КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ПОБУДОВА ВЕБ-ЗАСТОСУНКУ ДЛЯ ВЗАЄМОДІЇ З ДИТЯЧИМИ ГУРТКАМИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ WEBAPI

Виконав студент 4-го курсу
Остапчук Єгор Володимирович

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Федорус Олексій Мстиславович

(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«____» _____ 202_ р.,
протокол № ___
Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 60 сторінок, 26 ілюстрації, 22 джерела посилання.

ГУРТКИ ДЛЯ ДІТЕЙ, ВЕБ, ВЕБ-ДОДАТОК, ВІЗУАЛІЗАЦІЯ, СИСТЕМАТИЗАЦІЯ, АВТОМАТИЗАЦІЯ, ASP.NET CORE WEB API, ANGULAR, TYPESCRIPT, SQL

Об’єктом роботи є проект по створенню сервісу для гуртків і батьків.

Метою роботи є розробка сервісу гуртків з використанням найсучасніших методів, аналіз проблем та знайдення ефективних рішень.

Для розробки даної програмної реалізації було обрано мови програмування C#ASP.NET Core, Entity Framework, Redis, Elastic Search, Docker, Angular, Typescript, HTML, CSS, GIT. Під час розробки у якості інтегрованого середовища було обрано вільно поширювані Visual Studio Code та Visual Studio Community Edition, MS SQL DB, Postman, Git Hub.

Результати роботи: проведено докладний аналіз проблем, що зустрілися під час розробки. Розроблено веб додаток, що дозволяє реєструватися надавачам послуг, створювати нові гуртки, набирати дітей, реєструватися батькам, додавати дітей, записувати дітей на навчання, спроектовано сервер, клієнт та БД для зберігання інформації, задеплоєно сервіс до хмарної платформи. Ця реалізація може бути корисна для надання інформації про гуртки та запису дітей в зручному онлайн режимі.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки.

Задача розробки сервісу для автоматизації та систематизації роботи гуртків стоїть надзвичайно гостро, адже подібних сервісів не існує на даний момент. В нашій країні мільйони дітей відвідують різні гуртки, іноді навіть не знаючи про альтернативу, тому вирішення цього питання допомогло б як гурткам, так і батькам з дітьми.

Актуальність теми.

На даний час немає сервісів, які б дозволили впорядкувати, систематизувати роботу дитячих гуртків. Також немає засобів для обліку існуючих, діючих гуртків, та автоматизації набору дітей.

Подібний сервіс дозволяє вирішити дану проблему, полегшити роботу для гуртків , допомогти батькам з легкістю записувати своїх дітей у місця, що їм до вподоби.

Мета й завдання роботи.

Метою дипломної роботи є розробка веб-додатку та проведення дослідження проблем, які можуть протягом цього часу. Для досягнення цієї мети поставлено такі завдання:

- Проектування БД, у якій буде зберігатися інформація про надавачів послуг, гуртки, дітей, батьків і тому подібне.
- Проектування архітектури серверу
- Розробка і документація серверу
- Тестування серверу
- Розробка архітектури клієнту, зручною для користувача
- Розробка клієнту
- Тестування клієнту
- Деплой сервісу в хмарне середовище

Об’єкт і методи дослідження або розробки.

Перед розробкою ПЗ був проведений аналіз ринку на предмет схожих рішень та актуальність даної теми. Виявилось, що тема я дуже актуальною у наш час і все ще набирає популярність. Після цього було спроектовано структуру додатку у відповідних сервісах для полегшення подальшої розробки.

Для розробки серверної частини було обрано C# ASP.NET Core 3. Мова є дуже ефективною в реалізації програм, особливо, використовуючи Restful архітектуру на основі шаблону WebAPI. Її переваги перед іншими мовами, які використовують для бекенд розробки: кроссплатформерність, легкість у написанні і підтримці сервісу, а також швидкодія.

Для веб частини було обрано стек CSS + HTML + Angular + Typescript. Це дуже популярний вибір у цій сфері, обумовлений простотою та зручністю розробки, детальною документацією, повним циклом розробки без використання додаткових засобів та швидкістю алгоритму пошуку змін у дереві DOM у Angular.

Під час розробки даного програмного засобу використовувалося два інтегрованих програмних середовища від Microsoft: Visual Studio (для серверної частини) та Visual Studio Code (для вебу). Для управління та створення бази даних було обрано Microsoft SQL Server Management Studio. Для цього ПЗ розробник надає безкоштовну ліцензію для студентів вищих навчальних закладів та розробки поза межами компаній.

Можливі сфери застосування. Розроблений додаток може бути застосований як для комерційного, так і некомерційного використання, або як соціальний проект.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ (ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ)

1.1 Сучасна клієнт-серверна архітектура, принцип роботи веб додатків

На даний час веб-додатки отримали великий розвиток в різних сферах діяльності суспільства. Робота веб-додатків проводиться за допомогою клієнт-серверной технології, де клієнтом є браузер, а в якості сервера виступає веб сервер. Для початку наведемо загальну схему роботи веб-додатків, яка описана нижче на рис. 1.

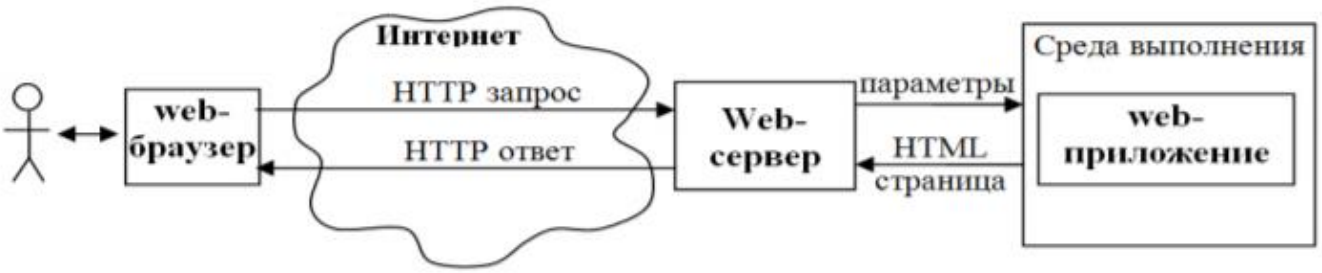


Рис 1. Принцип роботи веб-додатку

З рис. 1 видно, що клієнт, звертаючись до веб-браузеру, відправляє HTTP-запит за певною URL адресою, що вказує на деякий динамічний ресурс, а саме сам веб-додаток. Далі сервер формує на основі веб-додатки HTML-сторінку, яка за допомогою браузера відображається клієнту. З опису схеми можна зробити висновок, що основна робота веб-додатку здійснюється на стороні сервера.

На даний момент існує безліч технологій, що реалізують логіку веб додатків на стороні сервера. У даному розділі будуть розглянуті найпопулярніші з них.

Першою широко використаною технологією стала CGI (Common Gateway Interface), яка особливо застосовна для створення динамічних веб-сторінок і служить для забезпечення зв'язку між клієнтом (веб-браузером) та веб-сервером. Дана технологія являє собою набір правил, дотримуючись яких, програма здатна виконуватися на різних серверах операційних системах. Відповідно до технології CGI, HTTP запит, який містить посилання на динамічну сторінку, вступаючи на веб-сервер, генерує новий процес і запускає потрібну прикладну програму.

Наступною технологією, яка досить широко поширена, стала Java Servlets або просто сервлети. Дана технологія дозволяє вирішити проблему продуктивності шляхом виконання всіх запитів в одному процесі шляхом розподілу їх по потокам всередині процесів. Це означає, що програмний код сервлету повинен бути потокобезпечним. Також плюсом використання сервелетов є їх незалежність від платформи, тому що вони виконуються на віртуальній Java-машині. Java Servlets володіє широким функціоналом, який можна досягти завдяки великій кількості бібліотек.

Новітньою технологією розробки web-додатків є .NET технологій, розроблена компанією Microsoft. Платформа .NET значно спростила процес розробки додатків і підвищила надійність коду. Стали доступними функції автоматичного управління часом життя об'єктів, обробка виключень і їх налагодження, в наявності з'явилися бібліотеки, нейтральні до мов програмування. Набір стандартних базових класів забезпечують розробнику доступ до сервісів платформи при використанні будь-яких мов програмування, сумісних з .NET. Common Language Runtime спільно з базовими класами складають основу платформи .NET і пропонує розробникам високо рівневі сервіси, такі як ADO.NET (вдосконалений ADO, використовуваної

SOAP і XML з метою обміну даними), ASP.NET (нове покоління ASP, що дає можливість використовувати будь-яку мову програмування, сумісний з .NET) і Windows Forms і Web Forms (класи, що реалізують локальні і web-орієнтовані додатки). Компілювання вихідного коду відбувається за такою схемою: створюється код на проміжному мовою (Microsoft Intemediate Language), а далі компілюється в середовищі CLR (Common Language Runtime). На відміну від старої версії, де компілятор створював машинний код, даний вид компіляції дозволяє скомпілювати файлу виконуватися на платформі будь-якого процесу. Нові можливості ASP.NET відповідають сучасним вимогам. Ось лише деякі з них: великий набір бібліотек, кроссплатформерність, мовна незалежність платформи, нові шляхи обробки помилок і т. п.

Кожна з перерахованих вище платформ має можливості і обмеження в індивідуальному порядку, а також свою власну сферу застосування, що надає розробнику широкий вибір інструментів розробки.

1.2 Опис технології API

API. Розшифровується як прикладний програмний інтерфейс. API насправді є свого роду інтерфейсом, який має набір функцій. Цей набір функцій дозволить програмістам отримати деякі специфічні функції або дані програми.

Веб-API. Це API, як випливає з назви, доступ до нього можна отримати через Інтернет за допомогою протоколу HTTP. Це фреймворк, який допомагає вам створювати та розвивати RESTFUL-сервіси на основі HTTP. Веб-API можна розробити за допомогою різних технологій, таких як java, ASP.NET тощо. Веб-API використовується або на веб-сервері, або у веб-браузері. В основному Web API - це концепція веб-розробки. Він обмежується клієнтською стороною веб-програми, а також не містить

відомостей про веб-сервер або веб-браузер. Якщо програма має використовуватися в розподіленій системі та надавати послуги на різних пристроях, таких як ноутбуки, мобільні телефони тощо, використовуються служби веб-API. Web API – це розширена форма веб-додатка. [2][3][4]

ASP.NET Web API. ASP.NET означає Active Server Pages.NET. Здебільшого використовується для створення веб-сторінок і веб-технологій. Він вважається дуже важливим інструментом для розробників для створення динамічних веб-сторінок за допомогою таких мов, як C# і Visual Basic. ASP.NET Web API — це фреймворк, який допомагає створювати служби, спрощуючи доступ до широкого кола клієнтів, включаючи браузери, мобільні пристрої, планшети тощо. За допомогою ASP.NET ви можете використовувати ту саму структуру та шаблони для створення веб-сторінок і служб.

Використання. Веб-API дуже корисні для реалізації веб-сервісів RESTFUL за допомогою .NET Framework. Веб-API допомагає розробити HTTP-сервіси для зв'язку з клієнтськими об'єктами, такими як браузер, пристрої або планшети. ASP.NET Web API можна використовувати з MVC для будь-якого типу програми. Веб-API може допомогти вам розробити додаток ASP.NET через AJAX. Отже, веб-API полегшує розробникам створення програми ASP.NET, сумісної з будь-яким браузером і майже будь-яким пристроєм.[6]

Чому варто обрати Web API. Послуги Web API є кращими перед іншими службами для використання з рідною програмою, яка не підтримує SOAP, але вимагає веб-сервісів.

Для створення ресурсно-орієнтованих служб найкраще вибрати служби веб-API. Ці служби встановлюються за допомогою HTTP або служби спокою.

Якщо вам потрібна висока продуктивність і швидкий розвиток сервісів, служби веб-API дуже корисні.

Для розробки легких і обслуговуваних веб-сервісів служби веб-API дійсно корисні для розробки цієї служби. Він підтримує будь-який текстовий шаблон, наприклад JSON, XML тощо.

Пристрої, які мають обмежену пропускну здатність або мають обмеження пропускної здатності, послуги Web API є найкращими для цих пристроїв.

Як використовувати Web API. Веб-API отримує запити від різних типів клієнтських пристроїв, таких як мобільний телефон, ноутбук тощо, а потім надсилає ці запити на веб-сервер для обробки цих запитів і повертає клієнту бажаний результат. Веб-API — це взаємодія «Система-система», за якої до даних або інформації з однієї системи може отримати доступ інша система, після завершення виконання отримані дані або, можна сказати, як вихідні дані відображаються глядачу.

API надає дані своїм програмістам, які доступні для зовнішніх користувачів. Коли програмісти вирішують зробити деякі зі своїх даних загальнодоступними, вони «розкривають кінцеві точки», тобто публікують частину мови, яку використовували для створення своєї програми. Інші програмісти можуть потім витягувати дані з програми, створюючи URL-адреси або використовуючи HTTP-клієнти, щоб запитувати дані з цих кінцевих точок.

Сторона сервера: веб-API на стороні сервера – це програмний інтерфейс. Він складається з однієї або кількох загальнодоступних кінцевих точок. Він визначає систему повідомлень запит-відповідь. Mashup — це веб-додаток, який є серверним API, який поєднує кілька серверних API. Webhook — це API на стороні сервера, який приймає вхідні дані як єдиний ідентифікатор ресурсу.[5]

Клієнтська сторона: веб-API на стороні клієнта націлені на стандартизовані прив'язки JavaScript. Google створив рідну клієнтську архітектуру, призначену для заміни вбудованих плагінів безпечними вбудованими розширеннями та програмами в пісочниці.

Кроки використання веб-API. Для більшості API потрібен ключ API. Коли ви знайдете API, з яким хочете грати, перегляньте в документації вимоги до доступу. Більшість API запропонують вам пройти підтвердження особи, як-от увійти за допомогою облікового запису Google. Ви отримаєте унікальний рядок літер і цифр, які можна використовувати під час доступу до API.

Найпростіший спосіб розпочати використання API – це знайти в Інтернеті клієнт HTTP, наприклад REST-Client, Postman або Paw. Ці готові інструменти допомагають структурувати ваші запити на доступ до існуючих API за допомогою отриманого вами ключа API. Вам все одно знадобиться знати деякі синтаксиси з документації, але для цього потрібно дуже мало знань з кодування.

Наступний найкращий спосіб отримати дані з API — створити URL-адресу з наявної документації API.

Популярні приклади використання API:

- API Карт Google: API Карт Google дозволяє розробникам використовувати Карти Google на веб-сторінках за допомогою інтерфейсу JavaScript або Flash.
- API YouTube: API Google дозволяє розробникам інтегрувати YouTube і функціональні можливості у веб-сайти або програми. API YouTube включає API аналітики YouTube, API даних YouTube, API прямої трансляції YouTube, API програвача YouTube та інші.
- API Flickr: його використовують розробники для доступу до даних спільноти Flick для обміну фотографіями.

- API Twitter: Twitter пропонує два API: REST API дозволяє розробникам отримати доступ до основних даних Twitter, а API пошуку надає розробникам методи взаємодії з даними пошуку Twitter і тенденціями.

1.3 Опис архітектури REST

REST визначає організацію клієнт-сервер, в якій внутрішні дані надаються клієнтам через формат обміну повідомленнями JSON або XML. За словами Роя Філдінга, API кваліфікується як «RESTful», якщо відповідає наступним обмеженням:

- Єдиний інтерфейс: API повинен надавати споживачам API певні ресурси програми.
- Незалежність клієнт-сервера: клієнт та сервер працюють незалежно. Клієнт знатиме лише ті URI, які вказують на ресурси програми. Зазвичай вони публікуються у документації API.
- Без збереження стану: сервер не зберігає дані, які стосуються запиту клієнта. Клієнт зберігає ці "дані стани" на своєму кінці (через кеш).
- Кешуються: ресурси програми, що надаються API, повинні бути кешуються.
- Багаторівнева: архітектура є багаторівневою, що дозволяє підтримувати різноманітні компоненти на різних серверах.
- Код запиту (COD): це єдине необов'язкове обмеження REST. Це дозволяє клієнту отримувати виконуваний код як відповідь від сервера. Іншими словами, саме сервер визначає, як виконуватимуться конкретні дії.

Нарешті, архітектура REST API зазвичай спирається на протокол HTTP, а REST API є найпоширенішим форматом створення веб-додатків і

підключення мікро сервісів. Коли REST API стає загальнодоступним як веб-служба, кожен компонент (або послуга), що надається веб-службою, є клієнтом як ресурс. Клієнти можуть отримати доступ до цих ресурсів через загальний інтерфейс, який приймає різні HTTP-команди, такі як GET, POST, DELETE та PUT.

1.4 Опис технології ORM

Однією з проблем використання мов і баз даних об’єктно-орієнтованого програмування (ООП) є складність узгодження коду програмування зі структурами баз даних. **Об’єктно-реляційне відображення (ORM)** — це техніка, яка створює шар між мовою та базою даних, допомагаючи програмістам працювати з даними без парадигми ООП.

Проблема, що постала перед розробниками ООП, полягає в тому, щоб зрозуміти й кодувати мовою структурованих запитів (SQL), щоб підключити свою програму до бази даних SQL. Розробники, які знають SQL, можуть писати код доступу до даних. Це необроблене кодування SQL може зайняти надзвичайно багато часу, оскільки воно вимагає від розробника витягнути елементи даних рядків коду. Конструктори запитів SQL додають рівень абстракції до коду SQL, щоб надати більше інформації про дані. Однак розробникам все одно потрібно розуміти і писати SQL.[1]

ORM популярні та суперечливі водночас. Прихильники ORM стверджують, що вони підвищують продуктивність, покращують дизайн програми, повторно використовують код і підтримують програму з часом. На думку недоброзичливців, негативним аспектом ORM є продуктивність. Надаємо огляд ORM, порівняємо їх із інструментами SQL, а також розглянемо плюси та мінуси цих інструментів, щоб ви

могли вирішити, чи допоможуть ORM або зашкодять вашим зусиллям з розробки додатків баз даних.

Що таке ORM. Об’єктно-реляційний картограф забезпечує об’єктно-орієнтований рівень між реляційними базами даних і об’єктно-орієнтованими мовами програмування без необхідності писати SQL-запити. Він стандартизує інтерфейси, скорочуючи шаблон і прискорюючи час розробки.

Об’єктно-орієнтоване програмування включає багато станів і кодів у форматі, який є складним для розуміння та інтерпретації. ORM перекладають ці дані та створюють структуровану карту, щоб допомогти розробникам зрозуміти основну структуру бази даних. Відображення пояснює, як об’єкти пов’язані з різними таблицями. ORM використовують цю інформацію для перетворення даних між таблицями та створення коду SQL для реляційної бази даних, щоб вставляти, оновлювати, створювати та видаляти дані у відповідь на зміни, які програма вносить до об’єкта даних. Після написання відображення ORM керуватиме потребами програми в даних, і вам більше не потрібно буде писати низькорівневий код.

Як працює ORM. ORM створюють модель об'єктно-орієнтованої програми з високим рівнем абстракції. Іншими словами, це створює рівень логіки без основних деталей коду. Відображення описує зв’язок між об’єктом і даними, не знаючи, як дані структуровані. Потім модель можна використовувати для з’єднання програми з кодом SQL, необхідним для керування даними. Цей «сантехнічний» тип коду не потрібно переписувати, що економить розробнику величезну кількість часу.

Типи ORM. ORM використовують дві різні стратегії: шаблон активного запису і шаблон відображення даних.

Шаблон активного запису. Ця стратегія відображає дані в структурі об’єктів у коді. Ви керуєте даними за допомогою класів і структур у своєму програмному коді. Цей метод має проблеми, оскільки структура бази даних тісно пов’язана з кодом, що ускладнює видалення бази та перенесення її в іншу програму.

Шаблони відображення даних. Шаблон відображення даних намагається відокремити бізнес-логіку в об’єктах від бази даних. Це поділ може полегшити зміну баз даних і використовувати ту саму логіку програмування.

ORM проти SQL. Розробники можуть використовувати необроблений код SQL для створення прямого інтерфейсу між програмою та базою даних. Більшість реляційних баз даних підтримують SQL для створення інтерфейсів даних і програм. Він стабільний, а оскільки SQL використовується з 1970-х років, він добре задокументований і підтримується. Програмісти підтримують великий контроль над своїм інтерфейсом даних за допомогою SQL. Він вимагає багато роботи, але він більш гнучкий і детальний, ніж абстракція ORM.

Плюси об'єктно-реляційного відображення. Інструменти ORM популярні серед розробників ООП, оскільки вони мінімізують обсяг знань SQL, необхідних для підключення бази даних до програми. ORM також автоматично генерує код SQL, що дозволяє зосередитися на генерації бізнес-логіки. Використання об’єктних карт для керування інтерфейсом між програмами та базами даних має чотири значущі переваги.

Продуктивність. Написання коду доступу до даних займає багато часу і не додає великої цінності функціональності програми. Це, по суті, розробка коду. Використання такого інструменту, як ORM, який автоматично генерує код доступу до даних, заощаджує величезний час розробки, що не додає цінності додатку. У деяких випадках ORM може

записати 100 відсотків коду доступу до даних для програми. ORM також може допомогти вам відстежувати зміни бази даних, що полегшує налагодження та зміну програми в майбутньому.

Дизайн додатків. Добре написаний ORM реалізує шаблони проектування, щоб змусити вас використовувати найкращі методи розробки додатків. Якщо ви використовуєте ORM для керування інтерфейсом даних, вам не потрібно заздалегідь створювати ідеальну базу даних схеми. Ви зможете легко змінити існуючий інтерфейс. Відокремлення таблиці бази даних від програмного коду також дозволяє вимкнути дані для різних додатків.

Повторне використання коду. Одним із способів повторного використання даних є створення бібліотеки класів для створення окремої бібліотеки динамічних посилань (DLL). Ви можете створити нову програму без необхідності дублювати код доступу до даних.

Зменшене тестування. Оскільки код, згенерований ORM, добре перевірений, вам не потрібно витрачати стільки часу на тестування коду доступу до даних. Замість цього ви можете зосередитися на тестуванні бізнес-логіки та коду.

1.5 Опис технології Entity Framework

Entity Framework. Це об’єктно-орієнтована структура відображення (ORM), яку можна використовувати для зберігання та доступу до даних із бази даних.[6] Він забезпечує автоматизований механізм для розробників. Microsoft зробила його доступним для розробки як частину платформи .NET. Entity Framework має різні функції, такі як запити LINQ, відстеження змін, оновлення та міграції схем. Він також може ефективно працювати з такими базами даних, як SQL, MySQL, SQLite та Azure cosmos.

Використання Entity Framework в C#. Завдяки Entity Framework розробники можуть працювати з даними в об’єктах і властивостях домену, таких як клієнти та адреси клієнтів, не турбуючись про базові таблиці та стовпці бази даних, де ці дані зберігаються.

Entity Framework використовується в наступних сценаріях:

- Створіть веб-програму MVC
- Налаштуйте стиль сайту
- Встановіть Entity Framework 6
- Створити модель даних
- Створити контекст бази даних
- Ініціалізуйте БД тестовими даними
- Налаштуйте EF 6 для використання LocalDB
- Створіть контролер і представлення

Підходи до розробки сутності. Існують в основному три підходи до створення структур сутності:

Code First Approach. Цей підхід спочатку спрямований на базу даних, яка не існує, а потім створює її. Це дозволяє розробникам визначати та створювати нові моделі з класами C# та .NET. У цьому підході ви можете використовувати порожні бази даних і також додавати таблиці.

Model First Approach. Ця модель найкраще підходить для нових проектів, де не існує бази даних. Ця модель зберігається за допомогою файлу EDMX і може бути переглянута та відредагована розробником.

Database First. Цей підхід є альтернативою для підходу перш за кодом і підходу на основі моделі. Він створює модель і коди з бази даних у проекті та з'єднує їх з базою даних і розробником.

Особливості Entity Framework. Entity Framework має наступні особливості:

- Кроссплатформерність. EF — це кросплатформний фреймворк, який може працювати на Linux, Windows і Mac.
- Відстеження змін. Структура сутності відстежує різні зміни, що відбулися в екземплярах, які необхідно подати до бази даних.
- Збереження. EF допомагає виконувати операції вставки, оновлення та видалення на основі змін, що відбулися в об’єктах бази даних.

Спочатку потрібно визначити модель. Визначення моделі складається з класів домену, класів контексту, похідних від DbContext, і конфігурації. Щоб вставити дані, вам потрібно додати об’єкт домену для контексту та викликати метод savechanges(). Вам потрібно використати команду вставки та виконати її в базі даних.

Для читання даних буде корисно виконати запит LINQ-to-Entities на бажаній мові, як-от C# або .NET. EF API перетворить запит у запит SQL, який буде надано базі даних для виконання.

Для редагування, оновлення, видалення та видалення об’єктів сутностей слід викликати метод savechanges(). EF API буде створювати та виконувати команди в базі даних.

Як працює Entity Framework. Entity Framework API може відображати зміни домену зі схемою бази даних, перекладати та виконувати запити LINQ до SQL. Це також допомагає відстежувати зміни та зберігати їх у базі даних.

Модель даних сутності. EF API починає свою роботу з створення моделі даних сутності. Він представляє всю базу даних: концептуальну модель, модель зберігання та відображення між ними.

Запит. EF API перетворює запити LINQ у запити SQL для реляційних баз даних за допомогою EDM і перетворює їх назад у об’єктні сутності.

Збереження. EF виконує операції редагування, оновлення, видалення та видалення під час виклику команди savechanges(). ChangeTrack відстежує всі дії та дії.

Контекстний клас у Entity Framework. Це найважливіший клас, коли мова йде про роботу з EF 6 або EF Core. Він допомагає виконувати команди створення, читання, оновлення та видалення в схемі бази даних. Це похідний клас у Entity Framework. Він також використовується для налаштування класів домену, карт баз даних, відстеження змін і зберігання кешу.

Що таке Entity в Entity Framework. Сутність — це клас у Entity Framework, який відображає таблиці бази даних.

EF API може зіставити кожну сутність з таблицями, а кожну властивість об'єкта – зі стовпцем у базі даних. Сутності в основному складаються з двох типів у структурі сутності:

1) Скалярна властивість. Скалярні властивості - це ті властивості, які є примітивними. Він відображає стовпець у таблиці бази даних, у якій зберігаються дані. Наприклад, ідентифікатор студента, ім'я студента, клас є скалярною властивістю студентської сутності.
2) Властивість навігації. Ця властивість відповідає за відносини одного суб'єкта з іншим. Існує два типи властивостей навігації:
    i) Властивість довідкової навігації. Якщо сутність має властивість іншого типу властивості сутності, вона називається властивістю посилання навігації. Він вказує на одну сутність, але представляє множинність одного.
    ii) Колективна навігаційна власність. Коли сутність має властивість загального типу колекції, вона відома як колективна навігаційна властивість. Він має множинність багатьох.

Типи сутностей у Entity Framework. У структурі сутності є два типи сутностей:

1) Об'єкти POCO (звичайні старі об'єкти CLR). Це клас сутності, який не залежить від жодного базового класу, специфічного для фреймворку. Це звичайний клас .NET CLR, тому він називається "Звичайні старі об'єкти CLR". Він підтримує багато операцій, таких як оновлення, створення та видалення, які генеруються моделлю даних сутності.
2) Динамічні проксі-об'єкти (POCO-проксі). Динамічні проксі-суб'єкти є класами часу виконання і завершують сутність POCO. Ці сутності є класами з відкладеним завантаженням. Він підтримується лише EF 6; EF Core 2.0 поки що не підтримує його.

Стани сутностей. EF API відповідає за підтримку станів сутностей протягом їхнього життєвого циклу. Кожна сутність має стан відповідно до операції, яку виконує контекстний клас. Стан сутності може бути представлений перерахуванням System.Data.Entity.EntityState в EF 6.

Зберігання в Entity Framework. У структурі сутності є два типи сценаріїв збереження:

1) Пов'язані сценарії. У цьому сценарії той самий клас контексту використовується для отримання та збереження об’єктів. Він відстежує всі бази даних протягом усього життєвого циклу. Це корисно для баз даних для тієї ж мережі.
2) Відключені сценарії. У цьому сценарії для отримання та збереження даних використовується інший контекст. Один екземпляр класу контексту використовується для отримання даних, а інший використовується для збереження даних.

1.6 Redis

Redis (віддалений сервер словників) — це сховище структури даних у пам’яті, що використовується як розподілена база даних ключ-значення в пам’яті, кешу та посередника повідомлень, з необов’язковою

довговічністю. Redis підтримує різні види абстрактних структур даних, такі як рядки, списки, карти, набори, відсортовані набори, HyperLogLogs, растрові зображення, потоки та просторові індекси. Проект був розроблений і підтримуваний Сальваторе Санфіліппо.

Відповідно до щомісячних рейтингів DB-Engines, Redis часто є найпопулярнішою базою даних ключів і значень. Redis також зайняв рейтинг №4 NoSQL бази даних NoSQL за задоволеністю користувачів і присутності на ринку на основі відгуків користувачів[24], найпопулярнішою базою даних NoSQL у контейнерах і №4 сховищ даних 2019 року за рейтингом веб-сайту stackshare.io.

Починаючи з версії 2.6, Redis пропонує сценарії на стороні сервера мовою Lua.

Багато мов програмування мають мовні прив'язки Redis на стороні клієнта, зокрема: ActionScript, C, C++, C#, Chicken, Clojure, Common Lisp, Crystal, D, Dart, Elixir, Erlang, Go, Haskell, Haxe, Io, Java, Nim, JavaScript (Node.js), Julia, Lua, Objective-C, OCaml, Perl, PHP, Pure Data, Python, R, Racket, Ruby, Rust, Scala, Smalltalk, Swift і Tcl. На цих мовах існує декілька програм-клієнтів.

Redis зіставляє ключі з типами значень. Важливою відмінністю Redis від інших структурованих систем зберігання є те, що Redis підтримує не тільки рядки, а й абстрактні типи даних:

- Списки рядків
- Набори рядків (колекції неповторюваних несортованих елементів)
- Відсортовані набори рядків (колекції неповторюваних елементів, упорядкованих за числом з плаваючою комою, яке називається оцінкою)
- Хеш-таблиці, де ключі та значення є рядками
- Журнали HyperLogLog, які використовуються для наближеної оцінки розміру потужності набору.

- Потік записів із групами споживачів дозволяє зберігати кілька полів і рядкових значень з автоматичною послідовністю на основі часу в одному ключі.
- Геопросторові дані через реалізацію техніки геохеш.

Тип значення визначає, які операції (так звані команди) доступні для значення. Redis підтримує високорівневі атомарні операції на стороні сервера, такі як перетин, об'єднання та різницю між наборами та сортування списків, наборів і відсортованих наборів.

Більше типів даних підтримується на основі Redis Modules API:

- JSON – RedisJSON реалізує ECMA-404 (стандарт обміну даними позначення об'єктів JavaScript) як власний тип даних.
- Graph – RedisGraph реалізує запитуваний графік властивостей
- Часовий ряд – RedisTimeSeries реалізує структуру даних часового ряду
- Фільтр Блума, фільтр зозулі, ескіз Count–min і Top-K – RedisBloom реалізує набір імовірнісних структур даних для Redis.

1.7 JWT токен для авторизації та аутентифікації

Що таке JWT (веб-токен JSON). JWT, або JSON Web Token, — це відкритий стандарт, який використовується для безпечного обміну інформацією між двома сторонами — клієнтом і сервером. У більшості випадків це закодований JSON, що містить набір вимог і підпис. Зазвичай він використовується в контексті інших механізмів аутентифікації, таких як OAuth, OpenID для обміну інформацією, пов'язаною з користувачами. Це також популярний спосіб аутентифікації/авторизації користувачів в архітектурі мікро сервісів.

Аутентифікація JWT. Це механізм автентифікації без стану на основі маркерів. Він широко використовується як сеанс без стану на

стороні клієнта, це означає, що серверу не потрібно повністю покладатися на сховище (або) базу даних для збереження інформації про сеанс.

JWT можуть бути зашифровані, але зазвичай вони закодовані та підписані. Ми зосередимося на підписаних JWT. Метою Signed JWT є не приховування даних, а забезпечення достовірності даних. І саме тому настійно рекомендується використовувати HTTPS з підписаними JWT.

Структура JWT. Структура JWT розділена на три частини: заголовок, корисне навантаження, підпис і відокремлена один від одного крапкою (.), і буде відповідати структурі нижче:

- Заголовок. Заголовок складається з двох частин:
  i) Використовується алгоритм підписання
  ii) Тип токена, який у даному випадку переважно «JWT»
- Корисне навантаження. Корисне навантаження зазвичай містить претензії (атрибути користувача) та додаткові дані, такі як емітент, час закінчення терміну дії та аудиторія.
- Підпис. Зазвичай це хеш розділів заголовка та корисного навантаження JWT. Алгоритм, який використовується для створення підпису, — це той самий алгоритм, який зазначено в розділі заголовка JWT. Підпис використовується для підтвердження того, що маркер JWT не був змінений або змінений під час передачі. Його також можна використовувати для підтвердження відправника.

Заголовок і розділ корисного навантаження JWT завжди кодуються Base64.

Як працює аутентифікація JWT. Коли справа доходить до аутентифікації API та авторизації між серверами, веб-токен JSON (JWT) є особливо корисною технологією. З точки зору єдиного входу (SSO), це означає, що постачальник послуг може отримувати надійну інформацію від сервера аутентифікації.

Подаючи секретний ключ постачальнику ідентифікаційних даних, постачальник послуг може хешувати частину отриманого токена та порівняти його з підписом токена. Тепер, якщо цей результат збігається з підписом, SP знає, що надана інформація надійшла від іншого об’єкта, який володіє ключем.

Оскільки маркери використовуються для авторизації та аутентифікації в майбутніх запитах і викликах API, необхідно бути дуже обережними, щоб запобігти проблемам безпеки. Ці маркери не повинні зберігатися в загальнодоступних областях, як-от локальне сховище браузера або файли cookie. Якщо іншого вибору немає, корисне навантаження має бути зашифровано.

Як JWT Single Sign-On (SSO) працює для кількох веб-програм. Єдиний вхід (SSO) дозволяє вам автентифікувати користувачів у ваших системах і згодом інформує програми про те, що користувач пройшов автентифікацію. Після успішної аутентифікації генерується і повертається маркер JWT, який може використовуватися програмою для створення сеансу користувача. Маркер автоматично перевіряється з IDP, коли вони входять. Тоді користувачеві надається доступ до програм без запиту на введення окремих облікових даних для входу.

Цей механізм безпеки дозволяє програмам довіряти запитам на вхід, які вони отримують від систем. Крім того, ці програми нададуть доступ лише користувачам, які пройшли автентифікацію вами/адміністратором, і, отже, єдиний вхід (SSO) покладається на веб-токен JSON (JWT) для забезпечення обміну даними автентифікації користувачів. Необхідно дуже уважно ставитися до того, як цей маркер зберігається та керується.

1.8 Популярні види архітектури веб додатків

Існує багато різних видів архітектури, але найуживанішими є багаторівнева та цибулева (Onion). Розглянемо їх:

Багаторівнева. У програмній інженерії багаторівнева архітектура (часто її називають архітектурою n-рівня) або багатошарова архітектура - це архітектура клієнт-сервер, в якій функції презентації, обробки додатків та управління даними фізично розділені. Найпоширенішим використанням багаторівневої архітектури є трирівнева архітектура.

Архітектура N-рівня додатків забезпечує модель, за допомогою якої розробники можуть створювати гнучкі та багаторазові додатки. Розділяючи програму на рівні, розробники отримують можливість змінювати або додавати певний шар, замість того, щоб переробляти всю програму. Трирівнева архітектура, як правило, складається з рівня презентації, логічного рівня та рівня даних.

Хоча поняття рівня та рівня часто використовуються як взаємозамінні, одна досить загальна точка зору полягає в тому, що насправді існує різниця. З цієї точки зору стверджується, що рівень є логічним механізмом структурування елементів, що складають програмне рішення, тоді як рівень - це фізичний механізм структурування для системної інфраструктури. [7][8] Наприклад, тришарове рішення можна легко розгорнути на одному рівні, наприклад, на персональній робочій станції. [7]

Onion Архітектура. Більшість традиційних архітектури порушують фундаментальні питання тісного зв’язку та розділення проблем. Цибулеву архітектуру запровадив Джеффі Палермо, щоб забезпечити кращий спосіб створення додатків з точки зору кращої перевірочної, ремонтопридатності та надійності. Onion Architecture вирішує проблеми, з якими стикаються 3-рівневі та n-рівневі архітектури,

та надає рішення загальних проблем. Шари архітектури цибулі взаємодіють між собою за допомогою інтерфейсів.

Цибульна архітектура базується на принципі інверсії управління. Цибульна архітектура складається з безлічі концентричних шарів, що взаємодіють один з одним до ядра, що представляє домен. Архітектура залежить не від рівня даних, як у класичних багаторівневих архітектурах, а від реальних моделей доменів.

Відповідно до традиційної архітектури, рівень інтерфейсу користувача взаємодіє з бізнес-логікою, а бізнес-логіка розмовляє з рівнем даних, і всі рівні змішані і сильно залежать один від одного. У 3-рівневій та n-рівнівній архітектурах жоден з шарів не є незалежним; цей факт викликає розділення проблем. Такі системи дуже важко зрозуміти та підтримувати. Недоліком цієї традиційної архітектури є непотрібне зчеплення.

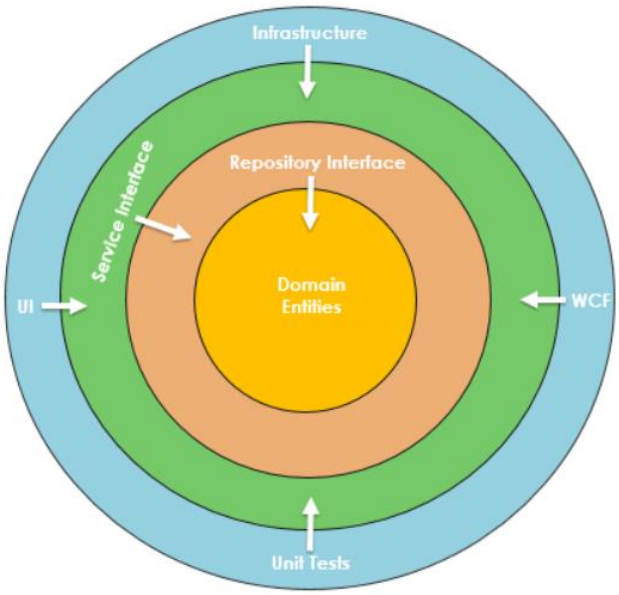


Рис. 2 Шари цибулевої архітектури

Цибульна архітектура вирішила цю проблему, визначивши шари від ядра до інфраструктури. Він застосовує основне правило, переміщуючи всі зчеплення до центру. Ця архітектура, безсумнівно, має тенденцію до об'єктно-орієнтованого програмування, і вона ставить об'єкти перед усіма іншими. У центрі Цибульної архітектури знаходиться доменна модель, яка представляє об'єкти бізнесу та поведінки. Навколо шару домену розташовані інші шари, які мають більше поведінки.

Цибульна архітектура використовує концепцію шарів, але вони відрізняються від шарів архітектури 3-рівня та n-рівня. Давайте подивимося, що кожен із цих шарів представляє і що повинен містити.

У центральній частині Цибульної архітектури існує доменний шар; цей рівень представляє об'єкти бізнесу та поведінки. Ідея полягає в тому, щоб усі ваші об’єкти домену були в цьому ядрі. Він вміщує всі об'єкти домену програми. Окрім об'єктів домену, ви також можете мати інтерфейси домену. Ці сутності домену не мають залежностей. Об'єкти домену також плоскі, як і слід, без будь-якого важкого коду або залежностей.

Шар Сховища. Цей рівень створює абстракцію між сутностями домену та бізнес-логікою програми. У цей шар ми зазвичай додаємо інтерфейси, що забезпечують збереження та отримання поведінки об’єктів, як правило, залучаючи базу даних. Цей рівень складається із шаблону доступу до даних, що є більш вільно пов'язаним підходом до доступу до даних. Ми також створюємо загальне сховище та додаємо запити для отримання даних із джерела, зіставлення даних із джерела даних до суб'єкта господарювання та збереження змін у бізнесі до джерела даних.

Рівень служби містить інтерфейси із загальними операціями, такими як Додати, Зберегти, Редагувати та Видалити. Крім того, цей рівень використовується для зв'язку між рівнем інтерфейсу та рівнем

сховища. Рівень служби також може містити бізнес-логіку для сутності. У цьому рівні службові інтерфейси тримаються окремо від його реалізації, маючи на увазі вільне з'єднання та відокремлення проблем.

Шар інтерфейсу користувача. Це самий зовнішній шар і зберігає периферійні проблеми, такі як інтерфейс та тести. Для веб-додатків він представляє веб-API або проект Unit Test. Цей рівень має реалізацію принципу введення залежностей, так що додаток будує слабо пов'язану структуру і може взаємодіяти з внутрішнім рівнем через інтерфейси.

1.9 Принципи та патерни проектування

SOLID. В об'єктно-орієнтованому комп'ютерному програмуванні SOLID є мнемонічною абревіатурою п'яти принципів проектування, спрямованих на те, щоб зробити дизайн програмного забезпечення більш зрозумілим, гнучким і ремонтопридатним. Принципи є підмножиною багатьох принципів, пропагованих американським інженером-програмістом та інструктором Робертом К. Мартіном, вперше представленим у його роботі "Принципи проектування та шаблони дизайну".[9][10]

SOLID принципи наступні:

- Принцип єдиної відповідальності: "[11]Ніколи не повинно бути більше однієї причини для зміни класу". Іншими словами, кожен клас повинен нести лише одну відповідальність. [12]
- Принцип відкрито-закрито: "Суб'єкти програмного забезпечення ... повинні бути відкриті для розширення, але закриті для модифікації".[13]
- Принцип заміщення Ліскова: "Функції, що використовують покажчики або посилання на базові класи, повинні мати можливість використовувати об'єкти похідних класів, не знаючи цього".[14]

- Принцип розділення інтерфейсу: "Багато клієнт-інтерфейси кращі за один інтерфейс загального призначення." [15]
- Принцип інверсії залежності: "Залежність від абстракцій, [не] конкрементів"

Абревіатура SOLID була введена пізніше, приблизно в 2004 році, Майклом Фезерсом .

Хоча принципи SOLID застосовуються до будь-якого об'єктно-орієнтованого проектування, вони також можуть сформувати основну філософію таких методологій, як гнучка розробка або адаптивна розробка програмного забезпечення.

KISS, абревіатура, keep it simple, stupid - це принцип дизайну, відзначений ВМС США в 1960 році.[16]17] Принцип KISS стверджує, що більшість систем найкраще працюють, якщо їх спрощувати, а не ускладнювати; тому простота повинна бути ключовою метою проектування, а також уникати зайвої складності. Фраза асоціюється з авіаційним інженером Келлі Джонсон.[18] Термін "принцип KISS" був популярним у користуванні до 1970 р. Варіації фрази включають: "Keep it simple, silly", "keep it short and simple", "keep it simple and straightforward", "keep it small and simple", "keep it simple, soldier",or "keep it simple, sailor".

DRY, Не повторюйся (DRY, або іноді не повторюйся) - це принцип розробки програмного забезпечення, спрямований на зменшення повторення програмних шаблонів,[19] замінюючи його абстракціями або використовуючи нормалізацію даних, щоб уникнути надмірності.

Принцип DRY визначається як "Кожна знання повинна мати єдине, однозначне, авторитетне представлення в системі". Цей принцип сформулювали Енді Хант та Дейв Томас у своїй книзі "Прагматичний програміст"[20]. Вони застосовують його досить широко, включаючи "схеми баз даних, плани випробувань, систему збірки, навіть

документацію". Коли принцип DRY застосовується успішно, модифікація будь-якого окремого елемента системи не вимагає зміни в інших логічно не пов’язаних елементах. Крім того, елементи, які логічно пов'язані, змінюються передбачувано і рівномірно, і, таким чином, синхронізуються. Окрім використання методів і підпрограм у своєму коді, Томас і Хант покладаються на генератори коду, автоматичні системи збірки та мови сценаріїв, щоб дотримуватися принципу DRY на різних рівнях.

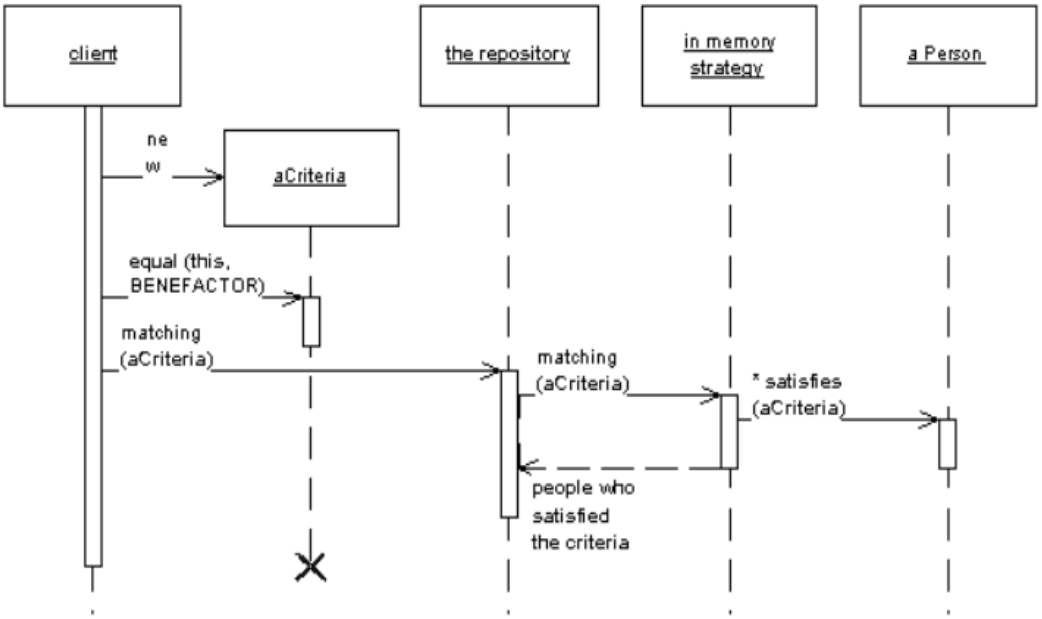


Рис.3 Патерн Репозиторій

Патерн Репозиторій. Система зі складною моделлю домену часто виграє від рівня, такого як той, що надається Data Mapper (165), який ізолює об'єкти домену від деталей коду доступу до бази даних. У таких системах може бути корисним побудувати ще один шар абстракції над шаром відображення, де зосереджений код побудови запиту. Це стає більш важливим, коли існує велика кількість класів доменів або великі запити. Особливо в цих випадках додавання цього шару допомагає мінімізувати логіку повторюваних запитів.

Репозиторій здійснює посередницьку діяльність між шарами відображення домену та даних, діючи як колекція об’єктів домену в пам'яті. Клієнтські об'єкти декларативно створюють специфікації запитів і надсилають їх до сховища для задоволення. Об'єкти можна додавати та видаляти зі сховища, як і з простої колекції об'єктів, а інкапсульований сховищем код відображення буде виконувати відповідні операції за кадром. Концептуально Репозиторій інкапсулює набір об'єктів, що зберігаються в сховищі даних, і операції, що виконуються над ними, забезпечуючи більш об'єктно-орієнтоване уявлення про рівень стійкості. Сховище також підтримує мету досягнення чистого розділення та односторонньої залежності між шарами домену та відображення даних.

1.10 Юніт тестування веб додатків

Що таке модульне (юніт) тестування. Модульне тестування — це спосіб тестування одиниць — найменших компонентів вашого програмного забезпечення, найменшого фрагмента коду. Одиницею може бути одна функція. Мета полягає в тому, щоб підтвердити, що кожен блок працює належним чином. Модульний тест перевіряє поведінку окремо від інших тестів. Якщо тест покладається на зовнішню систему, це не модульний тест. Модульні тести повинні бути написані на етапі проектування, до впровадження, щоб запобігти розгортанню дефектів у виробництві. Вони повинні запускатися щоразу, коли код змінюється, і надавати чіткий опис функції, що тестується.

Покриття модульного тесту. Є підтипом покриття коду — воно показує, які рядки коду були перевірені модульними тестами. Однак мета не полягає в тому, щоб охопити 100% модульними тестами. Насправді ви повинні мати якомога менше модульних тестів і охоплювати якомога більше типів поведінки. 100% покриття модульних тестів існує лише в ідеальному світі. Під час написання модульних тестів зосередьтеся на

поведінці — додавання нових тестів лише для покращення охоплення призводить до тестів без будь-якої мети, і ви просто витрачаєте свій час.

Модульне тестування можна виконувати вручну, але зазвичай це автоматизовано. З точки зору автоматизованих тестів, існує концепція, яка називається тестовою пірамідою, яка показує, як ефективно збалансувати автоматизовані тести.

Піраміда тестів складається з наступних рівнів:

1) End-to-End Tests
2) Integration Tests
3) Unit Tests

Модульні тести зазвичай перевіряють функції, методи та класи. Вони короткі, дешеві та швидкі для написання та підтримки. Вони зосереджені на одній речі/функції. Вони не залежать від будь-яких зовнішніх систем.[21][22] Коли модульний тест зазнає невдачі, легко визначити проблему. Є два типи модульних тестів:

- Позитивне тестування. Тестування коду шляхом надання дійсних даних.
- Негативне тестування. Тестування коду шляхом надання недійсних даних.

Інтеграційний тест. Інтеграційний тест зосереджується на точці, де окремі підрозділи об’єднуються та тестуються як група. Мета – перевірити, чи працює взаємодія між підрозділами.

End-to-End тести. Наскрізні тести охоплюють всю програму - від початку до кінця. Вони дорожчі і повільніше працюють.

Піраміда показує, що чим більше високий рівень ви отримаєте, тим менше тестів вам слід виконати. Напишіть багато невеликих і швидких модульних тестів, деякі інтеграційні тести і лише кілька тестів високого рівня. Щоб підтримувати цей баланс, використовуйте безперервну

доставку – практику, коли ви автоматично гарантуєте, що ваш продукт готовий до випуску у виробництво.

Переваги модульного тестування. Модульне тестування допоможе вам:

- збереження вашого коду
- виявити дефекти, що виникли внаслідок зміни коду
- зменшити потенційно шкідливий вплив змін у вашому коді
- повторне використання коду
- швидший розвиток
- зниження витрат на усунення дефектів на нижчому рівні тестування
- документація коду, оскільки модульні тести описують, що робить ваш продукт

РОЗДІЛ 2. ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

2.1 Бізнес-логіка проекту

Бізнес модель проекту складається з наступних пунктів:

- Сервіс створюється для батьків та гуртків.
- Надавач послуг може створити обліковий запис ввівши відповідні дані. Надавач послуг може створювати нові гуртки ввівши про них відповідні дані.
- Надавач послуг може додавати вчителів до гуртків.
- Батьки можуть створювати обліковий запис, після чого додавати дітей вводячи про них відповідні дані.
- Батьки можуть записувати дітей до гуртків, залишати коментарі про гуртки, вчителів та ін.
- Гуртки можуть проводити набір, набирати обмежену кількість дітей. Також батьки і гуртки мають спільні чати, тому, в разі потреби, можуть спілкуватися.
- Доступна фільтрація гуртків за ціною, соціальними групами, категоріями т а ін.
- Також всі адреси відображаються на картах.
- Існує адміністратор що може проводити адміністрування сайту. Наприклад додавати нові категорії різних рівнів.
- Інше

2.2 Юзер інтерфейс сервісу

Знайомство юзера з сервісом починається на головні сторінці, де він може побачити популярні напрямки, популярні гуртки, виконати пошук по назві чи місту, подивитися інформацію про сам портал, а також увійти до особистого кабінету чи зареєструватися.

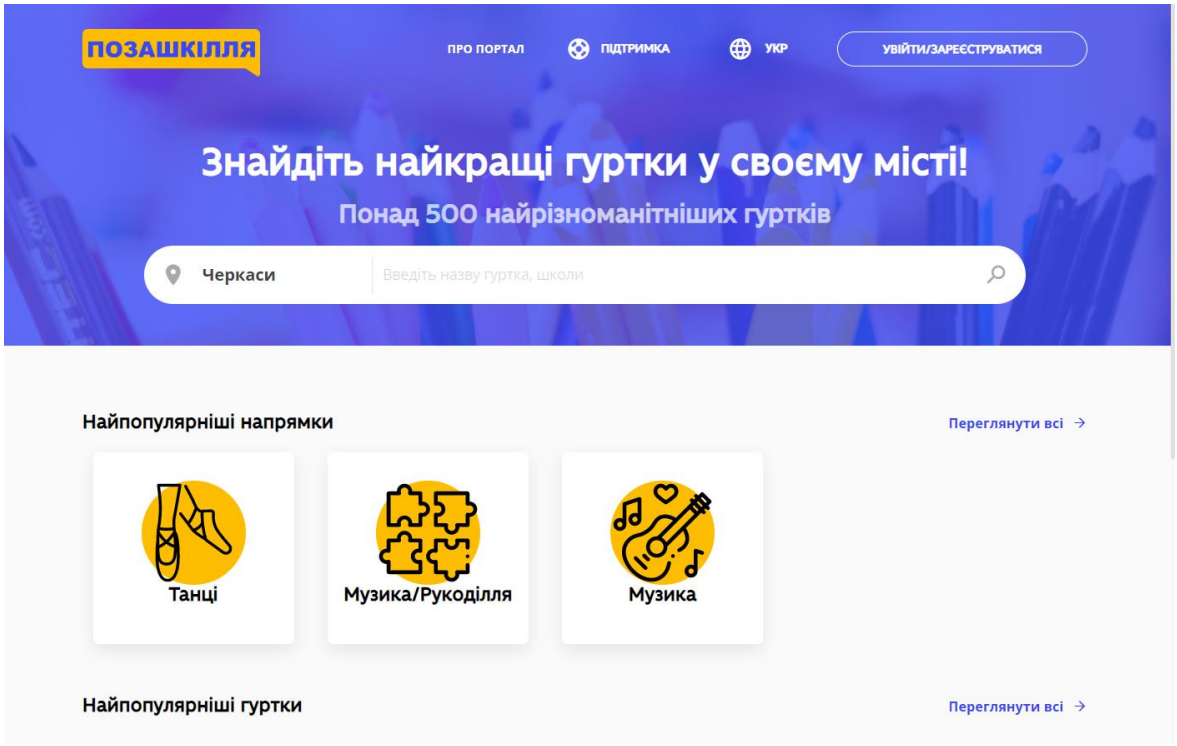


Рис.3 Головна сторінка

Перейдемо до важливого кроку, що дає можливість користуватися сервісом у повній мірі – вхід до кабінету, або авторизація. Авторизація та аутентифікація здійснюється за допомогою окремого проекту .NET MVC з використанням технології Identity Server 4 на основі JWT токену.

Для успішного логіну потрібно ввести електронну пошту та пароль від особистого кабінету юзера.

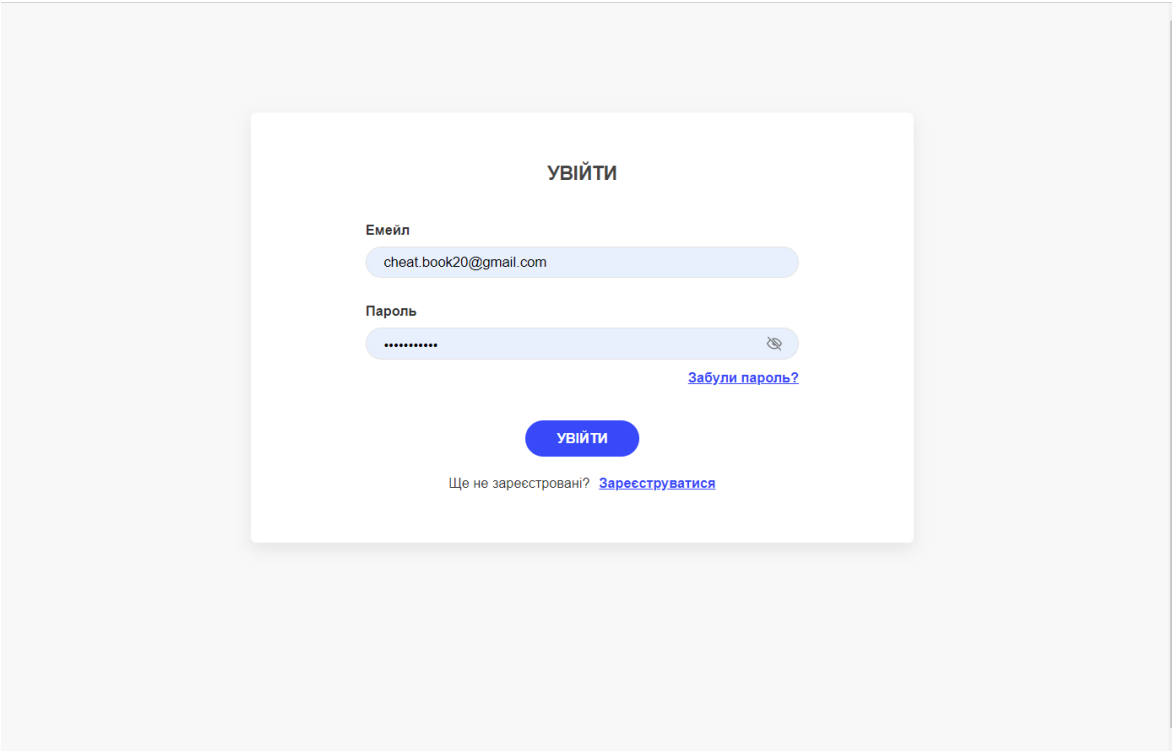


Рис.4 Вхід до особистого кабінету

При потребі пароль можна оновити. При відсутності облікового запису існує можливість реєстрації. Існує два кейси реєстрації:

- Якщо юзер є одним з батьків
- Якщо юзер є надавачем послуг (гуртком)

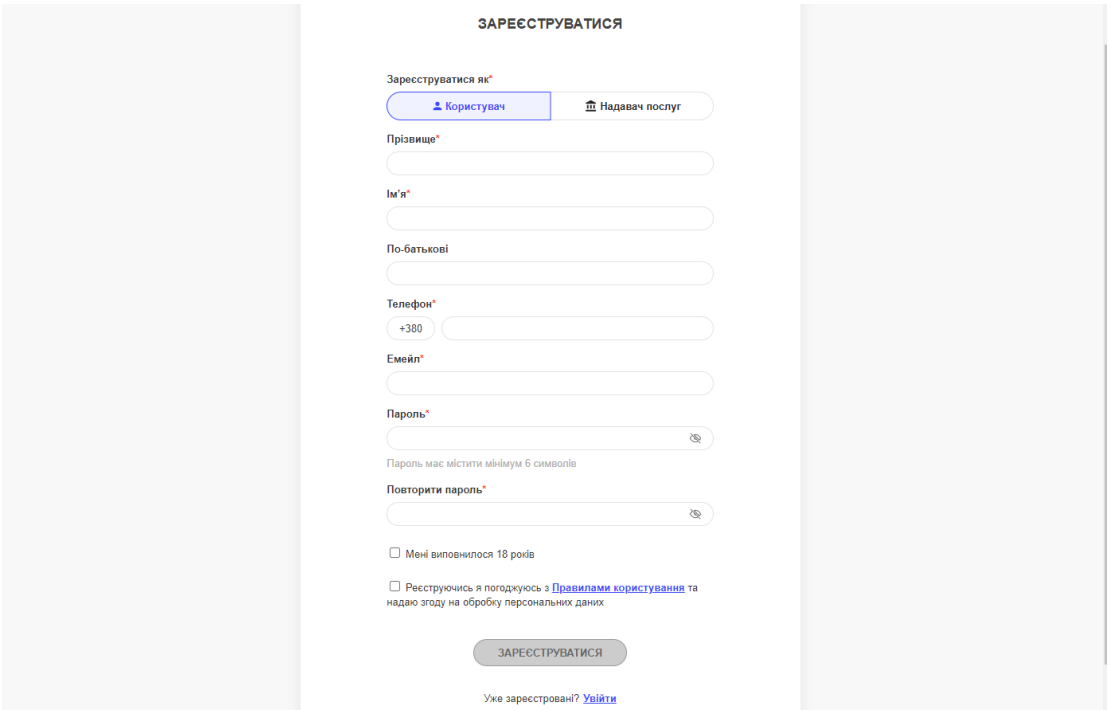


Рис.5 Реєстрація

Для успішної реєстрації необхідно заповнити наступні поля:

- Роль. Вибрати між користувачем та гуртком
- Ім’я
- По-батькові
- Номер телефону, що належить до України
- Електронна адреса
- Пароль та його підтвердження
- Погодження з віком та правилами користування

Після проведення реєстрації проводиться автоматичний логін і перехід назад до сервісу, але тепер ми знаходимося в сервісі як залогінений користувач, маємо більше прав та можливостей.

2.2.1 Функціонал батьків

Після реєстрації користувач отримує можливість потрапити до особистого кабінету, де йому доступні наступні розділи:

- **Особиста інформація** - де користувач може редагувати данні про себе (Прізвище, Ім’я, По-батькові, Номер телефону), а також має можливість змінити свій email або пароль до облікового запису.
- **Інформація про дитину** - можливість додати дитину (Цей функціонал буде описано нижче).
- **Мої гуртки** – сторінка, де користувач може подивитися до яких гуртків записана дитина.
- **Заяви** – сторінка, де можна подивитися статус заявки до гуртка по кожній дитині, всього існує 4 статуси: Очікують підтвердження, Зараховано, Відмовлено, Гурток залишено.
- **Улюблене** – можна зберегти улюблені гуртки.

Логічно, що першим чином новий користувач має додати дитину. Для цього треба перейти в особистий кабінет, вкладка інформація про

дитину та натиснути кнопку додати дитину, після чого відкриється форма, що містить усі необхідні поля.

Щоб додати дитину користувачу необхідно заповнити наступні поля:

- Прізвище
- Ім’я
- По-батькові
- Дата народження
- Стать
- Соціальна група (вибрати з існуючих)
- Місце проживання
- Місце навчання
- Серія та номер свідоцтва про народження

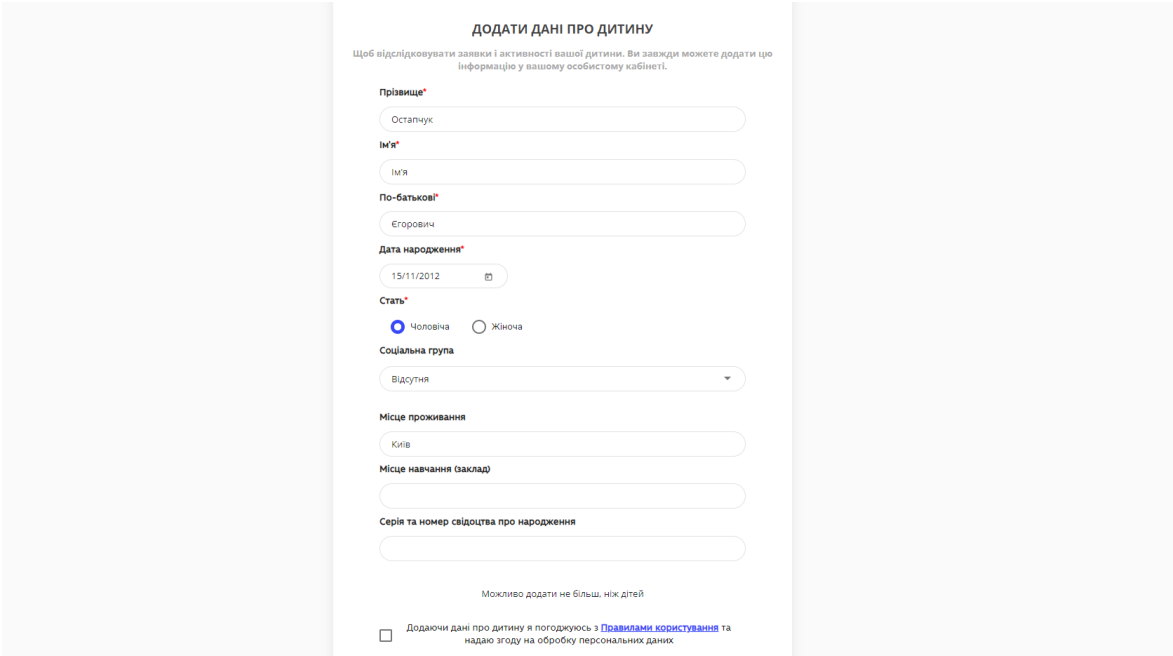


Рис.6 Реєстрація дитини

Після реєстрації дитини користувач може перейти до сторінки з гуртками. Ввівши назву свого міста користувач може отримати список найпопулярніших гуртків, а також можливість вибрати гурток за напрямком. Зазначені напрямки: Музика, Танці, Спорт, Співи,

Малювання, Рукоділля, Театр, Кіно, Наука, Техніка, Програмування, Фотографія, Садівництво, Оздоровлення та багато всього іншого.
Користувач може подивитися на усі можливі гуртки на карті та вибрати той, що знаходиться ближче для нього.

Також користувач може вибрати свої налаштування у фільтрах. Доступні наступні налаштування:

- Напрямки
- Вік
- Дні роботи
- Години роботи
- Вартість
- Наявність пільгових місць

Після застосування фільтра отримуємо список гуртків по нашим бажанням.

Кликнувши на іконку гуртка ми переходимо до інформації про сам гурток.
Доступні наступні вкладки:

- Про гурток (вартість заняття, опис гуртка, наявність пільгових місць, дні та години роботи)
- Про заклад (форма власності, тип організації, ПІБ керівника, опис, засновник)
- Вчителі (список вчителів з їхніми фото, ПІБ та описом)
- Інші гуртки школи
- Відгуки, де користувач може залишити свій відгук
- Контакти, де можна побачити адресу гуртка, номер телефону, посилання на соціальні мережі

При подачі заявки на гурток форма заповнюється автоматично, треба лише проставити галочки навпроти різних погодженостей.

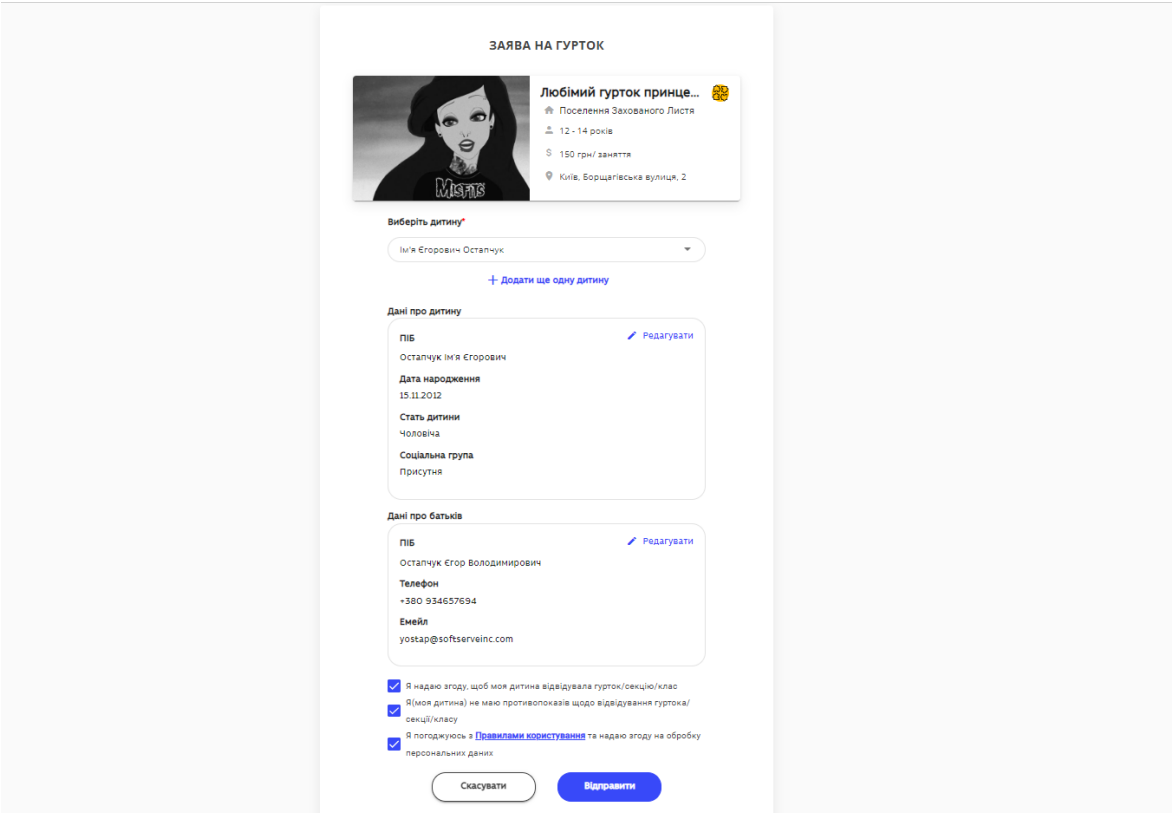


Рис.7 Вигляд заявки до гуртка

Після відправки форми заявка буде знаходитися в особистому кабінеті, в статусі “Очікує підтвердження”. Відповідний менеджер гуртку повинен зконтактувати з вами та перевести її до іншого статусу. При бажанні користувач може відмінити свою заявку.

2.1.2 Функціонал гуртків

Після реєстрації користувача як надавача послуг з’являється форма, яку треба заповнити в 3 етапи.

Перший етап – **Інформація** про заклад включає в себе наступні поля:

- Форма власності (Громадська організація/Приватна/Державна)
- Тип організації (ФОП/ТОВ/ПП/Заклад освіти/Громадська орг./ Інше)
- Повна назва організації

- Скорочена назва організації
- ПІБ Керівника
- Дата народження керівника
- ІПН
- Номер телефону
- Електронна пошта
- Сторінки в соціальних мережах
- ПІБ Засновника

Другий етап – **Контакти** про заклад включає в себе наступні поля:

- Область
- Місто
- Район
- Вулиця
- Будинок

Дана форма дублює ці поля для юридичної та для фактичної адреси. Існує можливість поставити галочку, якщо ці адреси збігаються і заповнити лише одну із них.

Третій етап – **Опис** закладу. Для того, щоб опистаи заклад необхідно додати повноцінний опис до 2000 символів, та вибрати відповідний статус закладу (Працює/Не працює/Зареєстровано). А також дати згоду на обробку особистої інформації.

Після збереження даних створюється нова організація. Аккаунт користувача стає валідним і він може перейти до особистого кабінету.

У власному кабінеті йому доступні наступні вкладки:

- Особиста інформація. Де користувач може побачити своє Прізвище, Ім’я, По-батькові, Телефон, Електронну адресу. За потреби змінити їх, а також має можливість змінити пароль.

- Інформація про заклад. Уся інформація, що була вказана при створенні закладу з можливістю редагувати її.
- Гуртки. Можна побачити свої гуртки або створити новий.
- Заяви. Можна побачити заяви до гуртків, а також управляти ними.
- Адміністрування

Логічно, що після реєстрації закладу наступним іде реєстрація гуртка. Для цього потрібно перейти до вкладки Гуртки в особистому кабінеті та натиснути кнопку “Додати гурток”. Після чого відкриється форма для створення гуртка. Форма складається з 4-х частин:

- Про гурток. В цій частині потрібно заповнити Назву, Емейл, Веб сайт, Facebook, Instagram, Вік дітей, Дні та години роботи, Вартість, а також додати емблему гуртка.
- Опис. В цьому розділі потрібно додати фото гуртка, опис до 500 символів, ПІБ Директора, Наявність пільг, Напрямок навчання, Додати ключові слова
- Адреса. На карті потрібно вибрати положення гуртка, данні автоматично заповняться у відповідних полях.
- Викладачі. Існує можливість додати декількох викладачів закладу. Для того, щоб додати викладача потрібно заповнити наступні поля: Фото, Прізвище, Ім’я, По-батькові, Дата народження, Опис.

Після завершення гурток появиться в особистому кабінеті користувач, та буде видимим іншим користувачам.

При надходженні нової заявки в особистому кабінеті буде індикатор сповіщення, а в розділі заявки з’явиться нова заявка яку можна обробити. Прийняти чи відхилити.

Після підтвердження заявка перейде до статусу зараховано. При бажанні зарахованого учня можна виключити вказавши причину

відмови, після чого вона перейде до розділу відмовлено та може бути повернута до розділу зараховано.

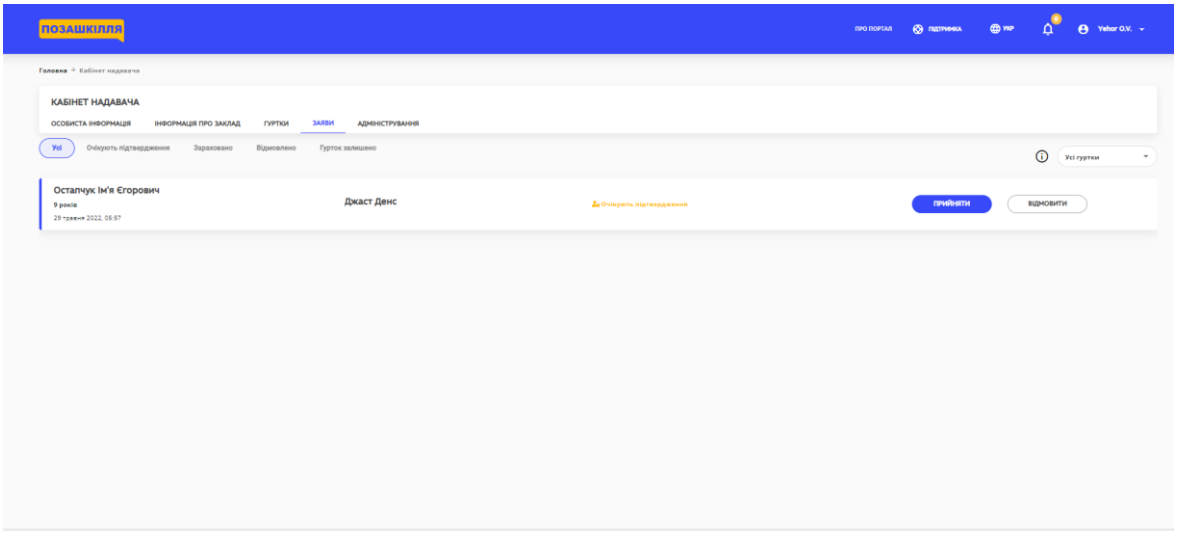


Рис. 8 Заявки в особистому кабінеті надавача послуг

Також надавачу послуг доступний розділ адміністрування, де він може створити нового користувача з роллю заступника директора або адміністратора гуртка додав відповідні контакти, після чого запрошення до порталу прийде їм на електронну пошту і вони зможуть адмініструвати гуртки.

2.2 Розробка структури БД

Для збереження даних було обрано СКБД Microsoft SQL Server Management Studio і усе керування таблицями початково виконувалось за допомогою нього.

Також для взаємодії з базою даних використовувався ORM від компанії Microsoft – Entity Framework.

Наша БД повинна зберігати наступні сутності:

- Сутність Юзера (User) – для збереження інформації про обліковий запис. Користувача. Входу/виходу з нього, взаємодії з поштою.
- Роль Юзера (User Role) – для визначення прав нашого користувача.

- Сутність надавача послуг (Provider) з усіма необхідними полями. Ця сутність є юзером, може створювати, змінювати гуртки.
- Сутність Гуртка (Workshop) з необхідними полями.
- Сутність батька (Parent). Батько також є користувачем. Саме на взаємодію з цим користувачем орієнтований весь функціонал веб сервісу.
- Сутність дитини (Child) – створюється батьком, та закріплюється до полів в таблиці батька.
- Сутність адреси (Address) – для можливості додавання об’єктів на карті. Містить в собі назву і координати. Закріплюється до таких сутностей: Provider, Workshop, Child
- Сутність Соціальної групи (Social Group). Закріплюється до дитини.
- Сутність свідоцтва про народження (Birth Sertificate). Створюється при додавання дитини. Додається до таблиці дитини.
- Сутність вчителя (Teacher). Створюється надавачем послуг та закріпляється до гуртків. Вчитель не є користувачем у системі.
- Сутність категорії (Category) – напрямок виду діяльності (мистецтва, спорту, тощо). Закріпляється до гуртків.
- Сутність категорії 2-го рівня (Subcategory) – для більшої деталізації категорій.
- Сутність категорії 3-го рівня (Subsubcategory) – для більшої деталізації категорій 2-го рівня.

Можна трохи детальніше поглянути на структуру БД на рисунку 4. Цикли створення нових БД займають досить велику кількість часу, тому було створено модель міграції у ASP.NET Core і перенос проекту на нову

машину став набагато простіше. Після цього уся робота створення нового екземпляру БД стала лише однією строчкою в консолі:

```
>> dotnet ef database update
```

Ознайомитися з структурою бази даних можна у додатку Д.

2.3 Архітектура проекту

Для проектування сервісу була обрана многорівнева архітектура, адже вона є простою, надійною та гарно зарекомендувала себе в подальшій підтримці проекту.

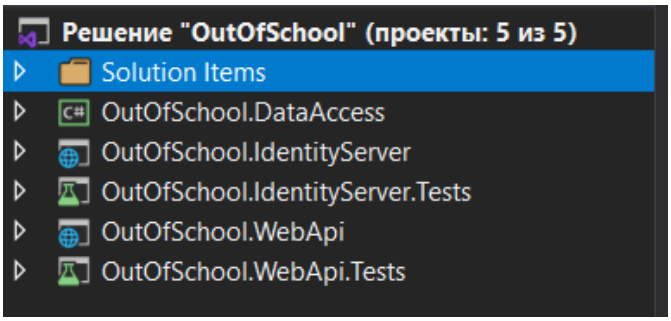


Рис.10 Структура проекту

Солюшн, як видно на рис.10 складається з 4-х проектів:

- DataAccess – для доступу та взаємодії з даними.
- Identity Server – для авторизації та аутентифікації на основі JW token по типу OAuth.
- WebApi – сам сервіс, що взаємодіє з клієнтською частиною.
- WebApi.Tests – юніт тести, що покривають функціонал усього проекту.

Розберемо тепер многорівневу архітектуру.

- Рівень DAL (Data Access Layer) винесений в окремий проект як бібліотека класів.
- Роль рівня BLL (Business Logic Layer) виконує папочка Services, де розташовуються сервіси бізнес логіки, та відповідні інтерфейси.

- Роль рівня PL (Presentation Layer) виконує папочка Controllers, де знаходяться відповідні контролери, що взаємодіють з сервісами, приймають запити від користувачів, повертають дані.

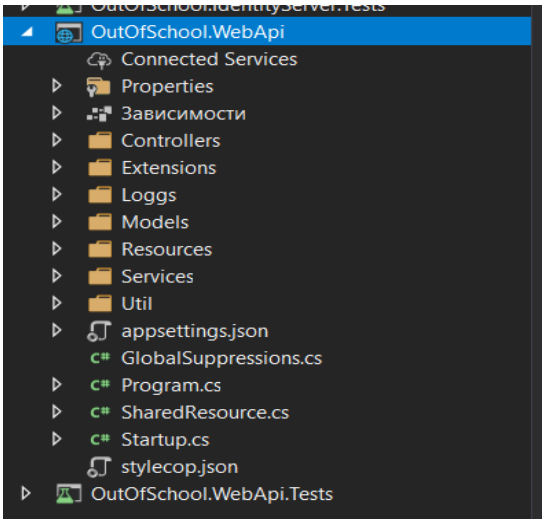


Рис.11 Services and Controllers Folders

Для спілкування нашого DAL з BLL та PL використовуються об’єкти DTO (Data Transfer Object). Тобто об’єкти з ДАЛ є доменними іноді їх треба мапити в ДТО, та навпаки, об’єкти ДТО в домені. Для цього прописані екстеншн методи з використанням автомаперу.

2.4 Рівень доступу до даних (DAL)

Розберемо рівень доступу до даних.

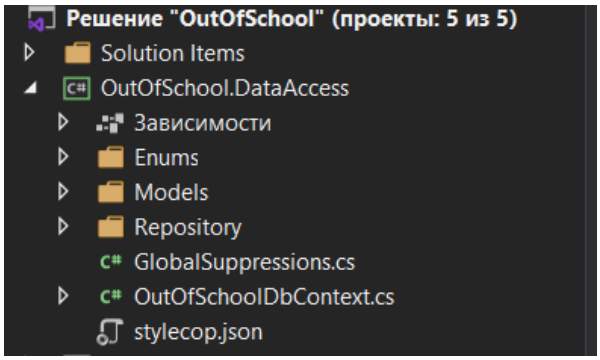


Рис.12 DAL

Рівень доступу до даних фактично включає в себе контекст (OutOfSchoolDbContext) за допомогою якого Entity Framework створює, та взаємодіє з базами даних. Моделі (Models), на основі яких будуються таблиці в БД, та відбувається взаємодія з записами. Також для кожної сутності, що має таблицю для доступу до контексту використовується

патер Репозиторій (Repository). Реалізацію можна розглянути в Додатку А.

```
Ссылок: 46
public class OutOfSchoolDbContext : IdentityDbContext<User>
{
    Ссылок: 25
    public OutOfSchoolDbContext(DbContextOptions<OutOfSchoolDbContext> options)
        : base(options)
    {
    }

    Ссылок: 4
    public DbSet<Parent> Parents { get; set; }

    Ссылок: 3
    public DbSet<Provider> Providers { get; set; }

    Ссылок: 3
    public DbSet<Child> Children { get; set; }

    Ссылок: 2
    public DbSet<Workshop> Workshops { get; set; }

    Ссылок: 0
    public DbSet<Teacher> Teachers { get; set; }

    Ссылок: 5
    public DbSet<Subcategory> Subcategories { get; set; }

    Ссылок: 4
    public DbSet<Subsubcategory> Subsubcategories { get; set; }

    Ссылок: 4
    public DbSet<Category> Categories { get; set; }

    Ссылок: 3
    public DbSet<SocialGroup> SocialGroups { get; set; }

    Ссылок: 3
    public DbSet<Address> Addresses { get; set; }

    Ссылок: 0
    public DbSet<BirthCertificate> BirthCertificates { get; set; }

    Ссылок: 5
    public DbSet<Application> Applications { get; set; }
}
```

Рис.13 DbContext

2.5 Рівень бізнес-логіки (BLL)

Розглянемо рівень бізнес логіки. Бізнес логіка опрацьовується в сервісах. Сервіси є чимось на кшталт посередника між Рівнем доступу до даних та Рівнем презентації, а також інкапсулює операції зв’язані з обробкою даних.

Тобто для кожного контролера існує свій сервіс, в якого кожному методу контролера відповідає метод бізнес логіки, що опрацьовує дані які отримав контролер, взаємодіє з ДАЛ (тобто фактично з БД), оброблює

дані, або створює нові, та відправляє результат роботи назад на контролер. Також може генерувати помилки, та валідувати дані.

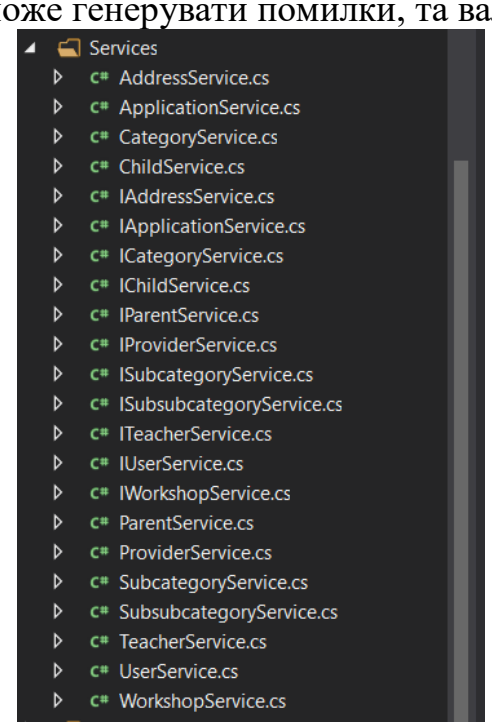


Рис.14 Business Logic Layer (Services Folder)

Робота Бізнес логіки тісно пов’язана з доступом до даних. Як було наведено раніше, доступ до даних має домені моделі сутностей, тоді як бізнес логіка та рівень представлення працюють безпосередньо з моделями DTO. Тому саме на цьому рівні потрібно використовувати перетворення одного в інше і навпаки (mapping).

Для цього були написані екстеншн методи для доменних та дто моделей використовуючи AutoMapper.

Приклад мапингу можна побачити на рис. 15 та рис.16.

```
public static Workshop ToDomain(this WorkshopDTO workshopDto)
{
    return Mapper<WorkshopDTO, Workshop>(workshopDto, cfg =>
    {
        cfg.CreateMap<WorkshopDTO, Workshop>();
        cfg.CreateMap<AddressDto, Address>();
        cfg.CreateMap<ProviderDto, Provider>();
        cfg.CreateMap<CategoryDTO, Category>();
        cfg.CreateMap<SubcategoryDTO, Subcategory>();
        cfg.CreateMap<SubsubcategoryDTO, Subsubcategory>();
        cfg.CreateMap<TeacherDTO, Teacher>();
    });
}
```

Рис.15 Мапінг DTO моделі в Domain модель

```
Ссылок: 7
public static WorkshopDTO ToModel(this Workshop workshop)
{
    return Mapper<Workshop, WorkshopDTO>(workshop, cfg =>
    {
        cfg.CreateMap<Workshop, WorkshopDTO>();
        cfg.CreateMap<Address, AddressDto>();
        cfg.CreateMap<Provider, ProviderDto>();
        cfg.CreateMap<Category, CategoryDTO>();
        cfg.CreateMap<Subcategory, SubcategoryDTO>();
        cfg.CreateMap<Subsubcategory, SubsubcategoryDTO>();
        cfg.CreateMap<Teacher, TeacherDTO>();
    });
}
```

Рис.16 Мапінг Domain моделі в DTO модель

Також приклад реалізації класу та інтерфейсу бізнес логіки можна побачити в Додатку Б.

2.5 Рівень представлення (PL)

Розглянемо Рівень представлення (презентації). Він складається з контролерів. Кожний контролер відповідає за окрему сутність. В кожному з контролерів фактично реалізовані операції CRUD (Create Read Update Delete) за допомогою методів типу Get, Post, Put, Delete.

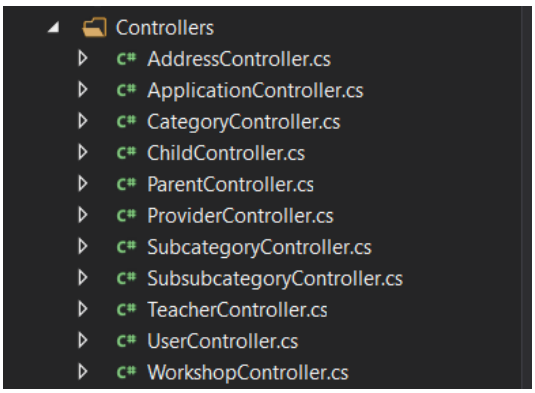


Рис.17 Структура PL

Мінімальний функціонал кожного з контролерів включає в себе методи:

- **[Get] GetEntities()** – для отримання всіх сутностей, що містяться в БД.
- **[Get] GetEntityById(long id)** – для отримання конкретної сутності з БД по її первинному ключу.
- **[Post] CreateEntity(Entity entityDto)** – для створення нової сутності, та запису її в БД.

- **[Put] UpdateEntity(Entity entityDto)** – для оновлення вже існуючої сутності в БД.
- **[Delete] DeleteEntity(long id)** – для видалення сутності з БД по її первинному ключу.

Рівень представлень ніяк не оброблює дані, лише первинно валідує їх, та хендлить помилки, що може надіслати бізнес логіка. Уся робота з даними передається у рівень бізнес логіки, рівень презентації просто очікує відповідь, та відправляє її назад користувачу.

Приклад реалізації контролера можна побачити в Додатку В.

2.6 Авторизація та аутентифікація

Для роботи з обліковими записами користувачів наш сервіс використовує Identity Server 4. Він доданий в наш солюшн як окремий проект, та являє собою MVC Server.

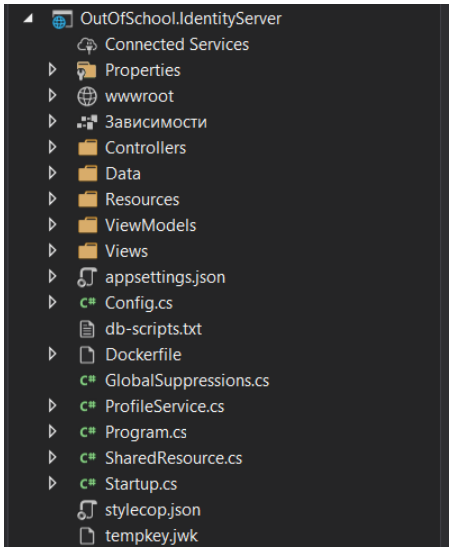


Рис.18 Структура Identity Server

Його роль полягає в тому, щоб віддати сторінку з параметрами реєстрації чи входу, отримати дані, та виконати відповідні дії.

Під час реєстрації створюється відповідний аккаунт з певною роллю (всього в нас зараз їх три: admin, parent, provider) та вноситься запис до БД, а також відправляється лист на пошту клієнта для підтвердження email.

При успішному вході генерується новий JWT token в якому шифруються основні дані такі як UserName, UserRole, ExpiredDate та ін., а також поле з ключем підтвердження, що згенерований за допомогою закритого ключа заданого в нашому проекті. JWT записується в cookie.

JWT записаний у користувача використовується в PL. Перш ніж користувач спробує виконати якісь дії, що потребують авторизації PL відправить токен нашому Identity Server на валідацію. В разі успішної валідації Identity поверне нам об’єкт юзера з усіма полями, після чого проводить перевірка доступу. Деякі методи доступні тільки для користувачів з певною роллю. Якщо наш юзер не відповідає цієї ролі – PL поверне 402 помилку, якщо ж виповідає – поверне результат виконання методу до якого ми зверталися.

```
[Authorize(Roles = "parent,admin")]
[HttpPost]
[Consumes(MediaTypeNames.Application.Json)]
```

Рис.19 Обмеження ролей та авторизація на методі

2.7 Модульне тестування (Unit Tests)

Юніт тестування – необхідна вимога у будь-якому серйозному проекті. Юніт тести повинні покривати не менше 80% функціоналу усіх публічних методів.

Юніт тесті в даному солюшині винесені в окремий проект. Структуру цього проекту можна побачити на Рис. 20.

Для написання тестів був використаний фреймворк NUnit. Також були використані Moq та InMemoryDatabase для симулювання роботи певних компонентів, та утворення чогось на кшталт БД на локальній машині для тестування.

Приклад реалізації можна побачити в Додатку Г.

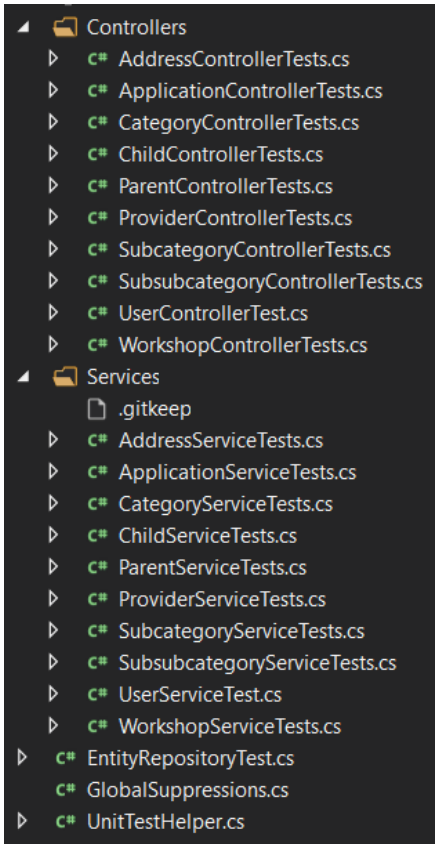


Рис.20 Структура проекту з юніт тестами

2.8 Документування, логування, локалізація, stylecop

Документування. Додатково, для орієнтації по проекту, для всіх публічних методів, полів та класів була додана XML документація. Приклад можна побачити на рис.21

```
/// <summary>
/// Get entity by it's key.
/// </summary>
/// <param name="id">Key in the table.</param>
/// <returns>Provider.</returns>
Ссылок: 7
Task<ProviderDto> GetById(long id);
```

Рис. 21 Документування методів та полів

Логування. Також, для аналізу роботи серверу було додано логування у файл. (Прописано в MiddleWare)

Всі методи сервісів (бізнес логіки) логують свої дії у відповідний файл за допомогою об’єкта logger інтерфейсу ILogger.

Приклад можна побачити на рис.22.

```
Ссылок: 5
public async Task<ProviderDto> Create(ProviderDto dto)
{
    logger.Information("Provider creating was started.");

    if (providerRepository.Exists(dto.ToDomain()))
    {
        throw new ArgumentException(localizer["There is already a provider with such a data."]);
    }

    Func<Task<Provider>> operation = async () => await providerRepository.Create(dto.ToDomain()).ConfigureAwait(false);

    var newProvider = await providerRepository.RunInTransaction(operation).ConfigureAwait(false);

    logger.Information($"Provider with Id = {newProvider?.Id} created successfully.");

    return newProvider.ToModel();
}
```

Рис.22 Метод сервісу, що використовує логування своїх дій

Локалізація. В проект була додана локалізація строк. Користувач може вказати мову, на якій хоче отримувати відповіді серверу, в своєму запиті. Якщо він цього не зробить – встановлюється дефолтна мова.

Для цього були додані відповідні сервіси в MiddleWare, а також файли з ресурсами та відповідні об’єкти інтерфейсу ILocalizer в проект. На разі підтримується англійська і українська.

```
if (providerRepository.Exists(dto.ToDomain()))
{
    throw new ArgumentException(localizer["There is already a provider with such a data."]);
}
```

Рис.23 Локалізація строк

Имя	Значение
Address is null.	Адреса null (пуста).
Id cannot be 0 or less than 0.	Id не може бути 0 або меншим за 0.
The id cannot be greater than number of table	Id не може бути більшим ніж кількість записів у таблиці.
The id cannot be less than 1.	Id не може бути меншим за 1.
There is already a provider with such a data.	Вже існує провайдер із подібною інформацією.

Рис.24 Файл ресурсів

Stylecop. До проекту був доданий пакет стайл коп, що генерує ворнінги при написанні “нечистого” коду. Тому код в проекті написаний з урахуванням всіх стилістичних правил.

```
private readonly IStringLocalizer<SharedResource> localizer;

/// <summary>
/// Initializes a new instance of the <see cref="Addre
/// </summary>
/// <param name="addressService">Service for Address m
/// <param name="localizer">Localizer.</param>
ссылка: 1
public AddressController(IAddressService addressServic
{
```

(Поле) readonly IStringLocalizer<SharedResource> AddressController.localizer

SA1214: Readonly fields should appear before non-readonly fields

Показать возможные решения (Alt+ВВОДилиCtrl+.)

Рис.25 Приклад ворнінгу від stylecop

2.9 Клієнтська частина

Клієнтська частина написана з використання технологій HTML, CSS, JS, Angular 12. В наш час дані технології є дуже популярними. Це дозволяє клієнтській частині бути підтримованою, розширюваною та документованою.

Структура проекту:

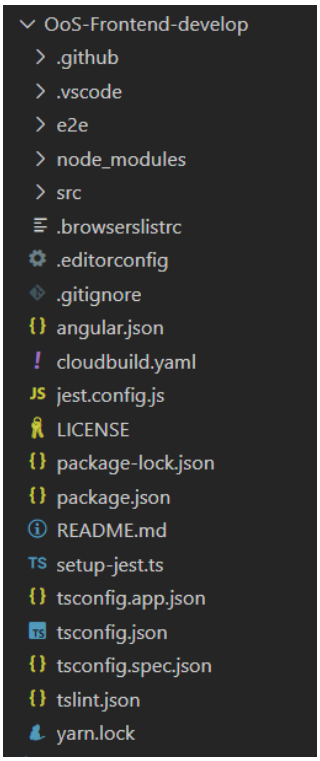


Рис.26 Структура Angular проекту

ВИСНОВКИ

Результатом даної роботи став соціальний проект, також було розібрано ключові етапи проектування додатку, проаналізовано проблеми, що траплялися під час розробки, були знайдені найефективніші рішення.

Опис роботи:

- Було спроектовано та розглянуто структуру бази даних, серверної частини та веб-клієнту.
- Було розглянуто та застосовано Best Practice для реалізації веб додатку
- Були вирішені проблеми на кшталт JWT Authorization, фільтрації, пагінації, локалізації та іншого.
- Було проведено документування проекту, та встановлення логування
- Було проведено тестування проекту за допомогою юніт тестів та вручну
- Була реалізована, протестована та задокументована клієнтська частина

В загальному можна зробити висновок, що тема сервісу гуртків є досить актуальною у наш час. На мій погляд, розроблений веб-додаток буде досить зручним для користувача після впровадження наведеного дизайну, також допоможе гурткам рости і знаходити клієнтів, батькам – знаходити найкращі гуртки за найкращими цінами, завжди бути в центрі новин пов’язаних з ними.

Отже, ще після декількох циклів розробки та тестування даний додаток може бути використаний як соціальний проект.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. *"What is Object/Relational Mapping?". Hibernate Overview. JBOSS Hibernate. Retrieved 19 April 2011.*
2. *"What is mash-up? - Definition from WhatIs.com". WhatIs.com. Retrieved 2015-11-04.*
3. *"Mashup Dashboard". ProgrammableWeb.com. 2009.*
4. *"An Online Platform for Web APIs and Service Mashups". IEEE Internet Computing. **12** (5). Sep–Oct 2008. doi:10.1109/MIC.2008.92.*
5. *"API Directory". ProgrammableWeb. Retrieved 2015-11-03.*
6. *Krill, Paul (20 July 2012). "Microsoft open-sources Entity Framework". InfoWorld. Retrieved 2012-07-24. 14*
7. *Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael (1996-08). Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. Wiley, August 1996. ISBN 978-0-471-95869-7. Retrieved from http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471958697.html. 16*
8. *Fowler, Martin "Patterns of Enterprise Application Architecture" (2002). Addison Wesley. 17*
9. *Martin, Robert C. (2000). "Design Principles and Design Patterns" (PDF). Archived from the original (PDF) on 2015-09-06. 21*
10. *"Single Responsibility Principle" (PDF). objectmentor.com. Archived from the original (PDF) on 2 February 2015. 22*
11. *Martin, Robert C. (2003). Agile Software Development, Principles, Patterns, and Practices. Prentice Hall. p. 95. ISBN 978-0135974445. 23*
12. *"Open/Closed Principle" (PDF). objectmentor.com. Archived from the original (PDF) on 5 September 2015. 24*

13. *"Liskov Substitution Principle" (PDF). objectmentor.com. Archived from the original (PDF) on 5 September 2015. 25*
14. *"Interface Segregation Principle" (PDF). objectmentor.com. 1996. Archived from the original (PDF) on 5 September 2015. 26*
15. *"Dependency Inversion Principle" (PDF). objectmentor.com. Archived from the original (PDF) on 5 September 2015. 27*
16. *The Routledge Dictionary of Modern American Slang and Unconventional English, Tom Dalzell, 2009, 1104 pages, p.595, webpage: BGoogle-5F: notes U.S. Navy "Project KISS" of 1960, headed by Rear Admiral Paul D. Stroop, Chicago Daily Tribune, p.43, 4 December 1960. 28*
17. *The Concise New Partridge Dictionary of Slang, Eric Partridge, Tom Dalzell, Terry Victor, Psychology Press, 2007, p.384. 29*
18. *Clarence Leonard (Kelly) Johnson 1910–1990: A Biographical Memoir (PDF), by Ben R. Rich, 1995, National Academies Press, Washington, DC, p. 13. 30*
19. *Foote, Steven (2014). Learning to Program. Addison-Wesley Professional. p. 336. ISBN 9780133795226. 31*
20. *Hunt, Andrew; Thomas, David (1999). The Pragmatic Programmer : From Journeyman to Master (1 ed.). USA: Addison-Wesley. pp. 320. ISBN 978-0201616224. 32*
21. *Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. p. 75. ISBN 978-0-470-04212-0. 33*
22. *Hamill, Paul (2004). Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media, Inc. ISBN 9780596552817. 34*

ДОДАТОК А

Repository Interface:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.DataAccess/Repository/IEntityRepository.cs

Repository Implementation:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.DataAccess/Repository/EntityRepository.cs

ДОДАТОК Б

Service Interface:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.WebApi/Services/IParentService.cs

Service implementation:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.WebApi/Services/ParentService.cs

ДОДАТОК В

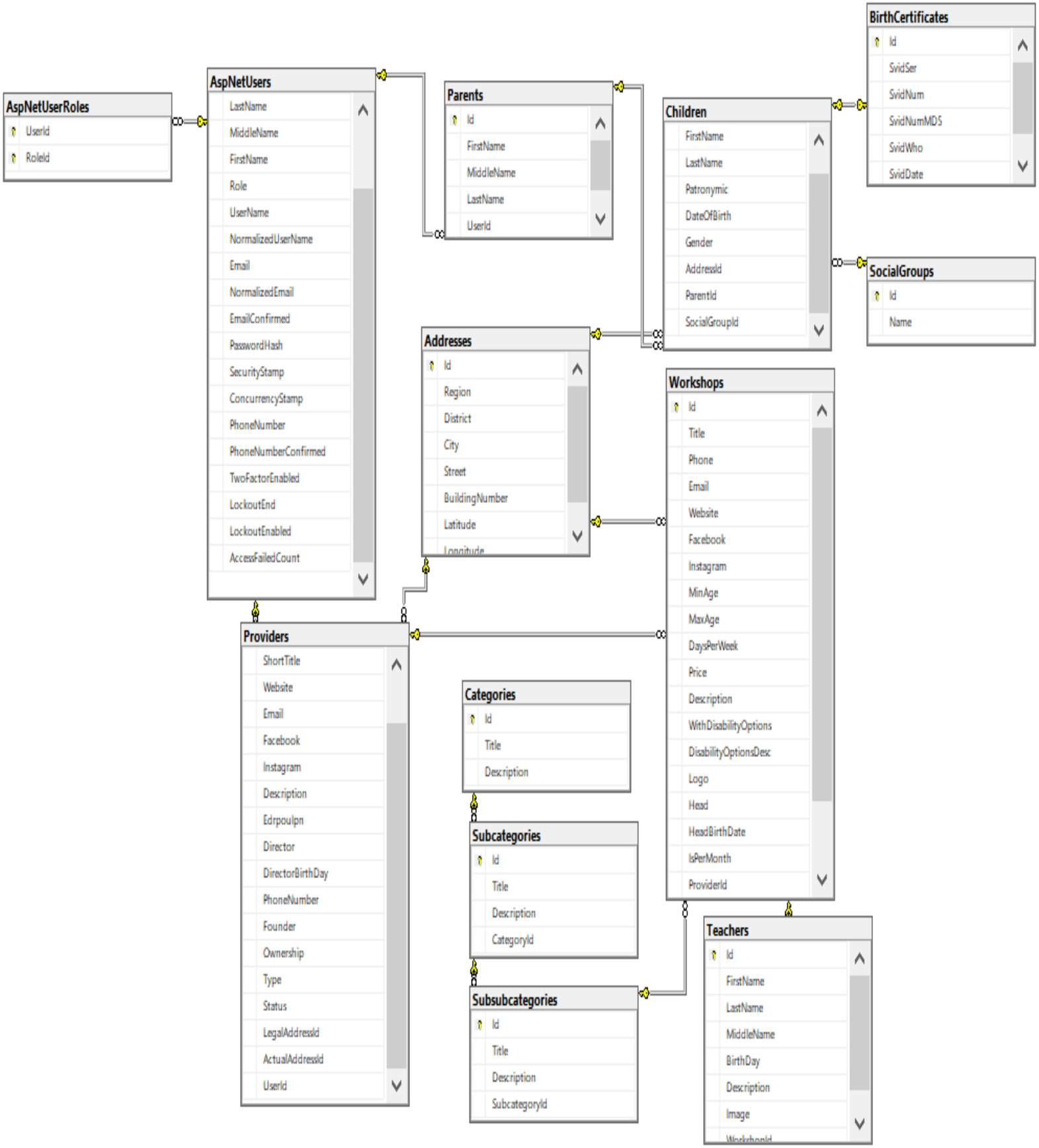
Controller:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.WebApi/Controllers/ParentController.cs

ДОДАТОК Г

ServiceTests:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.WebApi.Tests/Services/ParentServiceTests.cs

ControllerTest:
https://github.com/ita-social-projects/OoS-Backend/blob/develop/OutOfSchool/OutOfSchool.WebApi.Tests/Controllers/ParentControllerTests.cs

ДОДАТОК Д


AspNetUserRoles
UserId
RoleId
AspNetUsers
LastName
MiddleName
FirstName
Role
UserName
NormalizedUserName
Email
NormalizedEmail
EmailConfirmed
PasswordHash
SecurityStamp
ConcurrencyStamp
PhoneNumber
PhoneNumberConfirmed
TwoFactorEnabled
LockoutEnd
LockoutEnabled
AccessFailedCount
Parents
Id
FirstName
MiddleName
LastName
UserId
Children
FirstName
LastName
Patronymic
DateOfBirth
Gender
AddressId
ParentId
SocialGroupId
BirthCertificates
Id
SvidSer
SvidNum
SvidNumMD5
SvidWho
SvidDate
SocialGroups
Id
Name
Addresses
Id
Region
District
City
Street
BuildingNumber
Latitude
Longitude
Workshops
Id
Title
Phone
Email
Website
Facebook
Instagram
MinAge
MaxAge
DaysPerWeek
Price
Description
WithDisabilityOptions
DisabilityOptionsDesc
Logo
Head
HeadBirthDate
IsPerMonth
ProviderId
Providers
ShortTitle
Website
Email
Facebook
Instagram
Description
Edrpoulpn
Director
DirectorBirthDay
PhoneNumber
Founder
Ownership
Type
Status
LegalAddressId
ActualAddressId
UserId
Categories
Id
Title
Description
Subcategories
Id
Title
Description
CategoryId
Subsubcategories
Id
Title
Description
SubcategoryId
Teachers
Id
FirstName
LastName
MiddleName
BirthDay
Description
Image

To see more Details:

Front-End:

https://github.com/ita-social-projects/OoS-Frontend

Back-End:

https://github.com/ita-social-projects/OoS-Backend