулювання віртуальним DOM набагато швидше, оскільки на екрані нічого не відбувається. Подумайте як редагування плану про маніпулювання віртуальним DOM, на відміну від переміщення кімнат у будинку даному фактом.

Коли ви уявляєте елемент JSX, то оновлюється кожен окремий об'єкт

віртуального DOM. Хоча це і звучить неймовірно неефективно, проте така вартість незначна, це все тому, що віртуальний DOM може так швидко оновлюватися. React порівнює віртуальний DOM після оновлення віртуального DOM, зі зробленим безпосередньо перед оновленням знімком віртуального DOM. React з'ясовує, порівнюючи нову віртуальну DOM з попередньою версією, які саме змінилися віртуальні об'єкти DOM. Даний процес має назву «розбіжний».

Як тільки React дізнається, які об'єкти віртуального DOM змінилися, то оновлює ці об'єкти, а саме ті, що є на реальному DOM. У нашому прикладі з React, щоб відновити ваш один відредагований список-елемент та залишити решту списку поодинці було б достатньо розумним.

Це має неймовірне значення! Тільки необхідні частини DOM може оновлювати реакція. З цією інновацією значною мірою пов'язана репутація за ефективність компанії React.

Загалом, ось що відбувається, коли ви намагаєтеся оновити DOM у React: увесь віртуальний DOM оновлюється;

- віртуальний DOM порівнюється з тим, як він виглядав, перш ніж оновитийого. Реагуйте, які об'єкти змінилися;
- змінені об'єкти і тільки змінені об'єкти оновлюються на реальному DOM;
- зміни на реальному DOM призводять до зміни екрана.

У минулому, продуктивність JavaScript була слабкою, кожна сторінка надходила з сервера, в свою чергу браузери були набагато менш здатними, ніж сьогодні. Коли ви кожного разу щось натискали, новий запит був зроблений на сервер, і лише згодом браузер завантажив нову сторінку.

Тільки надзвичайно інноваційні продукти працювали інакше і проводили експерименти з новими підходами. Популяризуючи сучасні фреймворки JavaScript сьогодні, наприклад такі як React, зазвичай програма створюється як додаток для однієї сторінки. Ви завантажуєте код програми (HTML, CSS, JavaScript) тільки один раз, і що зазвичай відбувається

коли взаємодієте з програмою JavaScript перехоплює події веб-переглядача і замість того, щоб зробити новий запит на сервер, який потім повертне новий документ, виконує дію на сервері або клієнт запитує деякий JSON, проте та сторінка, яку бачить користувач на той час, не буде повністю знищена ніколи, та веде себе більше як настільний додаток.

Для однієї сторінки програми вбудовані або, хоча б, зібрані в JavaScript і працюють у браузері. Технологія завжди однакова, проте деякі основні компоненти того, як працює дана програма чи філософія - різні. SPA користувач відчуває набагато швидше користування, а все тому, що замість очікування, що відбудеться за допомогою зв'язку типу клієнт-сервер та очікувати, поки веб-переглядач знову ж відобразить сторінку. Зараз можна отримати миттєвий відгук. Це відповідальність виробника програм, але ви можете мати не лише переходи та пряди, а і взагалі будь-яке поліпшення UX, що, звісно ж, набагато краще, ніж звичайний робочий процес.

Сервер споживатиме менше ресурсів на додаток до набагато швидшого використання досвіду користувачеві, тому що ви можете зосередитися на забезпеченні більш ефективного API, замість того, щоб створювати частини макетів на сервері. Якщо ви також створюєте мобільну програму поверх API, це робить його ідеальним, а все тому, що ви можете повністю використати повторно існуючий серверний код.

На прогресивні веб-програми легко перетворюються програми для однієї сторінки, що, в свою чергу, дає змогу для ваших служб локально кешувати та підтримувати автономний режим (або ще краще, якщо користувачі мають бути онлайн, повідомлення про помилку).

Коли немає необхідності SEO (search engine optimization) найкраще використовувати SPAs. Для програм, наприклад, які працюють за логіном. Вдосконалюючись щодня, пошукові системи, побудовані з підходом SPA, а не традиційними сторінками, що надаються сервером все ще мають проблеми з індексацією сайтів. Це стосується блогів. Не варто навіть створювати додаток для однієї сторінки, якщо ви збираєтеся покладатися на

пошукові системи, не маючи також частини, які їм надаватиме сервер.

Ви пишете багато JavaScript при кодуванні SPA. Вам потрібно буде приділяти більше уваги можливим витокам пам’яті, через те, що програма може бути довготривалою, – якщо ваша сторінка мала час життя в минулому, що вираховувалася в хвилинах, тепер SPA може залишатися відкритим протягом декількох годин час та якщо є будь-які проблеми з пам’яттю, що можуть збільшити використання пам’яті браузера набагато більше, і якщо ви не піклуєтесь про нього, то це призведе до повільного та достатньо неприємного досвіду.

При роботі в команді прекрасні SPA-центри. Back-end розробники можуть зосередитися лише на API, в той час, коли розробники інтерфейсу, використовуючи вбудований в back-end API, можуть зосередитися на створенні найкращого користувальницького досвіду.

Програми з однією сторінкою значною мірою покладаються на JavaScript в якості однієї сторінки. Це може призвести до негативного досвіду використання програми, яка працює на пристроях з низьким енергоспоживанням. Також потрібно розглянути доступність для всього, що ви збираєте, оскільки деякі відвідувачі можуть вимкнути JavaScript.

Деякі помітні приклади:

- Gmail;
- Google Maps;
- Facebook;
- Twitter;
- Google Drive.

URL-адреси потрібно керувати вручну, оскільки ви позбавляєтеся від навігації браузера за умовчанням.

Дана частина програми називається маршрутизатором. Наприклад такі фреймворки як Ember вже піклуються про них, а інші, наприклад, React Router вимагають бібліотек, які будуть виконувати таку роботу.

Так у чому ж проблема? Спочатку це було таким званим запізненням для

розробників, які з однієї сторінки створювали програми. Це призвело до виникнення загальної проблеми з «кнопкою зворотного ходу»: при навігації всередині програми URL не змінювався, тому що навігація за замовчуванням браузера вже була захоплена і натиснувши кнопку “назад” вона може перейти на веб-сайт, який був відвіданий давно.

За допомогою API-інтерфейсу History, який пропонують веб-переглядачі, тепер цю проблему можна вирішити, проте більшу частину часу ви будете використовувати бібліотеку, яка внутрішньо використовує цей API (наприклад ReactRouter).

Redux [14] – ні що інше, як сховище, яке містить стан програми. Коли розмір програми стає великим, щоб керувати станом кожного компонента вашої програми - це стає болісним завданням. Отже, підтримуючи та оновлюючи стан кожного компонента нашої програми, редукція приходить на допомогу.

Коли ми вперше пробуємо свої сили Redux часто буває незрозумілим. Я наведу приклад, щоб ви зрозуміли, що таке Redux і чому нам взагалі потрібні редукції. Все є компонентом у реактивному додатку. Уявіть, як стає важко, якщо у вашому додатку дуже багато компонентів, подібних до наведених на рисунку 1.5. Через це стає важко керувати потоком даних від батьківських до дочірніх компонентів.

Оскільки вона керує станами всіх компонентів для нас це перша причина, чому ми використовуємо Redux.

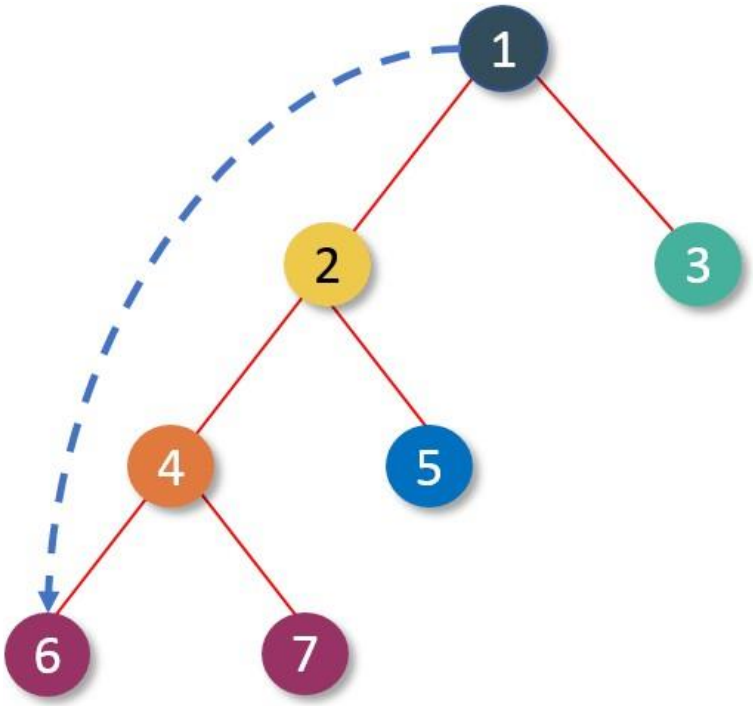

Рисунок 1.5 Зразок дерева компонентів React

Односпрямований потік даних існує у реактивному застосуванні. За односпрямованим я маю на увазі потоки даних від батьківських компонентів до дочірніх компонентів, а не навпаки. Ви посилаєте дані з батьківських компонентів до дочірніх компонентів у вигляді того, що ми називаємо реквізитами, і в цьому випадку цей дочірній компонент використання цієї підставки.

Redux – це бібліотека відкритого коду JavaScript для керування станом програми.

Після виконання дії в цьому випадку замовлення взуття з акцією, яке ви чекаєте на доставку, але чи відбувається це так, як тільки ви замовляєте щось із фліпкарту, ви отримаєте доставку відразу. Ні, насправді це вимагає часу, оскільки існує процес, який слідує кожен раз, коли ви замовляєте щось з вашого обраного веб-сайту.

Таким чином, подібно до Redux після виконання дії, існує термін, що має назву «диспетчер». Він посилає вашу дію на редуктор. Так само, як і після розміщення замовлення на сайті, ваш пакет з товаром буде відправлений

у найближчий склад на вашу адресу замовлення. Шляхом відправки така ж робота здійснюється в редукції. Для розуміння Redux існує декілька понять, які потрібно знати. Розглянемо їх за допомогою прикладу (рис. 1.6). Уявімо, що ви замовили пару взуття з фліпкарту. Після замовлення взуття, ви отримуєте його від агента доставки в певний час. Отже, ваше замовлення взуття – це дія, яка є однією зконцепцій Redux.

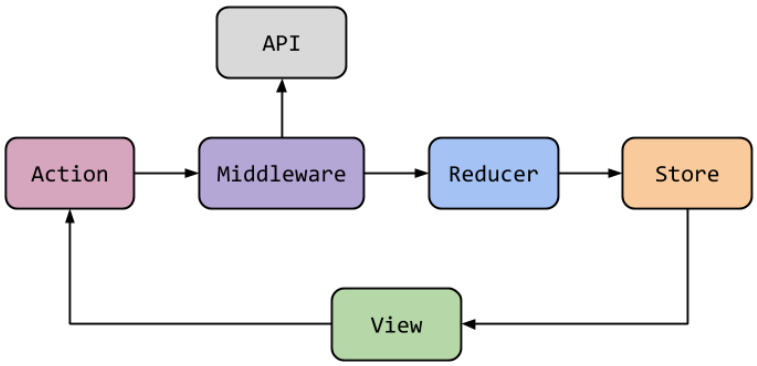

Рисунок 1.6 Архітектура Redux

Тепер Reducer дивиться на дію і відповідно до потреби робить те, що потрібно для зберігання даних. Редуктор – представлений у вигляді файлу, який містить в собі оператора case та виконує функції збереження даних у сховищі та повернення оновленого значення стану із глобального об’єкту стану. Тому, значення глобального об’єкту стану теж оновлюється, коли оновлюється стан.

1.2.2 Back-end

Мова програмування [15] C# – це одна з сходинок еволюції мов програмування. Створення мови програмування C# були викликане процесом покращення і адаптації, який визначав створення комп'ютерних мов протягом крайніх років. Схоже до всіх успішних мов, які побачили світ до сьогоднішнього дня, C# опирається на попередні звершення мистецтва програмування, яке зодня розвивається. У мові програмування C# (створеній корпорацією Microsoft для покращення середовища .NET Framework) загартовані часом засоби прикрашені за допомогою найсучасніших

технологій.

C# [16] пропонує дуже зручний і гнучкий спосіб створення програм для нового безпечного середовища розрахункової обробки даних, яка містить в собі операційну систему Windows, Internet та інші елементи. Протягом створення, мові C# передалася багата спадщина. Мова програмування C# – наслідником двох надзвичайно успішних інструментів програмування (C і C++). Розуміння властивостей цих взаємозв'язків дуже важливе для правильного сприйняття C#. C# включає багато нових інструментів, найзручніші з них пов'язані з наявною в мові підтримкою програмних елементів. Саме наявність вбудованих інструментів написання програмних елементів і дозволило мові C# прозвати себе компонентно-орієнтованою мовою програмування. Для прикладу, C# містить засоби, які безпосередньо підтримують основні частини елементів: властивості, функції та події. Все ж основною якістю компонентно-орієнтованої мови програмування є її можливість співпрацювати в середовищі мульти мовного програмування. Не дивлячись на те, що C # – самостійна мова програмування, вона місстить особливі зв’язки з середою .NET Framework. Для цього є дві причини. По-перше, C# це розробка корпорації Microsoft для написання коду, що запускається в середовищі .NET Framework. По-друге, в .Net середовищі наявні бібліотеки, що використовуються мовою програмування C#. І хоча C# від .NET Framework можна відокремити, два наявних середовища тісно зв’язані, через це вкрай важливо мати глобальне уявлення про середовище .NET Framework і сприймати, через що це середовище надзвичайно важливе для C#. Обгортка .NET Framework визначає середовище для програмування і виконання багато розподілених додатків, створених з використанням компонентних елементів. Мова C# дає можливість «мирно співіснувати» багатьом інструментам програмування і пропонує безпеку, універсальність програм і глобальну модель написання програм для платформи Windows. Досить важливо сприймати, що за своєю сутністю .NET Framework не прив’язаний до запуску в Windows, тобто програми, написані для неї, можуть потім бути перенесені в середовища, що відрізняються від Windows.

Спільність середовища .NET Framework з мовою програмування C# зумлена присутністю двох вкрай важливих інструментів. Одна з них, Common Language Runtime (CLR), представляє собою інструмент, який управляє виконанням написаних користувачем програм.

CLR – це одна з головних частин .NET Framework, що дає можливість програмам запускатись на різних платформах, дає можливість багатомовного написання програм і гарантує безпеку. Друга частина, бібліотека класів .NET-оболонки, надає програмам доступ до середи виконання коду. Для прикладу, якщо вам необхідно виконати операцію введення або виведення: зобразити якийсь контент на екрані, то для цього потрібно костуватися платформою .NET. В випадку обмеження програми використанням засобів, визначених .NET-бібліотекою класів, вона може запускатися на любій операційній системі, де присутня підтримка .NET системи. Оскільки C# використовує .NET-бібліотеку класів самостійно, програми написані мовою C# - автоматично відтворюються в усіх .NET середовищах. Система CLR керує виконанням коду написаного з .NET. Таким чином це відбувається. Після компіляції C# написаної програми результатом є не виконуваний код, а файл, в якому знаходиться спеціальний псевдокод, зо називається проміжною мовою Microsoft (Microsoft Intermediate Language - MSIL). Під MSIL розуміється набір функцій, які не залежать від варіанту процесора. Мета інструменту CLR - при виконанні написаного перетворити її проміжний код в виконуваний. Таким способом, програма, що може компілюватися MSIL, може бути запущена в будь-якій операційній системі, для чого створена CLR-система. В цьому і міститься суть середовища .NET Framework - можливість перенесення програм. Код, перероблений в проміжну мову Microsoft, перетворюється в виконуваний код з допомогою JIT – компілятора.

JIT – є скороченням від «just-in-time», що означає, що код виконуватиметься точно до конкретного моменту (так описується стратегія обрання рішення в край відповідний для цього момент виконання з метою гарантії їх максимальної якості). Виконання цього процесу виглядає наступним чином. Під час виконання програми написаної в середовищі .NET,

CLR-система вмикає JIT-компілятор, що перероблює MSIL-код в її «рідний» код в потрібному місці, через те, що важливо зберігати кожну частину виконуваної програми. В цей спосіб, C# програма виконуватиметься у вигляді сприйнятливого коду системою, не дивлячись на те, що спершу вона була скомпільована в MSIL-код. Під цим розуміється, що код буде виконано з тією ж швидкістю, як якби вона першочергово була скомпільована з присутнім «рідним» кодом, але з доповненнями переваг виконуватися на різних системах від переробки в MSIL-код. Результатом компіляції допоміжної програми окрім MSIL-коду створюються також метадані (metadata). Вони являють собою опис даних, що використовуються в програмі, і надають можливість коду співпрацювати з іншим кодом. Метадані знаходяться в тому ж файлі, де знаходиться MSILкод. В звичайному випадку при написанні допоміжної програми формується код, що називається керованим (managed code). CLR-система займається управлінням керованого коду. Результатом такого виконання є як чималі переваги так і певні обмеження. До обмежень можна віднести потребу спеціального компілятору бути присутнім, він мусить створювати MSIL-файл, що використовується для роботи під керуванням CLR-системи, а також, цей компілятор мусить застосовувати бібліотеки середовища .NET Framework. До переваг керованого коду можна віднести - сучасні підходи до керування пам'яттю, наявність можливості використання різних мов, вищий рівень безпеки, можливість підтримки управлінням версіями ти детальна організація між взаємодіями програмних елементів. Будь-які Windows програми до появи середовища .NET Framework схилялися до використання некерованого коду, що не запускається CLR-системою. Керований і некерований код мають можливість виконуватись разом, тому факт появи C# компілятором керованого коду аж ніяк не обмежує його можливість виконуватися спільно з попередньо написаними програмами. Не дивлячись на те, що керований код володіє перевагами, наданими CLR-системою, в випадку, якщо він знаходиться в використанні іншими програмами, створеними на інших мовах, то для отримання максимальної зручності і легкості використання він мусить слідувати специфікації

універсальної мови (Common Language Specification – CLS). Цей термін описує набір параметрів, які паралельно мусять керувати різними мовами. При створенні програмних особливо важливо дотримуватись відповідності компонентів CLS-специфікації, що призначені для надання можливості використання системами, створеними на інших мовах. CLS-специфікація вміщає підмножину систем підтримки глобальних типів (Common Type System - CTS).

Microsoft Visual Studio [17] – одна з розробок корпорації Майкрософт, що являє собою інтегроване середовище розробки програмного та системного забезпечення і низку інших засобів інструментів. Ця розробка надає можливість створювати та програмувати як консольні додатки, так і додатки з користувацькими інтерфейсами, включно з підтримкою десктопної технології Windows Forms. Також присутня можливість програмувати веб-служби як в не керованому, так і керованому кодах, , веб-додатки, веб-сайти, для всіх платформ, що підтримуються .NET Framework, Windows Mobile, Windows CE, Microsoft Windows, .NET Compact Framework і Microsoft Silverlight. В Visual Studio присутні редактор вихідного коду з можливістю підтримки технології IntelliSense і можливістю дуже простої синтаксичної перевірки коду. Visual Studio пропонує вбудований налагоджувач. Він може працювати як налагоджувач рівнів вхідного коду, так і як налагоджувач на рівні машинного коду. Багато інших вмонтованих інструментів містять в собі редактор форм, який спрощує написання графічних інтерфейсів для програм, дизайнер класів і дизайнер схеми бази даних, веб-редактор. Visual Studio надає можливість програмувати ти під’єднувати сторонні програми та бібліотеки (плагіни) для розширення функціональності фактично на будь-якому рівні програми, разом з додаванням підтримки систем контролю версій написаного коду, включення нових наборів елементів та інструментів, наприклад, для зміни візуального проектування коду програми на предметно-орієнтованих інструментах програмування або інших аспектах циклу створення програмного забезпечення. Основний вигляд вікна, що відображається при запуску Visual Studio, старт роботи та створення нового проекту для майбутньої програми.

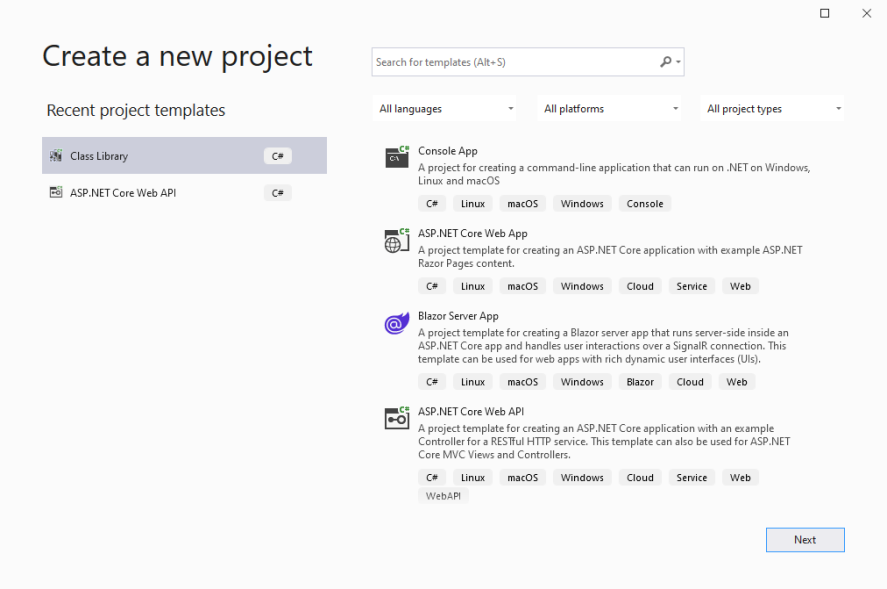

Рисунок 1.7 Створення нового проекту у Visual Studio

.NET Core [18] – це модульна платформа, яку створила компанія Microsoft. Вона призначена для розробки програмного забезпечення із відкритим кодом.

ASP.NET Core – це фреймворк від компанії Microsoft, який використовує платформу .NET Core. Він призначений для створення різних сучасних веб застосунків: від невеликих блогів до великих інтернет-магазинів.

Застосунки ASP.NET Core працюють на Windows, Mac і Linux. Він був спроектований для забезпечення оптимізованого середовища розробки для застосунків, які розгортаються в хмарі або запускаються локально. Основні переваги ASP.NET Core:

- відкритий висхідний код;
- підтримка національних алфавітів за допомогою використання Unicode висока продуктивність завдяки відкомпільованому коду;
- можливість запуску веб-застосунків на всіх пристроях (смартфони, планшети, ноутбуки, ПК);
- кросплатформенність – можливість роботи веб-застосунку на Windows, Linux и macOS;
- інтеграція з сервісами Microsoft;
- функціональність Razor Pages;

- можливість застосування типових шаблонів авторизації та автентифікації користувача;
- підтримка різноманітних сервісів, які дозволяють реалізувати типову логіку Клієнт-серверну архітектуру забезпечує використанням патерну MVC (Model-View-Controller).

Під час розробки веб-застосунку для взаємодії з базою даних була використана технологія Entity Framework [19]. Перевагою цієї технології є те, 19 що розробник працює з базою даних, використовуючи поняття сутності, об'єкта та методу замість таблиці та запитів. Щоб застосовувати Entity Framework з MS SQL потрібно встановити бібліотеку Microsoft Entity Framework Core SQLServer. Технологія Entity Framework Core – це обгортка реляційної бази даних, яка дозволяє представити базу даних в об'єктно-орієнтованій парадигмі. Цю технологію створила компанія Microsoft для розробників застосунків, які запускаються на платформі .NET Core. Завдяки технології Entity Framework Core застосунки, написані з використанням платформи .NET Core, дозволяють зберігати дані, використовуючи реляційну модель побудови даних. Реалізовуючи технологію Entity Framework Core Microsoft полегшила завдання розробникам по роботі з базою даних. Читання, зберігання, зміна, видалення та запит даних відбувається за допомогою Entity Framework. Базу даних, яку використовує Entity Framework, представляє собою реляційну модель. Презентація даних у реляційній моделі являє собою відображення таблиць з рядками. Робота з даними реалізовується через технологію запитів, які у свою чергу реалізовуються за допомогою діалектів мови SQL. Отже, технологія Entity Framework реалізовє підхід перетворення абстрактних об'єктів у форму зберігання даних, прийнятій в реляційній моделі, а саме: рядки, таблиці та запити до них. Одним із розширень технології Entity Framework є технологія Entity Framework Core, яка є оновленою технологією Entity Framework. Сумісність технології Entiti Framework Core та серверів баз даних покладається на версію серверу, на якому здійснюється робота з базою даних. Реалізація підходу завдяки якому Entity Framework Core отримує дані з бази даних являє собою реалізацію SQL-команд, якіс створюють запити до

серверу бази даних та отримають дані, що представляють собою об'єкти .NET.

Переваги технології Entity Framework Core:

- незалежність технологія Entity Framework Core та реалізації .NET;
- відкритий вихідний код;
- створення бази даних реалізовується написанням об'єктно-орієнтованого коду, тобто на основі абстракції класу технології Entity Framework Core самостійно створює реляційну базу даних
- технологія Entity Framework Core реалізована таким чином, що взаємодія з базою даних не залежить від реляційної СУБД;
- механізм реалізації SQL-запитів до бази даних у технології Entity Framework Core реалізований через вбудовану мову LINQ;
- підтримка запитів з параметрами та роботи з процедурами;
- реалізація CRUD-команд: create, read, update, delete;
- реалізація підтримки роботи не тільки з типами даних, які є вбудованими, але і підтримка реалізації з типами даних, які можна запрограмувати.

РОЗДІЛ ІІ. РОЗРОБКА ПОШУКОВОЇ СИСТЕМИ ДИСТАНЦІЙНИХ КУРСІВ

2.1 Добір архітектури для розробки програмного інтерфейсу

Розроблена система має мати клієнт-серверний тип архітектури. Це між-мережева або архітектура обчислень, в якій, задачі розділені між замовниками, що називаються клієнтами і постачальником послуг (функцій), що називається сервером. В цілому, сервер та клієнт являють собою вигляд програмного забезпечення. Завдяки обчислювальній мережі ці програми розташовані на різних обчислювальних машинах і можуть взаємодіяти між собою з використанням різних мережевих протоколів, також можуть бути розміщені і на одній машині.

Архітектура клієнт-сервер [20] – це модель яка займається обрахунками, в якій віддалений сервер розміщує, керує та доставляє більшість послуг і ресурсів, які споживаються клієнтом. Цей варіант архітектури включає один або більше комп'ютерів клієнтів, під'єднаних до центрального сервера через Інтернет або іншу мережу підключення.

Варіант архітектури клієнт-сервер також відомий як мережева обрахункова модель або мережа клієнт-сервер, через те, що всі запити та послуги транслюються через мережу.

Архітектура клієнт-сервер – це модель архітектури виробників та споживачів, в якій сервер грає роль виробника, а клієнт споживача (рис. 2.1). Сервер тримає в собі та надає споживачам послуги найвищого класу за вимогами. Такі послуги можуть містити доступи до інформації, спільний доступ до збережених файлів, доступ до принтера та, прямий доступ до необробленої обрахункової потужності сервера.

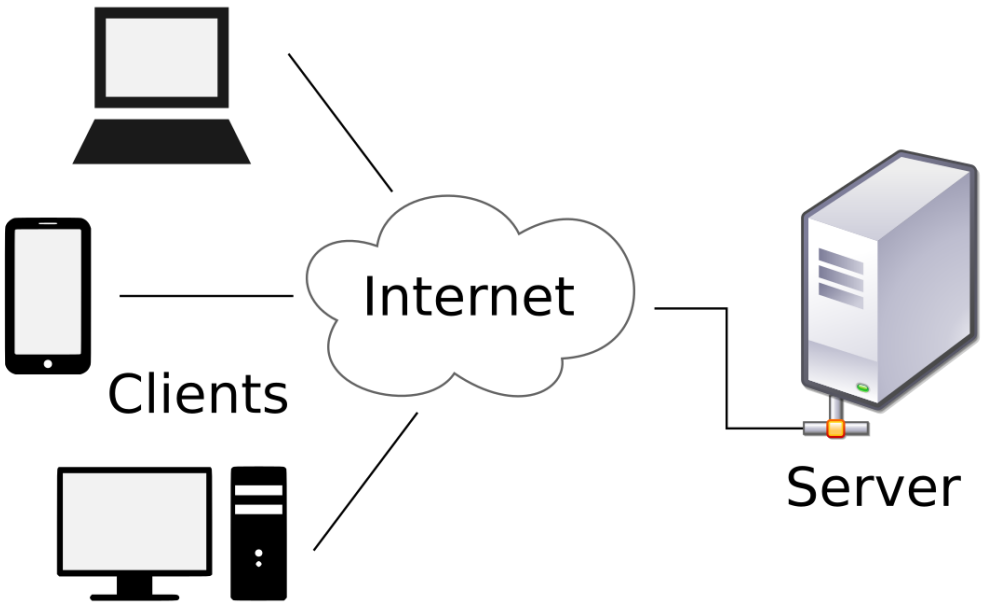

Рисунок 2.1 Архітектура клієнт-сервер

Клієнт-серверна архітектура працює, тоді, коли комп'ютер клієнта надсилає запит процесу на сервер через мережеве з'єднання або ресурсу, яке потім обробиться та доставиться на сторону клієнту. Комп'ютер серверу може керувати одночасно декількома клієнтами, в той же час як один клієнт може відкривати підключення до багатьох серверів одночасно, кожен з серверів може надавати різний набір послуг. Інтернет у найпростішій формі також використовує клієнт-серверну архітектуру, де обслуговуються веб-сервери багатьох різних користувачів з даними веб-сайту одночасно.

Модель клієнт-сервер направлена на підвищення впливу на вдосконалення онлайн індустрії, це породило нові вимоги клієнт-серверних додатків. Клієнт-серверні додатки відіграють значну роль у спілкуванні користувачів з онлайновими бізнес-організаціями, розповсюджені через Інтернет. Клієнт-серверна архітектура тут набуває незмінно важливого значення.

Клієнт-серверна архітектура – це суміжна система архітектур, в якій завантажується клієнт-сервер. Архітектура клієнт-сервер – це система ресурсів, яка є централізованою, в якій сервер містить в собі всі ресурси. Сервер очікує та обробляє численні надходження на своїй стороні для спільної

обробки ресурсів своїм клієнтам на відводі для запитів. Сервер і клієнт можуть бути на одній системі або в мережі. Сервер має бути одночасно масштабованим та стабільним, щоб могти клієнтам повертати відповіді. Дана архітектура спрямована на послуги, що означає, що повернення клієнтам відповідей не буде перервано. Архітектура клієнт-сервер залежить від мережевого трафіку, відповідаючи на запити клієнтів, а не на повні маніпуляції з файлами. Файлами займається файловий сервер під'єднаний до різних баз даних.

Клієнтські комп'ютери тримають зв'язок, щоб дати можливість користувачеві персонального комп'ютера надсилати запити по послуги сервера і очікувати результати, які повертає сервер не затримуючи інтерфейс. Сервери очікують на надходження запитів від клієнтів і повертають результат для них. Сервер, зазвичай, дає клієнтам стандартно сформований, простий тип інтерфейсу, задля уникнення плутанини програмного та апаратного забезпечення. Клієнти розміщаються на особистих машинах або на робочих місцях, в той же час, як сервери можуть будуть розташовані десь на потужних серверах в мережі. Дана архітектура особливо корисна, в основному, коли сервер та клієнти отримують окремі завдання, для виконання. Велика кількість клієнтів мають можливість одночасно очікувати та отримувати інформацію про сервери, в той же час клієнтські комп'ютери можуть обробляти інші завдання, наприклад, аналізувати веб сторінки.

Для імплементації серверної частини було використано API, який є інтерфейсом прикладного програмування. Прикладний програмний інтерфейс API – це набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено – це набір конкретних методів для співпраці різних елементів. Прикладний програмний інтерфейс API дозволяє розробнику використовувати засоби для ефективної розробки прикладного програмного забезпечення. Програмний веб-інтерфейс API може бути використаний для опису веб-базованих систем, апаратного забезпечення, баз даних, операційних систем, програмних бібліотек. Одним з найчастіших

призначень API є наявність набору широко використовуваних методів, як варіант - для малювання іконок на екрані чи вікна. Програмісти застосовують переваги API у гнучкій функціональності, таким чином вони не зобов’язані програмувати все з нуля. Прикладний програмний інтерфейс API є відносним, абстрактним поняттям – програмне забезпечення, яка пропонується певним API, зазвичай називають імплементацією даного API. В більшості випадків API це частина із наборів розробки прикладного програмного забезпечення, в той же час, набір розробки може містити як API, так і інші засоби або варіант апаратного забезпечення, тому ці два поняття не є взаємозамінюваними. Під час прямого використання прикладного програмного інтерфейсу в процесі веб розробки, зазвичай, API характеризується набором повідомлень - запитів HTTP, також описується структура повідомлень-відповідей, як варіант, у розширенні мови розмітки XML або в вигляді об'єктного запису JavaScript (JSON). В той же час, як програмний веб-інтерфейс у Web з давна був фактично синонімом для веб-служби, в сучасному світі тенденція змінилась (так званий Web 2.0) на відхід від Simple Object Access Protocol (SOAP) на базі веб-сервісів і сервісно-орієнтованої архітектури (SOA) на більш прості передачі відображеного внутрішнього стану (REST) стилів веб-ресурсів та ресурсів напрямленої архітектури (ROA). Напрямок цієї тенденції пов'язаний з шляхом семантичного веб-ресурсу для опису платформ (RDF) та концепцій розвитку сучасних веб-технологій - інженерних онтологій. Прикладні програмні вею-інтерфейси, що дають можливість об’єднувати декількома прикладними програмними веб-інтерфейсами для нових веб додатків називаються гібридними.

В роботі з серверною частиною використовується підхід «Onion Architecture» [21]. Більша частина традиційних архітектур розглядають основні питання грубого зв'язку та ділення проблем. Архітектура Onion (Clean Architecture) булапрезентована Джеффрі Палермо, щоб організувати кращий спосіб створення додатків задля перспективи зручнішого тестування, підтримуваності та стабільності. Clean Architecture виправляє проблеми, з

якими зіштовхуються архітектури побудовані з трьох рівнів, та n-ярусні архітектури, окрім того, забезпечують виправлення загальних проблем.

Схеми чистої архітектури працюють між собою за допомогою інтерфейсів. Clean Architecture – берез за основу принцип інверсії залежностей та управління. Чиста архітектура формується з декількох конкретних шарів, що працюють один з одним відносно до ядра, що презентує домен. Архітектура не прив’язана до рівня даних, як у стандартних багаторівневих архітектурах, а від явних доменів моделей. Інтерфейс UI взаємодіє з логікою обрахунків, слідуючи традиційній архітектурі, а бізнес-логіка - до рівню даних, всі шари в архітектурі змішуються і не схематично залежать один від одного. У трьох ярусної та n-ярусної архітектури ні один з рівнів не є незалежними; слідуючи цьому факту виникає відокремлення проблем. Ці системи дуже не легко зрозуміти і підтримувати в працездатному стані. Недоліком цієї не нової багаторівневої архітектури є непотрібне зв’язування. Clean Architecture - це принцип ділення програми на рівні. Разом з тим, є один рівень, що не залежить від інших, він знаходиться в центрі архітектури. Від нього залежить наступний рівень, від наступного - третій і так далі. Отже виходить, що над першим незалежним рівнем нашаровується наступний - залежний. Над другим розташовується третій, що також може бути відхідним і від першого. Наглядно це може бути відображено в вигляді цибулини, в якій також є середина (серцевина), навколо якої розміщаються всі інші верстви, аж до лушпиння.

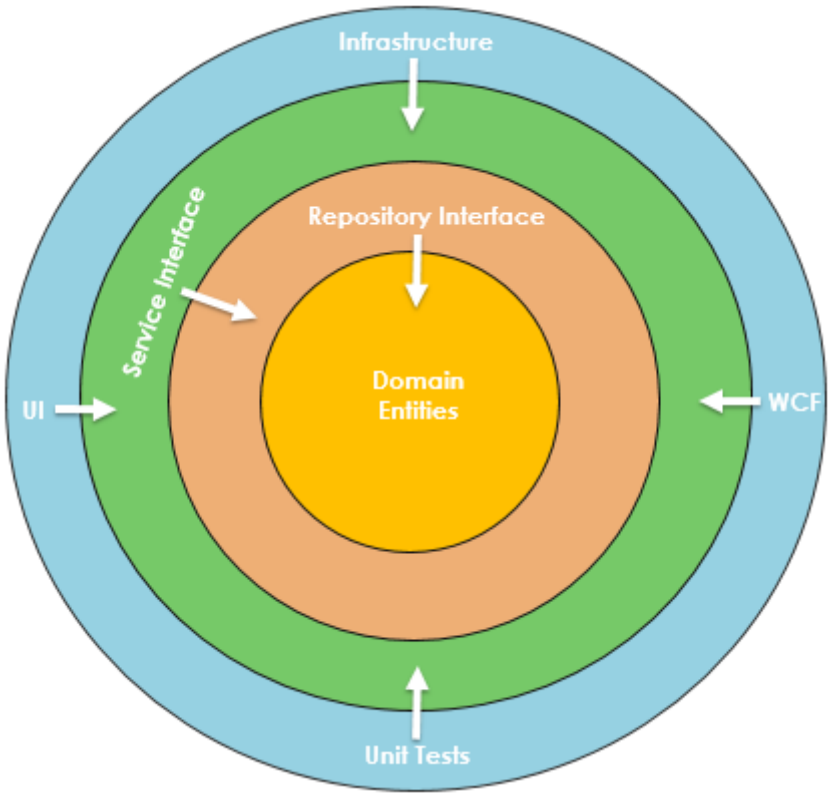

Рисунок 2.2 Onion-архітектура

Clean Architecture – займається вирішенням цих проблеми шляхом описування шарів від ядра до інфраструктури. Він використовує основне правило, направляючи всі зв'язки до центру. Такий підхід, безсумнівно, зміщений до об'єктно-орієнтованого типу програмування, і він презентує об'єкти перед всіма іншими. В центрі чистої архітектури розміщується доменна модель, яка репрезентує собою логіку обчислень і об'єкти поведінки. Шару домену оточують інші шари з значно більшою кількістю поведінки.

Традиційні архітектури використовують концепцію шарів, але вони не походять від 3-х рівневих і n-ярусних типів архітектур. В центрі схеми архітектури Onion знаходиться рівень домену; цей рівень репрезентує об'єкти логіки обчислень та поведінки. Основна суть полягає в тому, що всі об'єкти наявного домену містились в цьому ядрі. Воно розміщує всі об'єкти домену застосунків. Окрім предметів рівню даних, також можливо розміщувати інтерфейси домену. Залежностей ці об'єкти домену містити не мають.

Інтерфейси домену також є еквівалентними, як вони і повинні бути, без будь-якого громіздкого коду або залежностей.

Рівень сховища додає абстракцію між одиницями домену та логікою обчислень програми. В цей рівень зазвичай додаються об’єкти, що забезпечують зберігання та обробку отриманих об'єктів, зазвичай з використанням бази даних. Цей рівень формується з шаблону доступу до даних, що є більш незалежно підключеним варіантом до доступу до даних. Також створюється глобальний репозиторій і додаються запити на отримання даних з джерел надходжень, зіставляються дані з джерел даних до одиниць господарювання і збереження змін в обрахункових об'єктах до джерела даних.

Шар служби розміщує інтерфейси з звичайними операціями, такими як збереження, редагування, додавання, та видалення. Окрім того, цей рівень застосовується для зв'язку між рівнем інтерфейсу та рівнем сховища. Шар роботи з обслуговуванням також може розміщувати бізнес логіку обрахунків для сутності. В цьому рівні об’єкти обслуговування розміщені окремо від їх реалізації, підтримуючи вільний зв'язок і поділ проблем.

Рівень UI – являється верхнім шаром і зберігає зовнішні модулі прикладного веб-інтерфейсу, ними являються інтерфейс та тести. Для веб-додатку він представляє проект веб-API або тестовий проект. Цей рівень містить реалізацію процедури впровадження залежностей, ця процедура робить можливими для додатку будування вільно зв'язаної структури та посилання до внутрішніх шарів через інтерфейси.

Керівництво з Clean Architecture не надає жодних документацій щодо того, як мають бути побудовані шари. Програміст повинен вирішити імплементацію і вільно обрати будь-який рівень об’єкту, класу, інтерфейсу або будь-чого іншого, що є необхідним для розширення рішення. Нижче описано переваги імплементації чистої архітектури:

- Рівні структури додатку з'єднані через інтерфейси.
- Впровадження виконується під час роботи.
- Структура проектів побудована від проекту домену.

- Всі звертання до неконтрольованих ресурсів, наприклад, звертання до бази даних і активації сервісів, розміщена у зовнішніх шарах.
- Напрямок залежності від домену тільки вгору.
- Зворотна залежність до центру архітектури.
- Ефективна і стабільна універсальна архітектура.
- Відсутня потреба спільні проекти.
- Можливість гнучкої перевірки, оскільки серце програми не залежить від будь-якого рівня.

Кілька недоліків чистої архітектури:

- Високий рівень входження, не для початківців.
- Розробники неправильно формують обов'язки між шарами.

Висока залежність від інтерфейси. Чиста архітектура широко часто застосовується в промисловості. Це дуже ефективний і щільно пов'язаний з двома іншими стилями архітектури – Layered і Hexagonal. Чиста архітектура більш підходить для C# розробників, на відміну від програмістів Java.

2.2 Розробка сервісу із реалізацією пошуку

Пошукова система курсів – це платформа, звернувшись до якої користувач може відшукати потрібні йому курси по введеному ключовому слову або сполученню. На теперішній час пошукові системи курсів це один з найкращих інструментів для саморозвитку та отримання нових знань.

Щоб створити проект юзер інтерфейсу було використано command line interface npx create-react-app. Create React App – автоматичний інструмент для легкого та без проблемного створення React застосунку. Даний інструмент повертає як результат проект з вже попередньо налаштованими Webpack, Babel та іншими інструментами ручної розробки. Синтаксис команди для створення проекту: npx create-react-app app-name.

По створенню проекту потрібно налаштувати маршрутизатор. React пропонує власну систему маршрутизації, що дає можливість зіставляти запити в застосунок з певними елементами відтворення. Основним модулем в роботі

маршрутизації є модуль react-router, який містить головний функціонал по роботі з маршрутами. Проте для роботи у веб-браузері, необхідний також модуль react-router-dom. Скрипт для встановлення пакету: npm і react-router-dom. React router надає три ключові компоненти для налаштування роботи маршрутизатора в веб-додатку: BrowserRouter, Route, Link, Routes. Спочатку важливо обгорнути головний компонент додатку в файлі App.jsx в BrowserRouter, щоб надати доступ вкладеним компонентам до об’єкту History, що дає можливість відстежувати адресу сторінки. Елемент Route очікує до прийому два параметри: path і component. В цілому це визначає який елемент повинен бути відображеним при переміщенню по вказаному маршруті в веб-браузері. Компонент Routes надає можливість обрати один елемент для відображення. Без нього Router може відображати кілька елементів по одному шляху, якщо вони належать вказаному запиту. Для переходів по сторінках додатку був використаний компонент Link. При умові, що посилання створюються за допомогою звичайного тегу посилання (тегу a), кліки по них призвело б до оновлення сторінки. Елемент Link допомагає цьому запобігти. При кліку на елемент Link URL адреса оновиться, а інший вміст сторінки буде повторно відображено без оновлення сторінки. Через те, що було вирішено застосовувати функціональні компоненти, для наступної роботи з маршрутами будуть використані функції-хуки React Router 6. Всі вони підключаються з пакету react-router-dom. Хуки – це об’єкти-функції, які дають можливість використовувати стан і методи життєвого циклу класових компонентів в функціональних компонентах.

По завершенню налаштування маршрутизатору, прийшов час додати до застосунку сховище стану. Потрібно завантажити бібліотеку Redux і допоміжні модулі:

- Redux – основна модуль для роботи з стейтом;
- React Redux – бібліотека для роботи Redux з React;

Скрипт для встановлення: npm і redux react-redux. В Redux глобальний стан застосунку представлений у вигляді одного об’єкту JavaScript – state

(стан) або state tree (дерево станів). Статичне дерево станів використовується тільки для читання. Змінювати стейт можливо тільки при виклику action (дії). Дія (action) – це JavaScript-функція, яка коротко відображує суть зміни. Головною вимогою до цієї функції є наявність властивості type, яка містить значення рядок-назву дії. Генератори дій (actions creators) – це методи, які створюють дії. Редюсер (reducer) – це чистий метод, який перевизначає наступний стан стейту з використанням його попереднього стану і застосованою дією. Редюсер повертає новий об’єкт стану, який записується в глобальний стан замість попереднього. Сховище (store) – це об’єкт, в форматі JSON, який тримає в собі стан застосунку. Доступитися до стану можна через виклик функції getState(). Єдиним і основним правильним способом оновити стан, є виклик методу dispatch. Сховище дає можливість підписуватись і відстежувати змінами в стані з використанням функції subscribe.

Розглянемо спосіб роботи системи, який є досить простим. Користувачу, який зайшов на сайт платформи потрібно ввести у форму входу дані для входу.

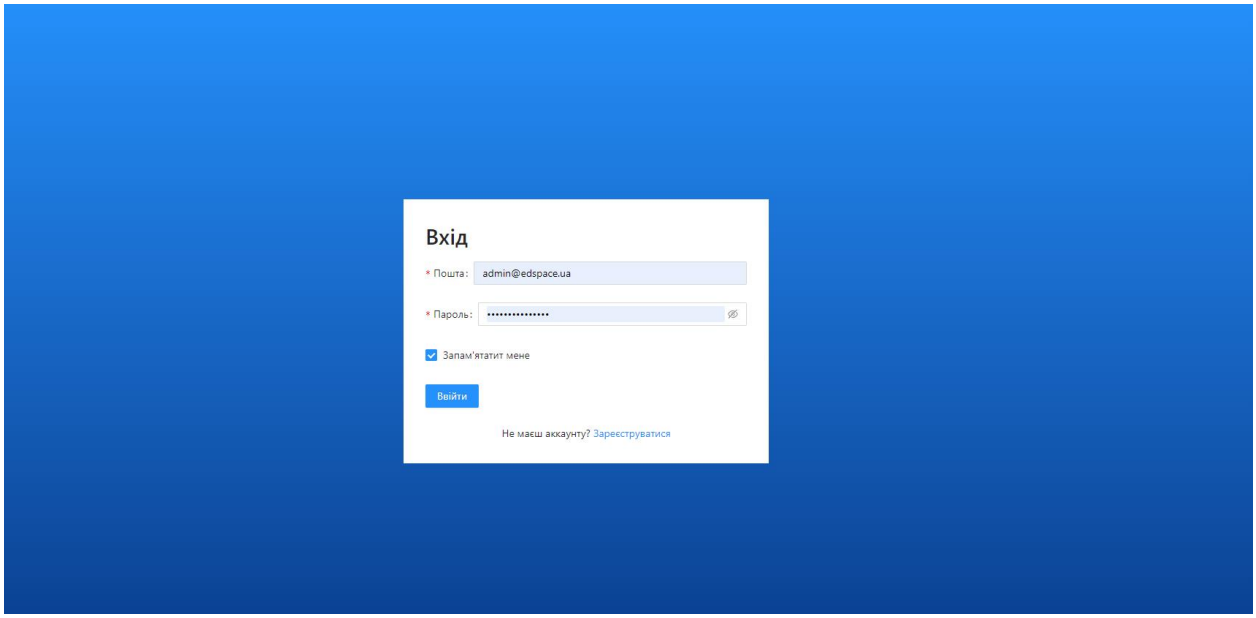

Рисунок 2.3 Форма входу

Якщо користувач вперше відвідує сайт, йому буде необхідно створити новий акаунт, натиснувши на посилання «Зареєструватися».

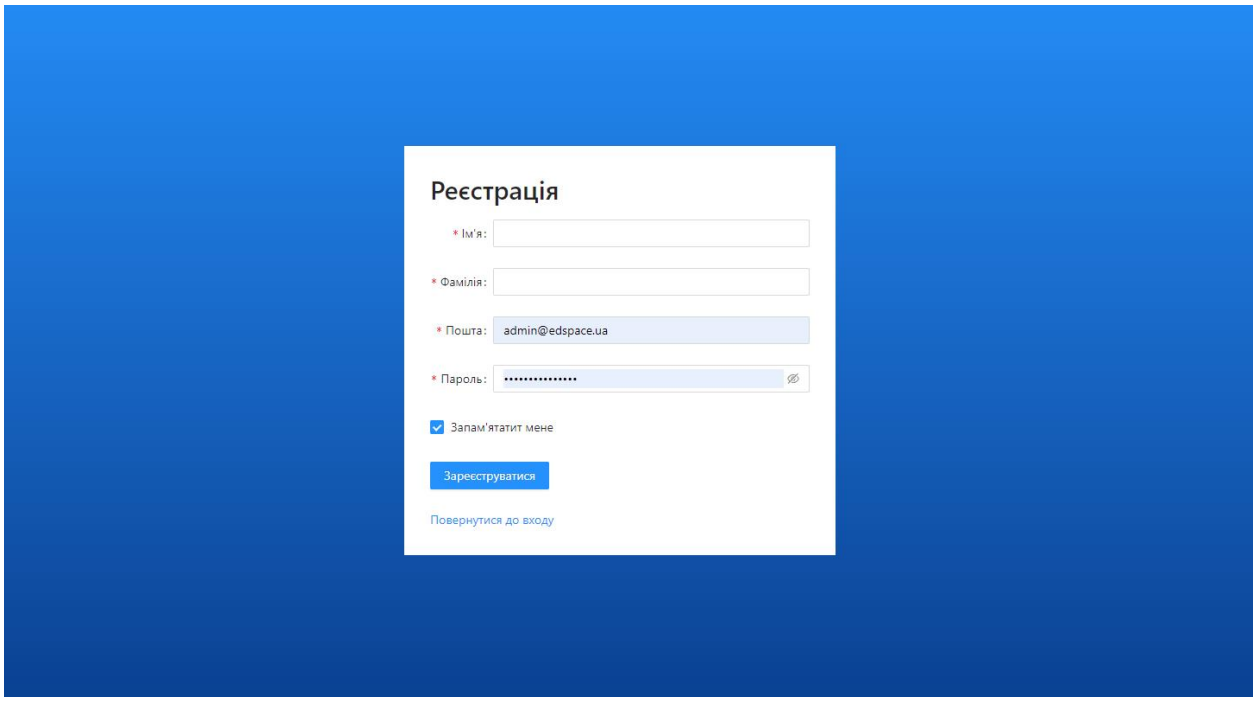

Рисунок 2.4 Створення нового акаунту

Після успішного входу або завершення реєстрації в систему буде відображено основну сторінку сайту.

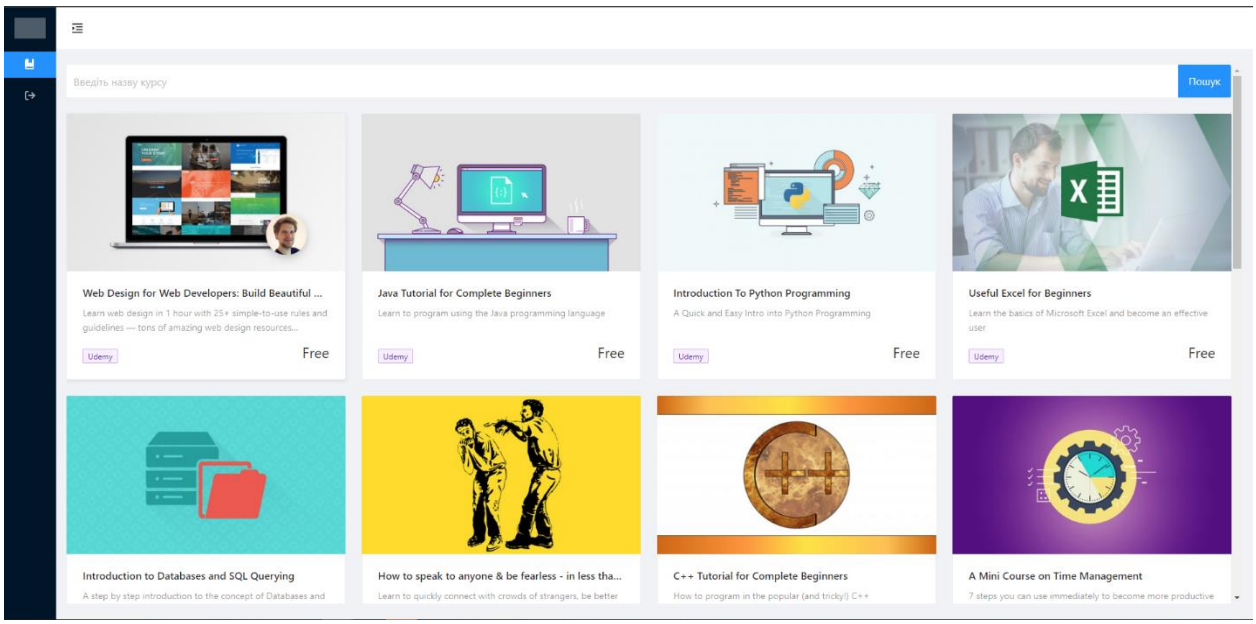

Рисунок 2.5 Основна сторінка сайту

Фреймворк Antd було обрано для стилізації через його бібліотеку інтерфейсів та вбудовану підтримку розмірів мобільного телефону. Для того, щоб робити як тестові так і робочі версії веб застосунку.

Antd – це бібліотека необхідних інструментів, розроблена для створення веб-застосунків, що містить в собі заготовки CSS та HTML для стилізації різноманітних компонентів взаємодії та інших компонентів інтерфейсу, а також додатковий функціонал для розширення і полешгення роботи з JavaScript. Він робить розробку сучасних веб-сайтів і веб-додатків значно простішою.

Для написання серверу використали REST API, яке являє собою інтерфейс прикладного програмування. Прикладний програмний інтерфейс API – це сукупність визначених підпрограм, різних протоколів взаємодії та інструментів для написання програмного забезпечення. Спрощено – це вибірка конкретно визначених методів для співпраці різних компонентів. Прикладний програмний інтерфейс API робить доступними розробнику засоби для ефективної розробки різного виду програмних забезпечень.

Програмний інтерфейс API використовується для десктопних та інтернет систем, систем програмного забезпечення, баз даних, апаратного забезпечення, програмних фреймворків. Надання набору широко використовуваних функцій є одним з найпоширеніших призначень API, наприклад для відображення вікна або іконок на екрані. Програмісти навчились використовувати функціональні переваги API, таким чином відпадає необхідність розробляти все з нуля.

Прикладний програмний інтерфейс API це також є абстрактним поняттям – реалізацію програмного забезпечення, що пропонують деякі API, як правило, називають - реалізацією даного API. Набір розробки застосунку може містити як API, так і інші варіанти веб забезпечення. Найчастіше API є інструментом набору розробки прикладного програмного забезпечення.

Використовуючи прикладний програмний інтерфейс в процесі веб розробки, зазвичай, API визначається наборами повідомлень запитів HTTP,

також обирається структура майбутніх повідомлень та відповідей, як правило, у форматі мови розмітки XML або в вигляді об'єктного запису JavaScript (JSON). Того ж часу, як програмний прикладний інтерфейс у Web, практично історично виступав синонімом до веб-служби. Сьогодні тенденція змінилась. Програмісти почали відходити від Simple Object Access Protocol (SOAP), що реалізований на основі веб-сервісів і сервіс-орієнтованої архітектури (SOA), обираючи більш безперешкодні передачі стану (REST) виглядів веб-ресурсів та ресурсів орієнтованої архітектури (ROA). Шматок цієї тенденції пов'язаний з стрімким рухом семантичного веб-ресурсу до Опису Платформ (RDF).

Шляхи розвитку інтернет-технологій інженерних напрямків. Програмні прикладні інтерфейси в інтернеті, що дозволяють поєднувати декілька прикладних програмних інтерфейсів в сучасні застосунки називають гібридними.

Для розробки серверної частини додатку було використано REST стиль. Зазвичай REST-стиль найбільш підходить для створення будь-якого роду Single Page Application, які часто використовують окремі спеціальні javascript фреймворки та бібліотеки наприклад: Angular, React або Knockout.

Фактично Web API являє собою веб-службу, яка може приймати запити выд інших додатків. До того ж, ці застосунки можуть репрезентувати будь-яку технологію і платформу – ними можуть виступати веб-застосунки, мобільні або десктопні клієнти.

Поняття Web API виведене на додаванні в додаток ASP.NET MVC Framework контролера конкретного виду. Цьому варіанту контролерів, що називається контролером API, притаманні такі характеристики:

- Методи дій повертають об'єкти класів репрезентації даних, а не класи типу ActionResult.
- Методи дій підставляються на основі HTTP-методу, що використовується в запиті.

Схематична структура додатку, реалізованого за допомогою технології ASP.NET Web API.

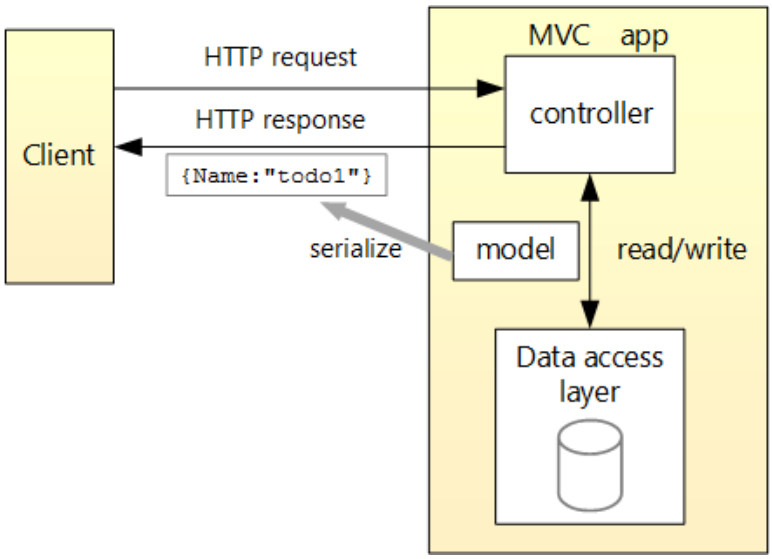

Рисунок 2.6 Структура додатку

Моделі даних, що повертаються як результат методу дії контролера API, приходять в форматі JSON і надсилаються клієнту. Контролери API слугують для доставки веб-службам даних, саме через це, вони не дотримуються уявлень, компонувань або любих інших засобів, які використовувалися для формування HTML-розмітки в прикладі написаного додатку. Контролер API не може генерувати HTML-розмітку з представлень, саме це є причиною, чому в додатках які працюють з однією сторінкою, поєднуються стандартні принципи ASP.NET MVC Framework з Web API. В схемі побудови ASP.NET MVC Framework виконуються кроки, необхідні для транспортування HTML-вмісту остаточному користувачу (вміщуючи авторизацію, аутентифікацію, візуалізацію та вибір уявлення). Після доставки HTML-вмісту в браузер, шляхом виконування запитів Ajax, які генеруються та містяться в середині стандартних бібліотек коду JavaScript, оброблятимуться контролером Web API.

Під розробку системи, після проведення аналізу наявних архітектур програмування, вирішили використати трирівневу архітектуру.

Presentation layer (верхній рівень, рівень презентації) – це частина системи, яка керує і надається споживачеві. На цьому рівні розміщуються

контролери які отримуватимуть запити від користувачів. Presentation layer реалізовано у проекті EDSpace.API.

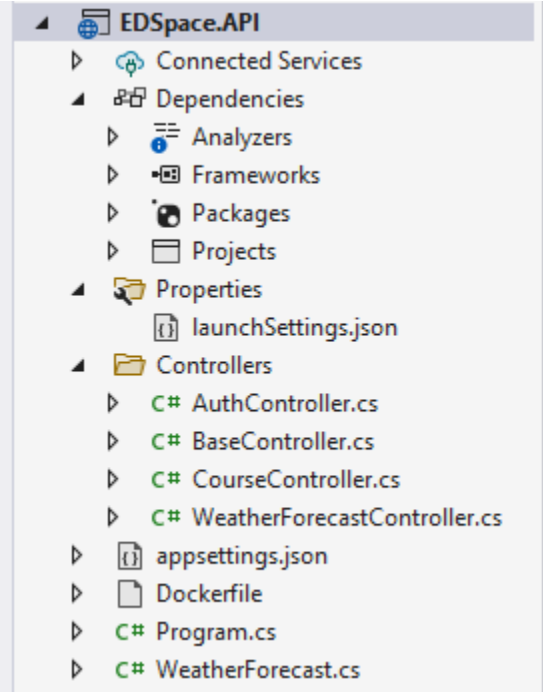

Рисунок 2.7 Presentation layer проєкту

Business Layer (рівень основної бізнес-логіки) – містить в собі перелік компонентів системи, що відповідають за оформлення отриманих від верхнього рівня (презентації) даних, місце для розміщення та реалізації всієї необхідної логіки додатку, усі обчислення, та додаткові класи хелпери. Через цей рівень ведеться взаємодія з базою даних, та передача на верхній рівень результатів обробки. Business Layer реалізовано у проекті EDSpace.BL.

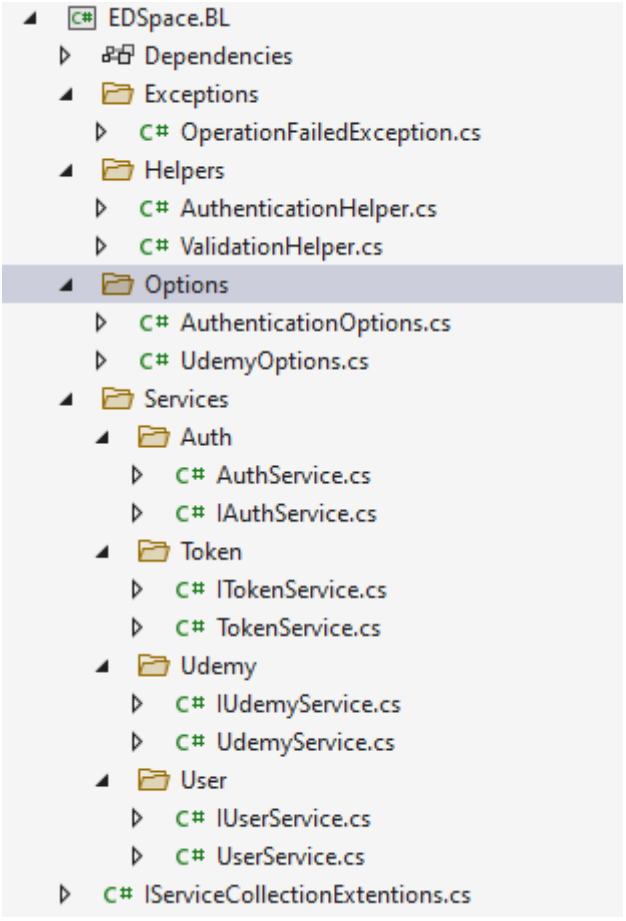

Рисунок 2.8 Business layer проєкту

Data Access Layer (місце збереження моделей бази даних) – займається зберіганням моделей, які описують сутності, також тут розміщуються особливі нестандартні класи для роботи з основною технологією доступу до даних, як варіант, клас контексту даних Entity Framework Core. Також, в цій бібліотеці знаходяться репозиторії, через які основна логіка взаємодіє з базою. Варто звернути увагу, що верхні рівні мусять не взаємодіяти між собою, що означає, що рівень презентації не має посилатись до бази даних без посереднього рівню. Така можливість може бути реалізована тільки шляхом використання основної бізнес логіки. Data Access Layer описано у проекті EDSpace.DAL.

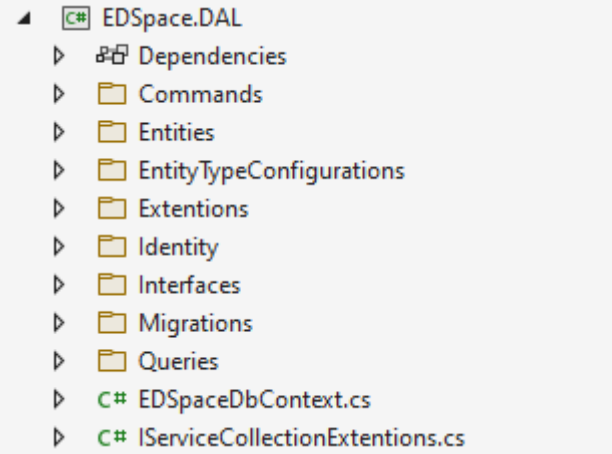

Рисунок 2.9 Data Access layer проєкту

Впровадження залежностей – функціонал, що дає можливість забезпечити слабку залежність компонентів системи. Сукупності програми зв’язані між собою шляхом абстракції, як варіант, через інтерфейси. Паралельно, відповідальність за ініціалізацію відповідних залежностей переходить на зовнішній, конкретно призначений для цього глобальний механізм. Нерідко, роль таких механізмів займають IoC-контейнери. Ці контейнери відпрацьовують своєрідними фабриками, що встановлюють щалежності між абстракціями та конкретними імплементаціями і, звичайно, керують ініціалізацією цих об’єктів. Такий підхід дозволяє підвищити гнучкість системи, упростити підтримку, розвиток та заміну компонентів системи. У попередніх версіях ASP.Net, для використання механізму впровадження залежностей потрібно було використовувати зовнішні IoC-контейнери, такі як Ninject, Unity, Autofac. Та починаючи з версії ASP.Net 5 у програмі присутній вбудований IoC-контейнер який представлений інтерфейсом IServiceProvider, керування яким проводиться у класі Startup.

ВИСНОВКИ

Навчальні електронні курси стали невід'ємною частиною навчального процесу. Вони допомагають якісно проводити навчальну діяльність у важких умовах сьогодення, сприяють формуванню нових предметних вмінь та навичок. Віддалене навчання має ряд переваг перед навчанням у навчальних закладах, підвищує мобільність та самостійність освітньої діяльності студентів. Електронні курси є доступними у використанні, дозволяють обирати зручний час та місце навчання, стимулюють розумову діяльність та творчу активність студента.

Розробка веб-додатку для оптимізації пошуку потрібних користувачу навчальних курсів із представлених на існуючих освітніх платформах значно спрощує студенту процес добору курсу та збільшує його шанси на повне проходження усього навчального матеріалу.

Проаналізувавши можливості та переваги використання дистанційних курсів нами було зроблено висновок про те, сучасна молодь широко використовує можливості дистанційного навчання на сучасних освітніх платформах.

Розглядаючи існуючі дистанційні курси, платформи та технології іхньої реалізації, нами було обрано технології C#, JavaScript, ASP.NET 6, React для розробки веб-додатку з пошуку навчальних курсів.

У ході кваліфікаційного дослідження нами було сплановано архітектуру і дизайн веб-додатку, розроблено прикладний програмний інтерфейс для збору даних та інтерфейс веб-додатку. Готовий веб-додаток для пошуку навчальних курсів було розгорнуто на веб-сервері.

Розроблений прикладний програмний інтерфейс та односторінковий веб-застосунок пройшов апробацію та може бути використаний студентами в подальшому для оптимізації пошуку навчальних курсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крайчук О., Остапчук Н., Використання можливостей хмарних сервісів у процесі навчання студентів у закладах вищої освіти // Нова педагогічна думка. Науково – методичний журнал. – Рівне : РОІППО, 2019. – № 1 (93). – С. 45–48.
2. Остапчук В. О., Остапчук Н. О. Аналіз систем сучасних освітніх дистанційних курсів // Інформаційні технології в професійній діяльності : матеріали XIV Всеукраїнської науково-практичної конференції. – Рівне : РВВ РДГУ. 2021. – С. 50 – 52.
3. Петренко С.В. Сутність та особливості українських платформ масових відкритих онлайн-курсів (МВОК) / Інноватика у вихованні. Випуск 11. Том 2. 2020. – С. 165 – 173.
4. Петренко С.В. Організація дистанційного навчання в умовах пандемії COVID-19: аналіз міжнародного досвіду європейських країн / Інноватика у вихованні. Випуск 13.Том 2. 2021. –С.137 – 152.
5. Петренко С.В. Оптимізація й аналіз результатів використання LMS MOODLE у системі змішаного навчання в університеті/ Інформаційні технології і засоби навчання, 2017, Том 61, N5. –С.140 – 150.
6. Переваги дистанційного навчання. [Електронний ресурс] – Режим доступу до ресурсу: https://kerivnyk.info/perevahy-ta-nedoliky-dystantsijnoho-navchannya
7. Курсера. [Електронний ресурс] – Режим доступу до ресурсу: https://ru.coursera.org/
8. Юдемі. [Електронний ресурс] – Режим доступу до ресурсу: https://www.udemy.com/
9. Прометеус. [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.org.ua/

10. UX. [Електронний ресурс] – Режим доступу до ресурсу: https://te.itstep.org/blog/ui-and-ux-design
11. Virtual DOM. [Електронний ресурс] – Режим доступу до ресурсу: https://learn-reactjs.ru/faq/virtual-dom-and-internals
12. JavaScript. [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/webtehnologiietawebdizajn/mova-javascript-ta-ieie-mozlivosti
13. Реакт. [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/
14. Редакс. [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/
15. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с.
16. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с.
17. Visual Studio. [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/
18. .Net Core. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/fundamentals/
19. Entity Framework. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/
20. Архітектура клієнт-сервер. [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/495698/
21. Цибулева архітектура. [Електронний ресурс] – Режим доступу до ресурсу: https://www.codeguru.com/csharp/understanding-onion-architecture/