КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

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

РОЗРОБКА ВЕБ СЕРВІСУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ GRAPHQL

Виконав студент 4-го курсу
Кльоз Віталій Вікторович _____
(підпис)

Науковий керівник:
Доцент, кандидат фіз.-мат. наук
Кузенко Володимир Федорович _____
(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 теорії та технології програмування.
«_____»_____ 2021 р,
протокол №_____
Завідувач кафедри
Нікітченко М.С. _____
(підпис)

Київ - 2021

ЗМІСТ

РЕФЕРАТ

Дипломна робота складається із вступу, чотирьох розділів, інструкції користувача, висновків, списку використаних джерел (XX найменувань) та 1 додатку.

Робота містить 30 рисунків та 1 таблиці. Загальний обсяг роботи становить 41 сторінка, основний текст роботи викладено на 34 сторінках.

Ключові слова: ПОШУК ТАНЦЮВАЛЬНИХ ПАРТНЕРІВ, ТРЕНЕРІВ, ШКІЛ, СПОРТИВНО БАЛЬНІ ТАНЦІ, ВЕБ СЕРВІС, GRAPHQL, NODE JS, WIX VELO

Реферат. В роботі досліджено проблеми комунікації в танцювальному спорті та проаналізовано недоліки існуючих систем, що працюють з танцівниками. Сформульовано задачі, які має вирішувати функціонал нової системи. Розроблено веб сервіс Dance Bright, використовуючи технологію GraphQL на серверній частині та Wix Velo на клієнстьку. В результаті отримано унікальну та ефективну платформу для пошуку танцювальних партнерів, тренерів та шкіл, а також відгуків про них.

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

БД - База даних

ПЗ - Програмне забезпечення

API - Application programming interface

DAO - Data access object

CRUD - Create, read, update, delete

ID - Identity document

ВСТУП

Оцінка сучасного стану об'єкта дослідження. Пандемія короновірусної хвороби трансформувала поведінку українців і прискорила неминучу цифровізацію в країні [1]. Протягом 2020-2021 року ми можемо спостерігати за стрімкою диджиталізацією в абсолютно усіх сферах економічного та суспільного життя: під час вимушеної ізоляції люди почали більше часу проводити в онлайн - просторі, навчились працювати, користуватись електронними послугами та мобільними застосунками, зросла роль соціальних мереж тощо.

«Велике перезавантаження» почалось! Для когось воно закінчиться втратою бізнесу, для когось – перепрофілюванням, а хтось відкриє абсолютно нові можливості і скористається ними. У цей період важливо мати бажання змінюватись та проявляти гнучкість. Міжнародне агентство реклами та PR Dentsu Aegis Network [2] під час проведення опитування визначило, що 52% респондентів чекають закінчення карантину, щоб знайти нове хобі і присвятити йому час. 34% опитаних мріють зайнятись танцями.

Танці – це дієва і доступна арт-терапія. Зібрані дослідниками дані показують, що саме танець допомагає людям пізнавати себе, розвиває емоційний інтелект, тримає людину в тонусі [3]. Для сучасної людини така активність є життєво необхідною, адже дозволяє зняти стрес і напругу, пережити складні емоції, відновити спокій і врівноваженість. Крім того, заняття танцями допомагає соціалізуватися, що особливо актуально в умовах пост-пандемії. Через взаємодію з партнером в парних танцях людина розвиває емпатію та вміння відчувати людину. Навички, набуті в танцях, допомагають і в особистому житті, і в роботі.

Спортивні бальні танці - особливий вид танцю, який поєднує у собі як і вишукане мистецтво, так і справжній спорт. Такий вид дозвілля приваблює людей абсолютно різного віку, статі, національності тощо. Для когось це спосіб гарно провести час, хтось обожнює шалені емоції, а комусь хочеться стати чемпіоном. І оскільки це парний і досить непростий вид танцю, на шляху до

своєї мети танцівникам потрібно поряд мати наполегливого та відкритого партнера, а також сильного тренера. Саме від них залежатиме як і прогрес, так і задоволення від танцю. Проте, знайти таких людей не завжди буває просто. Якщо немає потенційних партнерів серед знайомих, пошуки можуть тривати навіть роками, особливо для дівчат, яких у танцях переважна більшість [4]. А від неправильно вибраного тренера, який не вміє правильно підтримувати та надихати, можна взагалі втратити запал до танців.

Актуальність роботи та підстави для її виконання. З проблемою підбору партнерів та тренерів зустрічаються дуже багато танцівників різного рівня. У кожного є свої можливості, потреби та цілі, і щоб зробити правильний вибір потрібно познайомитись з великою кількістю різних людей. Для вирішення потрібні інтернет-ресурси, які сприяли б ефективній комунікації серед танцівників, були зручними та могли охопити велику аудиторію. Існуючі ресурси з домену танцювального спорту є або недостатньо зручними, або їм бракує певного ключового функціоналу, через це проблема є досі актуальною.

Мета роботи. Метою цієї роботи - є розглянути проблему інтернет-комунікації в танцювальному спорті, проаналізувати існуючі ресурси та потреби танцівників і створити систему, яка була б зручною, повною та значно спростила б підбір партнерів, тренерів та шкіл.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ НА РИНКУ СИСТЕМ

1.1 Танцювальний портал DanceSport

DanceSport.net.ua - це український танцювальний портал, який представляє такі функції, як пошук партнерів, інформацію про попередні та майбутні турніри, новини та статті з танцювального спорту, оголошення, вакансії тощо [5]. Згідно зі статистикою Рамблер [6], сайт щомісяця відвідує понад 9500 унікальних користувачів.

Найбільш використованою функцією цього порталу є пошук партнерів (рис. 1). На сайті представлено зручну систему фільтрації за такими критеріями, як зріст, рік народження, стать, танцювальний клас латинської та стандартної програм та місто.

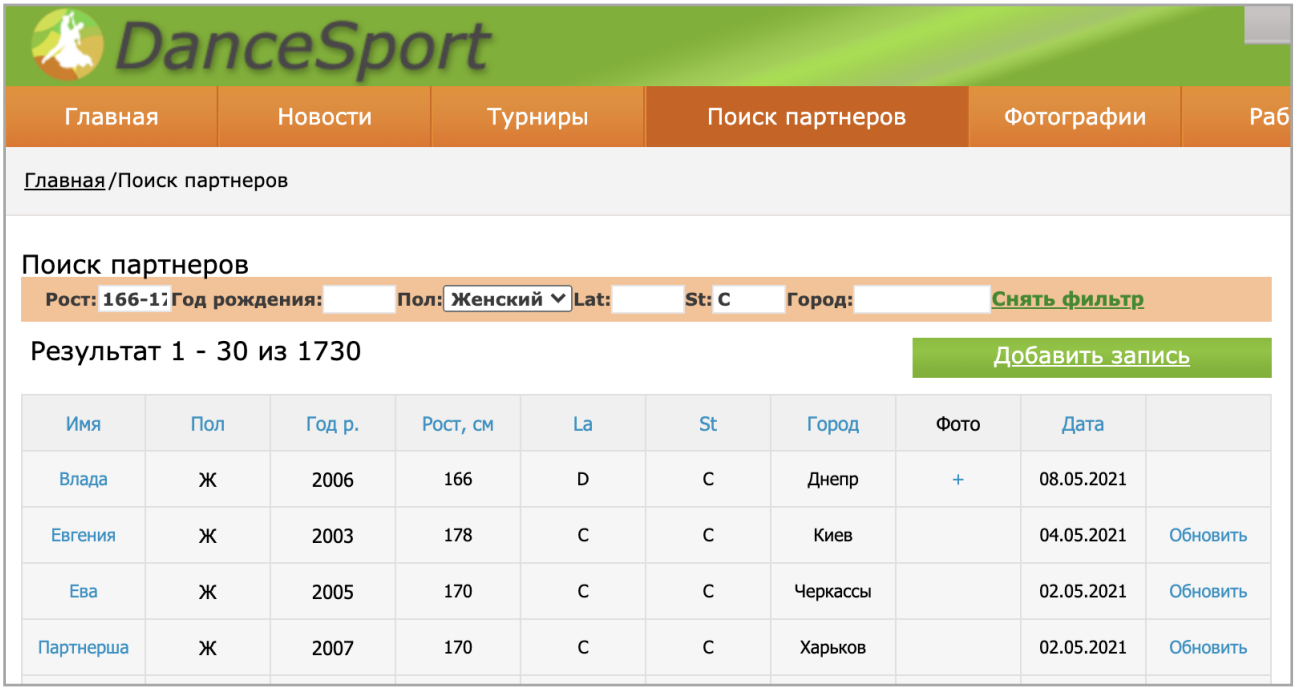

Рис. 1. Сторінка пошуку партнерів на dancesport.net.ua

Створення оголошень відбувається без авторизації, після натискання кнопки “Добавить запись” та введення необхідної інформації. Щоб побачити більш детальну інформацію та контактні дані необхідно перейти на сторінку анкети (рис. 2), натиснути на ім’я.

Для того, щоб підтримувати своє оголошення вгорі списку потрібно час від часу натискати кнопку “Обновить”, після чого воно автоматично буде

перенесено на початок. У тому випадку, коли людина вже знайшла партнера і перестала оновлювати оголошення, з часом воно спуститься донизу, проте системою не передбачена можливість видалити свою анкету. Через що персональні дані залишаються в загальному доступі на необмежений час.

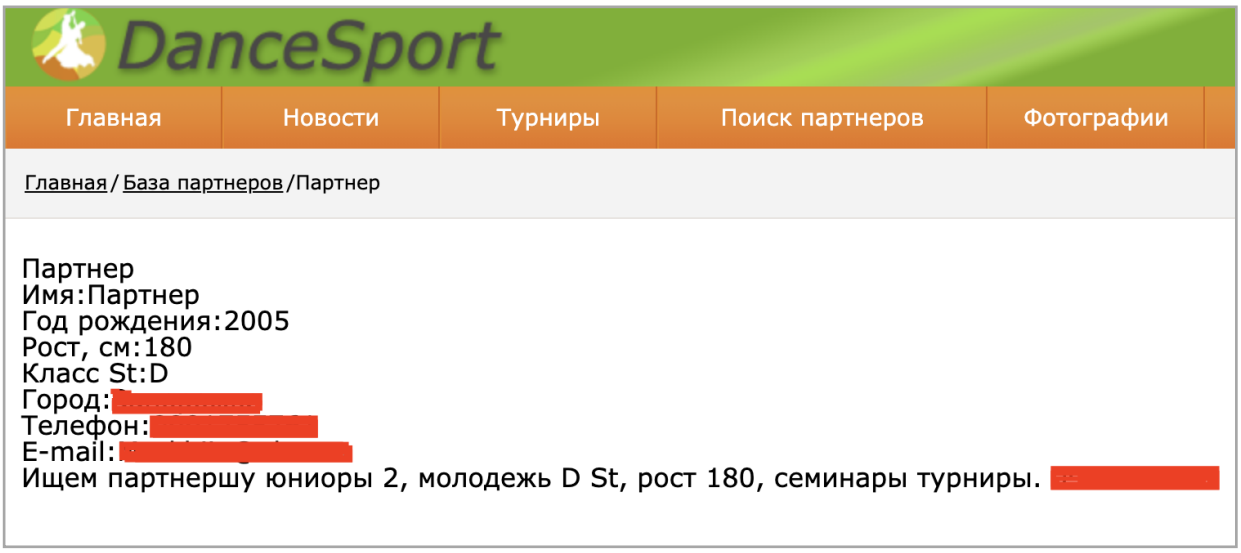

Рис. 2. Сторінка анкети на dancesport.net.ua

Не зважаючи на функцію оновлення оголошення, система не обмежує в кількості створених анкет, і як показано на скриншоті (рис. 3), деякі люди зловживають та заспамлюють список дублюючими анкетами.

Алина	Ж	2005	169	C		Киев		04.05.2021	Обновить
Лиза	Ж	2006	157	D	D	Киев	+	03.05.2021	Обновить
Лиза	Ж	2006	157	D	D	Киев	+	03.05.2021	Обновить
Лиза	Ж	2006	157	D	D	Киев	+	03.05.2021	Обновить
Лиза	Ж	2006	157	D	D	Киев	+	03.05.2021	Обновить

Рис. 3. Сторінка пошуку партнерів з дублюючими анкетами на dancesport.net.ua

Також в системі відсутня авторизація (рис. 4), а тому обмежити кількість створених оголошень неможливо за дизайном.

Рис. 4. Помилка при натисканні кнопок авторизації на dancesport.net.ua

Аналогічно з анкетами танцівників, в системі DanceSport реалізований зручний пошук турнірів (рис. 5), а також є окремі вкладки з корисною інформацією тощо.

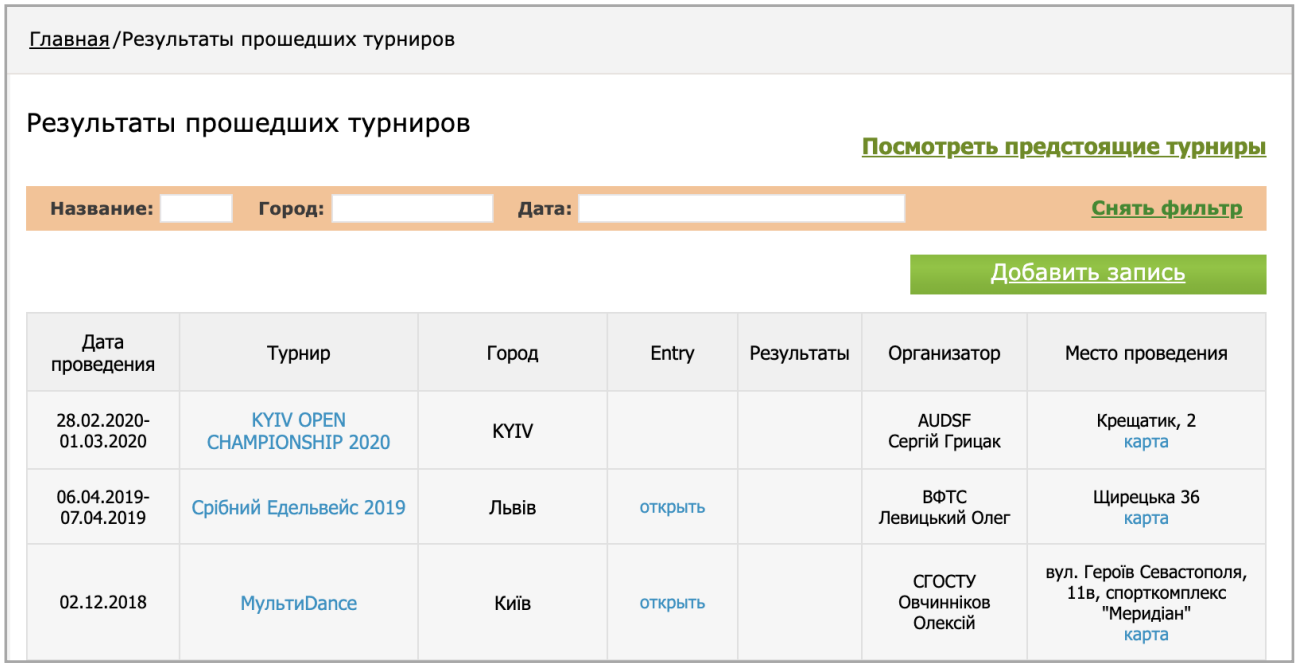

Рис. 5. Сторінка пошуку турнірів на dancesport.net.ua

Проте, в системі немає не зберігається інформація про танцювальні клуби, танцівників чи тренерів.

1.2 Танцювальний портал FlyMark

Flymark.com.ua - це український танцювальний портал, який зберігає дані про танцівників, клуби, турніри, і відображає оголошення та корисну інформацію [7]. Система також представляє технічну підтримку при проведенні танцювальних турнірів, після чого відображає їх результати на сторінках відповідних танцівників.

Flymark дозволяє шукати танцювальні школи лише по таких критеріях, як назва та місто (рис. 6), і на надає можливості відображати яку-небудь додаткову інформацію, наприклад адресу.

Пошук партнерів можливий лише по критеріях місто та стать (рис. 7), тому при високому попиті та великої кількості оголошень пошук є не дуже зручним.

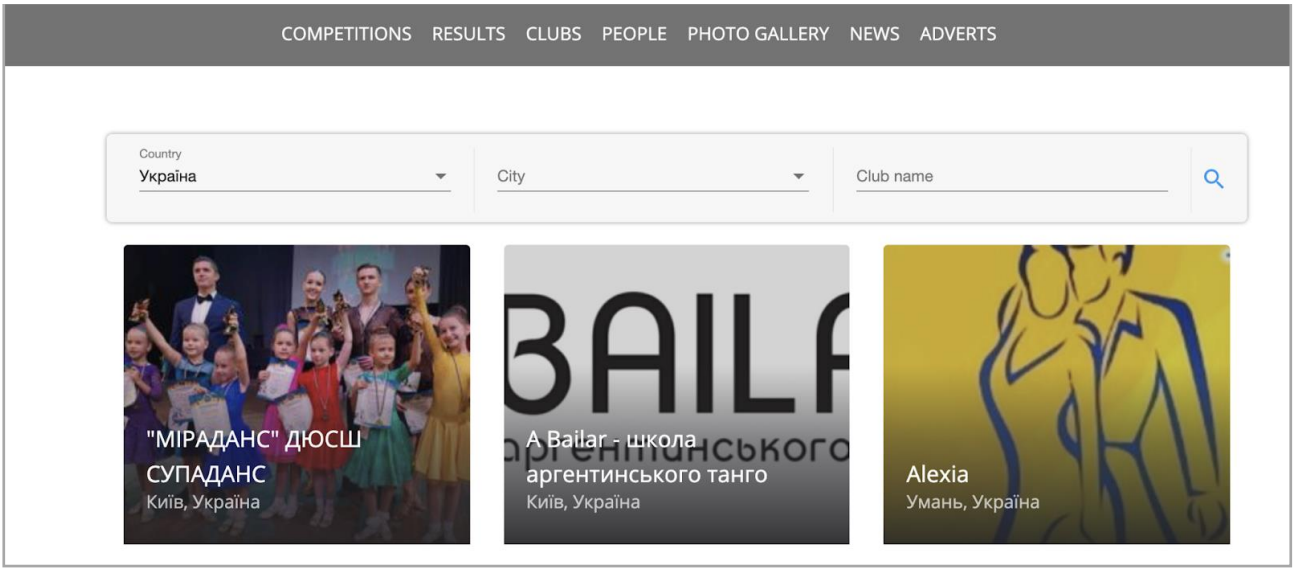

Рис. 6. Сторінка пошуку турнірів на flymark.com.ua

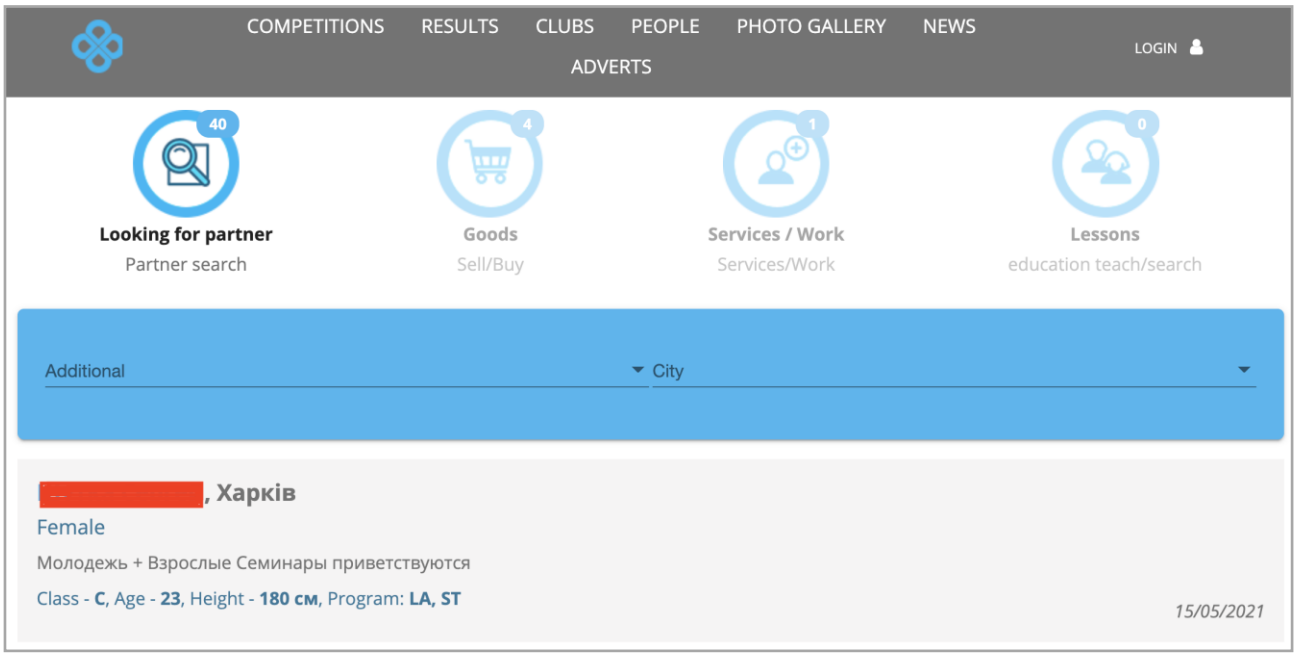

Рис. 7. Сторінка пошуку партнерів на flymark.com.ua

Проте, навідміну від dancesport.net.ua, Flymark має авторизацію, тому створення оголошень є захищеним від зловживання.

РОЗДІЛ 2. ВИМОГИ ДО СИСТЕМИ

2.1 Пошук партнерів

Після огляду декількох існуючих оголошень по пошуку партнерів, таких як на рис. 8, виділено найголовніші дані, які повинні відображатись в анкетах розроблюваної системи:

1. Дані про танцівника: ім’я, стать, рік народження, зріст, танцювальний клас стандартної та латино-американської програми, місто проживання, відвідувані танцювальні школи, контактні дані.
2. Критерії потенційного партнера: діапазон року народження, діапазон зросту, діапазон танцювального класу стандартної та латино-американської програм, можливі міста проживання, можливі танцювальні школи, можливість переходу в інший клубтмінімальні та максимальні рік народження, зріст, танцювальний клас стандартної латино-американської програми, діапазон бажаної кількості годин тренувань у тиждень.

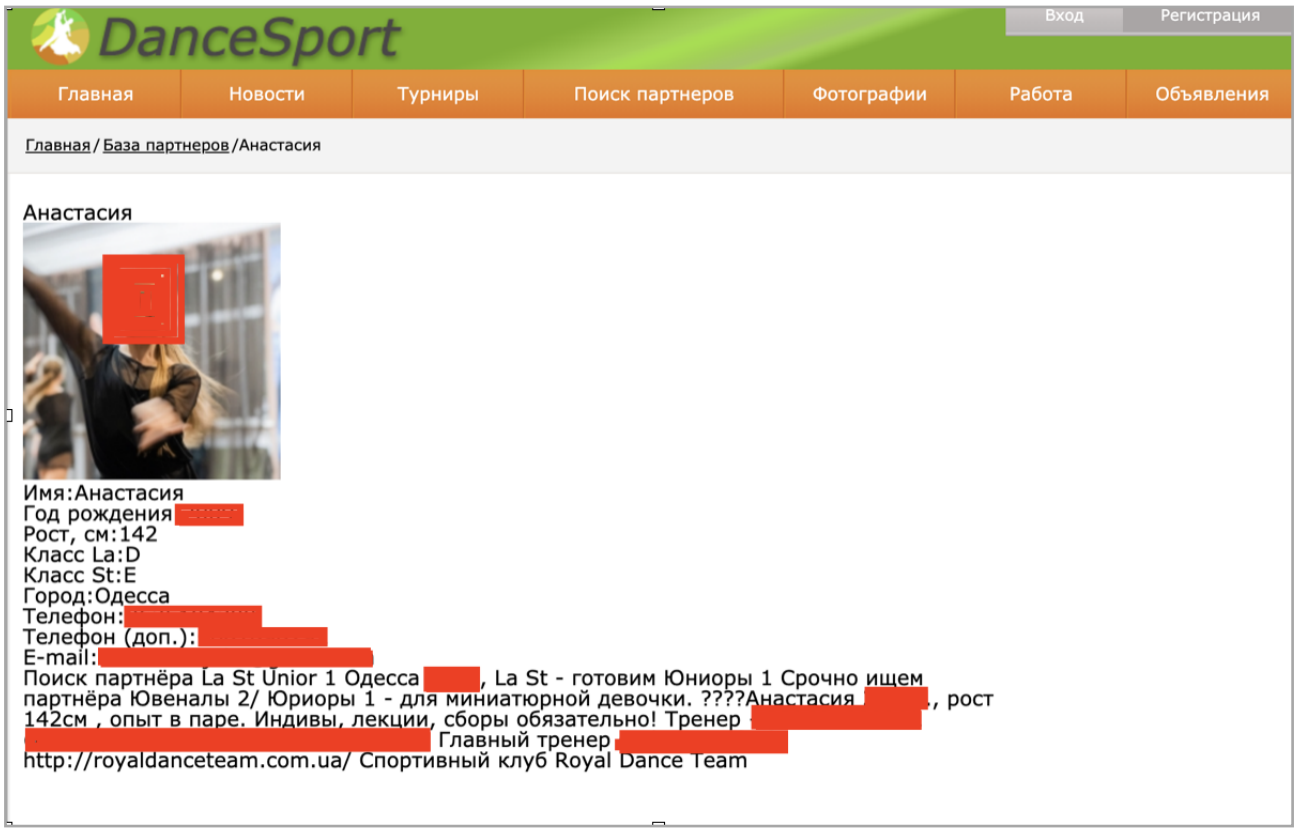

Рис. 8. Сторінка з прикладом анкети на dancesport.net.ua

Також, на сторінці повинна працювати фільтрація по заданих критеріях.

2.2 Сторінка танцівника (тренера)

На сторінкі танцівника (тренера) повинна відображатись вся та ж інформація, що була описана в п.2.1.1, а також:

1. Якщо має партнера, то також посилання на його сторінку, вікова категорія пари та дата початку спільного танцювання.
2. Якщо є тренером, то також відгуки та середню оцінку.

Система повинна мати наступні зв’язки між танцівниками (тренерами) та школами:

1. Одна школа може мати жодного або декілька танцівників
2. Одна школа може мати жодного або декілька тренерів
3. Один тренер може мати жоден або декілька відгуків
4. Одна школа може мати жоден або декілька відгуків
5. Один танцівник може мати жодного або одне оголошення про пошук партнера

2.3 Сторінка школи

На сторінці школи повинна відображатись наступна інформація: назва, адреси (якщо декілька), контактні дані, список танцівників та тренерів, фотографії, відгуки та середня оцінка.

2.4 Сторінка користувача

На сторінці користувача повинно бути можливість додати дані про себе, відповідно до типу користувача, змінити та видалити їх за бажанням.

РОЗДІЛ 3. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ТА СИСТЕМ

3.1 Мова запитів і маніпуляції з даними GraphQL

GraphQL - це мова запитів та операцій, що використовується для комунікації клінтських додатків з серверними. GraphQL можна використовувати з будь-якими мовами програмуваннями, оскільки вона описує дані, а вже розробнику потрібно написати код, який буде виконувати необхідні операції.

Сервіс з використанням GraphQL створюється шляхом визначення типів і полів для цих типів, а потім надання функцій для кожного поля відповідного. Операції поділяються на запити, що дістають дані, та мутації, що змінюють дані.

На рис. 9 наведено приклад запиту me, який повертає дані про поточного користувача, мутація changeName, яка дозволяє змінити ім’я поточного користувача, та тип даних User, в якому описується які поля має User.

```
type Query {
  me: User
}
type Mutation {
  changeName(newName: String): User
}
type User {
  id: ID
  name: String
}
```

Рис. 9. Приклад схеми GraphQL

Під час роботи, сервіс, який працює з GraphQL, перевіряє вхідні запити на коректність та відповідність типам та делегує їх резовлерам - функціям, які пишуться розробниками і визначають які дані потрібно звідки брати.

Знак оклику “!” Після назви типу означає, що це значення не може бути null і його повернення є гарантованим.

3.2 Apollo GraphQL

Apollo GraphQL - клієнтська та серверна бібліотеки для роботи з GraphQL [8]. Ця платформа значно спрощує роботу з GraphQL, завдяки виконанню наступних функцій:

1. Генерація типів за схемою
2. Генерація функцій-резолверів для запитів та мутацій за схемою
3. Валідація запитів
4. Обробка клієнтських запитів
5. Побудова запитів до сервера
6. Моніторинг часу, який витрачається на обробку різних частин запитів

Дані бібліотеки мають підтримку мов javascript та typescript.

3.3 Dataloader

DataLoader - утиліта для роботи з великою кількість операцій на отримання даних програми, яка будує взаємодію даних таким чином, щоб забезпечити спрощений та послідовний API для різних віддалених джерел даних, таких як бази даних або сервіси, за допомогою пакетних запитів та кешування [9].

DataLoader тісно використовується разом з технологією GraphQL і дозволяє забезпечити мінімальну кількість операцій для отримання всіх необхідних даних, без дублювання. На перший погляд може здатись, що аналогічна задача виконується за допомогою кешування - тимчасово зберігання певних частовикористовуваних даних в локальній (швидкій до доступу) пам’яті. Проте DataLoader працює ще ефективніше, оскільки такий підхід дозволяє агрегувати запити, які відбуваються майже одночасно. Для цього утиліта зберігає на короткий час всі однотипні та однакові запити, і тоді за потреби агрегує та виконує дублюючі запити лише один раз.

Побудова таких даталоадерів поділяється на такі частини:

1. Агрегація запитів - групування однотипних і/або ідентичних запитів за певними ключами

2. Виконання асинхронних операцій по згрупованих запитах

3. Передача результатів на запити

Для зображення принципу роботу розглянемо приклад з системою, в якій є університетські публікації, які в собі мають список авторів. Нехай нам потрібно виконати такий GraphQL запит, зображений на рис. 10.

```
{
  Publications (age >= 30) {
    name
    authors{
      name
      age
      countryOfStudy
    }
}
```

Рис. 10. Приклад запиту GraphQL

Для отримання заданої інформації у нас є доступ до серверу бібліотеки, який може одним запитом дати список всі публікацій університету з іменами авторів, а також доступ до серверу соціальної мережі, де можна по імені людини за один запит дізнатись його персональні дані.

Виходить, що нам потрібно зробити один запит на сервер бібліотеки університету, щоб отримати дані про публікації. Далі ми їх фільтруємо. Після того, для кожного унікального викладача потрібно зробити запит на соціальну мережу, а оскільки запити робляться по ключу “ім’я”, після налаштування Dataloader, всі запити автоматично виконаються лише по унікальних іменах, що нам і потрібно.

3.4 JavaScript, Typescript

JavaScript - інтерпретована мова програмування викого рівня, яка була створена для роботи у веб браузерах, проте зараз широко використовується також в розробці серверних та інших типів програмного забезпечення [10].

Разом з HTML та CSS, Javascript - одна з ключових технологій World Wide Web. Згідно з статистикою від w3techs.com [11], Javascript використовується в понад 97% сайтах світу.

TypeScript - це мова програмування, побудована на основі Javascript, яка додає статичні типи. Тим самим, це дозволяє описувати та валідувати об'єкти, а також писати більш зрозумілий та безпечний код, тим сами значно розширюючи сферу застосування мови Javascript.

TypeScript код трансформується в JavaScript за допомогою TypeScript компілятора або Babel, тим самим генеруючи простий, чистий код, яким можна запускати всюди, де працює JavaScript - в браузері, NodeJS або інших застосунках. А схожість мов дозволяє з легкістю навчитись одній, якщо вже знаєш другу.

3.5 Node JS

Node.js - платформа для розробки серверних застосунків на Javascript, що дозволяє запускати їх поза браузером [12]. Найголовнішою функцією є можливість створювати та використовувати бібліотеки інших розробників.

3.6 База даних Mongo

MongoDB - кросплатформенна документ-орієнтована база даних, що класифікується як NoSQL [13]. Працює з JSON-об'єктами та необов'язковими типами даних, що дає гнучкість у типізації та збереженні даних різного формату.

Попри те, що в MongoDB відсутня строга типізація, яка знайома SQL базам даних, вона дає широкий функціонал для роботи, маніпуляції з даними, а

також можливість налаштовувати індекси колонок, що пришвидшують запити над ключовими типами значень.

Для роботи з базою на платформі NodeJS використовується NPM бібліотека mongodb, яка представляє зручний інтерфейс для роботи з повним функціоналом даної БД.

Також, для невеликих проектів, команда MongoDB надає в безкоштовне користування свої хостинги з базами даних, що є дуже зручним для ознайомлення з платфомою.

3.7 Конструктор сайтів Wix, платформа Velo

Wix.com - передова платформа для створення професійних HTML5 сайтів для Web та Mobile [14]. На вибір представлено сотні різноманітних готових шаблонів, а зміна будь-яких візуальнихи елементів здійснюється за допомогою інтуїтивних drag-n-drop панелей, через що розробка не потребує технічних знань html та css.

Користувачі можуть додавати на свій сайт різноманітні плагіни, електронну комерцію, маркетинг, контакт форми, форум, блог та багато інших елементів, створеними Wix або сторонніми розробниками. Сам редактор (рис. 11) побудований на безкоштовній бізнес моделі, надаючи преміум-підписки з професійним функціоналом, таким як власний домент, інтернет оплата тощо.

Wix.com підходить як для малого бізнесу, ресторанів, артистів чи фотографів, так і для середнього - за рахунок наявності платформи для роботи з роширеною функціональністю, можливістю створювати власні інтеракції та інтеграції - **Velo by Wix** [15]. Серед її можливостей:

1. Написання власного JavaScript коду на Wix сайті та можливість працювати з існуючими API та елементами на сайті.
2. Вбудовані бази даних, які дозволяють додавати на працювати з великою кількістю даних, а також використовувати Wix Apps додатки, що потребують збереження даних.

3. Відкритість платформи, що дозволяє використовувати існуючі NPM пакети, робити запити на сторонні API, а також самим надавати API для інших додатків через HTTP функції.

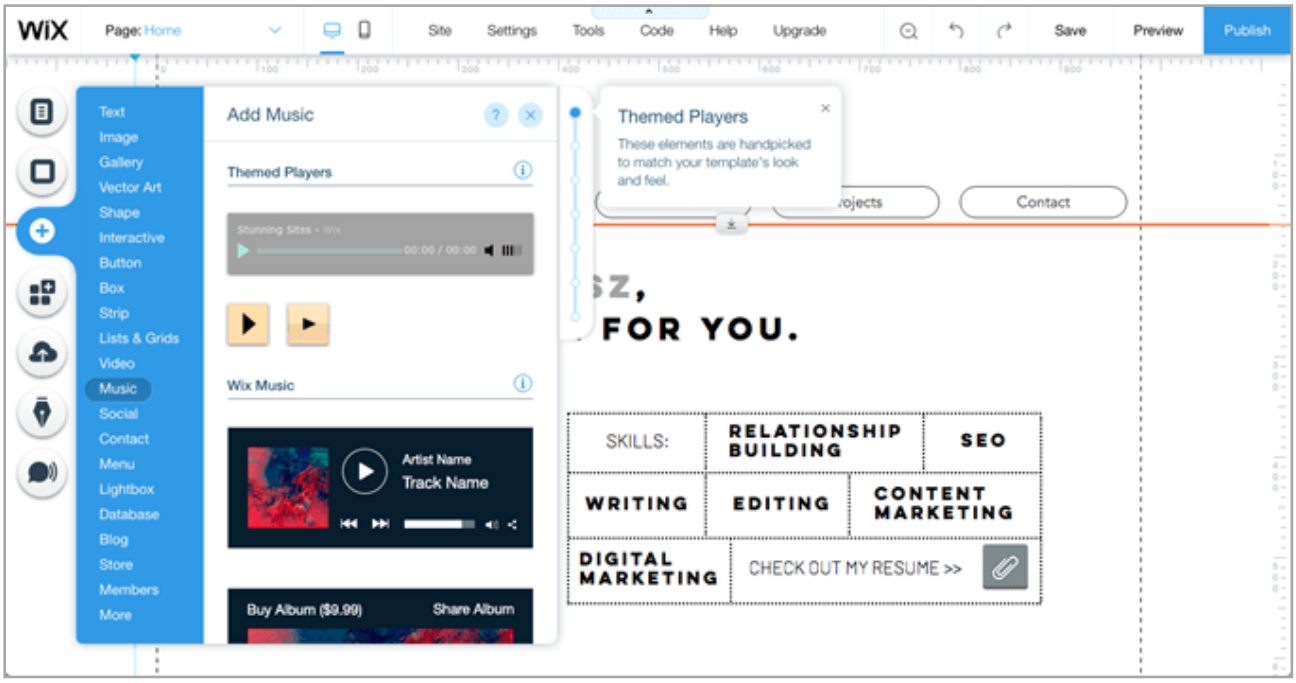

Рис. 11. Зображення редактора сайтів на wix.com

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Вибір оптимальних технологій для реалізації

Для того, щоб система задовольняла вимоги, описані в розділі 2, а також могла допрацьовуватись бажаючими без великого багажу знань технологій, для її реалізації потрібно обрати технології, для яких найкраще виконуються наступні вимоги:

- Розширюваність моделі даних
- Розширюваність функціоналу
- Ефективність, низька ресурсозатратність
- Простота, низький "поріг входження" для нових розробників

Для того, щоб вибрати необхідні технології, потрібно вирішити наступні питання:

1. Як зберігати дані?
2. Як передавати дані на клієнт?
3. Як ефективно розробляти клієнтську та серверну частину?

Розглянемо кожне з цих запитань, щоб прийти до необхідного рішення:

4.1.1 Збереження даних

Система повинна бути здатною зберігати та оперувати з великою кількість одиниць даних різного типу: танцівників, тренерів, шкіл тощо. Більшість операцій до бази даних будуть виконуватись на зчитування, інші - на запис. Зчитування повинно виконуватись швидко, для запису допускається невелика затримка - це нормально, якщо користувачі побачать нову анкету лише через пару хвилин.

Для таких цілей потрібна база даних - реляційна або нереляційна. Оскільки наперед невідомо які додаткові дані будуть додаватись у майбутньому, для гнучкості та простоти використання було обрано NoSQL базу даних - MongoDB.

4.1.2 Обмін даними між користувачем та системою

У створенні WEB застосунків домінуючою є клієнт серверна модель [16], яка полягає в розподілі застосунків на 2 компоненти:

1. Сервери, що надають певну інформацію та дозволяють керувати нею
2. Клієнти, що звертаються до серверів

Клієнти незалежні між собою, та для зв'язку з серверами використовують один з наперед визначених типів обміну даними, такими як, наприклад [17]:

1. Sockets
2. Pipes
3. RPC
4. REST
5. GRAPHQL

Кожен має свої переваги та недоліки, і в даній роботі було обрано GraphQL завдяки його гнучкості при роботі з широким спектром даних, зручності в розширенні та швидкодію за рахунок опису клієнтом даних, які йому потрібні. Останнє досягається завдяки можливості агрегувати декілька запитів в одному, наприклад, дані відразу по танцівниках, тренерах та школах.

Сервер відповідає за зберігання, видачу та обробку даних, а клієнт - за зручне користуванням сервером для кінцевого користувача, за рахунок побудови адаптованого інтерфейсу з текстовими полями, картинками, кнопками тощо (фронт-енду).

4.1.2 Ефективна розробка клієнта та сервера

Для того, щоб зменшити вартість підтримки системи у майбутньому, було вирішено обрати схожі технології для серверної та клієнтської частини. Таким чином одна людина зможе працювати відразу з усією системою.

Для серверної частини обрано мову програмування Typescript та платформу NodeJs, коли для клієнтської - мову Javascript та платформу Wix Velo. Обидві мови є дуже схожими, оскільки перша по факту компілюється в другу, а платформи працюють схожим чином та обоє підтримують роботу з NPM пакетами, що дозволяє при потребі використовувати одні й ті ж сторонні бібліотеки.

Крім того, використання платформи Wix Velo дозволяє не лише зручно працювати з кодом, але і редагувати дизайн фронт-енду без використання Javascript - за рахунок інтуїтивного інтерфейсу. Це дозволяє змінювати зовнішній вигляд системи для кінцевого користувача навіть без знання програмування, що значно спрощую її підтримку.

4.2 Створення моделі даних

Перед реалізацією системи необхідно створити GraphQL схему, по якій клієнтські застосунки будуть розуміти якими даними та функціями оперує наша система.

При дизайні API потрібно абстрагуватись від уявлення про те, як ці дані будуть зберігатись в базі даних, адже моделі для API та для зберігання не обов'язково повинні бути однаковими, згідно з рекомендаціями до архітектури ПЗ [18].

З точки зору користувача системи в нас є наступні типи ролей та об'єктів:

1. Користувач системи - зареєстрований аккаунт з мінімальною інформацією про себе, який ще не є танцівником
2. Користувач-танцівник - зареєстрований акаунт з даними про себе, характеристиками, танцювальним класом, нагородами тощо. Може танцювати в кількох танцювальних школах.
3. Користувач-тренер - зареєстрований аккаунт з даними про себе, характеристиками, танцювальним класом, нагородами, а також відгуками про викладання тощо. Може працювати в кількох танцювальних школах.
4. Анкета про пошук партнера - оголошення з бажаними параметрами парнера, танцювальним класом, віком, містом проживання тощо.
5. Школа - сторінка з інформацію про школу, адресу, танцівників, що відвідують її, тренерів, що працюють в ній, та відгуки.
6. Відгук - повідомлення та оцінка, залишена танцівником про тренера або танцювальну школу

Оскільки в майбутньому система може бути розширена іншими танцями, для легкої розширюваності потрібно виокремити всі дані, пов’язані з спортивно бальними танцями, в окремі об’єкти, що будуть складновими даних про танцівників, тренерів, а також зберігати та передавати інформацію про типи танців, до яких цей танцівник відноситься. Хоча на початку такий тип буде лише одним, саме цей підходить забезпечить безперешкодне додавання нових танців до системи Dance Bright.

Для того, аби зберігати інформацію про перебування в танцювальних школах не лише в даний момент, а протягом тривалого часу, потрібно додатково для кожного танцівника (або тренера) і школи зберігати дані про період відвідування (або викладання відповідно). Таким чином, до схеми потрібно додати проміжний об’єкт, який і буде тримати в собі додатково часовий період. Приклад того, як виглядатиме відповідний запит подано на рис. 12.

```
query A {
    dancers {
        schools {
            school {
                name
            }
            startedDate
            endedDate
        }
    }
}
```

Рис. 12. Приклад запиту з проміжним об’єктом, що зв’язує танцівника та школу

Аналогічно, аби забезпечити збереження історії танцювальних пар, потрібно створити свій проміжний об’єкт, який зв’язуватиме партнерів між собою. Додатково, якщо в майбутньому виникне потреба, можна буде додавати певні дані до цього об’єкту, які стосуватимуться обраної пари у певний період часу.

Повний опис об’єктів GraphQL наведено в додатку А (табл. 1).

Тепер, коли всі типи описані, необхідно визначити допустимі запити та мутації. Для танцівників вони показані на рис. 13 та рис. 14, де filter - об’єкт з критеріями для фільтрації, paging - об’єкт з інформацією про те, скільки саме потрібно отримати танцівників, та input - це поля типу Dancer, які необхідні для створення чи редагування даних про танцівника.

```
type Mutation {
    createDancer(input: CreateDancerInput!): Dancer
    updateDancer(input: UpdateDancerInput!): Dancer
    deleteDancer(id: ID!): Dancer
}
```

Рис. 13. GraphQL запити на танцівників

```
type Query {
    dancer(id: ID!): Dancer
    dancers(filter: DancerFilter, paging: Paging!): [Dancer!]
}
```

Рис. 14. GraphQL мутації з танцівниками

Для інших типів даних запити виглядають аналогічно.

4.3.2 Модель бази даних MongoDB

Основна частина даних, наведених в GraphQL схемі, буде зберігатись в базі MongoDB, і лише частину даних можливо буде обчислювати в процесі обробки запитів, не зберігаючи в базу (наприклад, вікову категорію пари можна обчислити на основі їхнього віку). Роглянемо ключові моменти в дизайні база даних, де структура відрізняється від наведеної в схемі GraphQL.

Оскільки частина даних про танцівників та тренерів повторюються, зручніше всього зберігати їх в одній колекції Users, з додаванням поля UserType, по якому буде визначатиметься тип користувача. Його можливі значення: USER, DANCER, TRAINER, і оскільки це значення буде постійно використовуватись при запитах на різних типів користувачів, воно повинно бути індексованим для швидкодії.

Дані про танцювальні школи варто зберігати у своїй колекції Schools, а оскільки найпопулярнішими полями для пошуку будуть Name та City, їх потрібно теж зробити індексними. Щоб забезпечити відношення “багато до багатьох" між танцівниками (та тренерами) і школами, на рівні шкіл буде зберігатись масив користувачів, які відвідують її, разом з попередньо згаданими даними про дату початку та кінця відвідування.

Оскільки відгуки належать і до тренерів, і до шкіл, зручно виділити для них окрему колекцію Reviews, а також додати індексоване поле ReviewType, яке визначатиме тип даних, до якого воно відноситься. Для того, щоб розуміти до чого саме вдіноситься цей відгук, потрібно додати індексоване поле RevieweeId, в якому зберігатиметься унікальний ідентифікатор школи чи тренера.

Дані, що стосуються специфічно бальних танців, такі як клас, нагороди, краще всього теж виділити в окрему колекцію Ballroom. Це потрібно для того, щоб при додаванні нових видів танцю, їх унікальні дані зберігались окремо і несправності даних однієї області не впливали на дані іншої.

Оскільки в системі буде виконуватись багато пошукових запитів на анкети з підбору партнерів, хоча їх і можна зберігати на рівні Users (разом з даними танцівника, який її створив), для оптимізації запитів потрібно зберігати анкети в окремій колекції з комбінованими індексами по найосновніших фільтрах.

Все інше, що стосується структури даних, відповідає структурі в схемі GraphQ, з точністю до конвертації примітивних типів.

4.4 Розробка серверної частини

4.4.1 Загальна архітектура

При розробці серверного коду скористаємось однією з найпоширеніших архітектурою для Web застосунків Three-Tier Architecture [19], згідно з якою логіка програми буде поділена на три частини:

1. Presentation Layer (рівень презентації) - код, який відповідає за спілкування з клієнтами, а саме GraphQL код, схема та резолвери
2. Logic Layer (рівень логіки додатку) - код, який буде виконувати основні функції системи (авторизація, збір та обчислення даних тощо) незалежно від того, як ці дані передаються на клієнт або як саме вони зберігаються
3. Data Tier (рівень доступу до даних) - код, який відповідатиме за збереження, зміну даних, тобто доступ до бази даних

Завдяки такому фізичному та логічному розподілу функціоналу досягається зручність в розробці, гнучкість у розширенні та висока надійність системи [19]. У майбутньому, якщо ми захочемо обмінюватись даними не лише через GraphQL, а й через інші мови запитів, потрібно буде лише доповнити рівень презентації - рівень логіки та доступу до даних залишаться без змін. І аналогічно, якщо буде потреба перейти до іншого способу збереження даних, доповнювати прийдеться лише рівень даних.

4.4.2 Рівень презентації

Після створення GraphQL схеми в п.4.2 нам потрібно інтегрувати її в наш проект. Оскільки синтаксис GraphQL не підтримується мовою typescript нативно, для того, щоб повертати описані доменні об’єкти нам потрібно або створити їх мовою TypeScript власноруч, або згенерувати їх на основі схеми автоматично, використовуючи сторонні бібліотеки.

Оскільки при створенні типів власноруч, при нових змінах нам потрібно змінювати і схему, і типи, краще всього генерувати їх за допомогою плагіна.

Тому, скористаємось одним з найпопулярніших генераторів, розробленим командою ApolloGraphQL [20]. Для цього в файл package.json необхідно додати наступні залежності:

- *apollo*
- *@types/graphql*
- *@graphql-codegen/cli*
- *@graphql-codegen/typescript*

- *@graphql-codegen/typescript-document-nodes*
- *@graphql-codegen/typescript-operations*
- *@graphql-codegen/typescript-resolvers*

Після цього, користуючись загальними рекомендаціями [21], опишемо в файлі codegen.yml за якими саме налаштуваннями будуть генеруватись typescript типи (рис. 15), де

- *schema/schema.graphql* - локальний шлях до файлу з GraphQL схемою
- *src/generated/types.ts* - локальний шлях до файлу, в якому будуть згенеровані typescript типи
- *plugins* - плагіни для генерації обов'язкових елементів роботи з GraphQL, таких як резолвери, операції та ноди [23]
- *avoidOptionals, useIndexSignature, immutableTypes* - рекомендовані налаштування для строгої типізації, яка необхідна для того, щоб виявити помилковий код на етапі компіляції
- *../types#GQLContext* - локальний шлях до файлу з класом, в якому будуть передаватись об'єкти для роботи з рівнем логіки

Тепер, коли все налаштовано, можемо генерувати типи за допомогою команди *npm graphql-codegen*. Приклад згенерованих типів показано на мал. 16.

Для того, щоб typescript типи оновлювались при зміні GraphQL схеми, додамо наведену вище команду до *build* скрипту в *package.json*.

codegen.yml

```yaml
schema: src/graphql/schema.graphql
generates:
  src/generated/types.ts:
    plugins:
      - typescript
      - typescript-operations
      - typescript-resolvers
      - typescript-document-nodes
    config:
      avoidOptionals: true
      useIndexSignature: true
      contextType: ../types#GQLContext
      immutableTypes: true

```

Рис. 15. Налаштування для генерації typescript типів

generated/types.ts

```typescript
export type TrainerData = {
  __typename?: 'TrainerData';
  reviews: Array<Review>;
  stars: Scalars['Int'];
};

export type TrainerFilter = {
  text: Scalars['String'];
  userId: Scalars['String'];
  stars: Scalars['Int'];
  type: ReviewType;
};
```

Рис. 16. Приклад згенерованих типів на основі схеми GraphQL

Далі по згенерованих типах залишається описати які дані звідки потрібно брати. На рис. 17 наведено код GraphQL резолвера танцювальної школи, де:

1. Дані, що стосуються танцювальної школи, отримуються за допомогою класу SchoolService по ID школи.
2. Дані, що стосуються танцівників, які відвідують школу, отримуються з допомогою класу DancerService, в який передається об’єкт фільтрування - ID школи.
3. Дані, що стосуються тренерів, які працюють в школі, отримуються з допомогою класу TrainerService, в який передається об’єкт фільтрування - ID школи.
4. Дані, що стосуються відгуків про школу, отримуються за допомогою класу ReviewService, в який передається ID школи та тип відгуку SCHOOL.

А на рис. 18 наведено код GraphQL мутацій для танцювальної школи, де операції на створення, редагування та видалення використовують відповідно SchoolService.

```
const schoolResolver: SchoolResolvers = {
    id: ({id}) => id,
    name: ({id}, args, {schoolService}) => schoolService.getSchool(id).name,
    locations: ({id}, args, {schoolService}) => schoolService.getSchool(id).locations,
    contacts: ({id}, args, {schoolService}) => schoolService.getSchool(id).contacts,
    about: ({id}, args, {schoolService}) => schoolService.getSchool(id).about,
    images: ({id}, args, {schoolService}) => schoolService.getSchool(id).images,
    dancers: ({id}, args, {dancerService}) => dancerService.listDancers({schoolId: id}),
    trainers: ({id}, args, {trainerService}) => trainerService.listTrainers({schoolId: id}),
    reviews: ({id}, args, {reviewService}) => reviewService.listReviews(id, ReviewType.SCHOOL),
    stars: ({id}, args, {reviewService}) => reviewService.getAverageStars(id, ReviewType.SCHOOL),
}
```

Рис. 17. Код GraphQL резолвера школи

```
const mutationResolver: MutationResolvers = {
    createSchool: ({input}, args, {schoolService}) => schoolService.createSchool(input),
    updateSchool: ({input}, args, {schoolService}) => schoolService.updateSchool(input),
    deleteSchool: ({id}, args, {schoolService}) => schoolService.deleteSchool(id),
};
```

Рис. 18. Код GraphQL мутацій школи

Таким чином, вся основна логіка делегується на рівень окремих класів: dancerService, trainerService, schoolService, reviewService та ballroomService. На рівні презентації лише описується хто саме “відповідальний” за кожен тип даних та операцій.

Аналогічно побудовано резолвери та мутації для інших сутностей.

4.4.3 Рівень доступу до даних

Для роботи з базою даних необхідно для кожної колекції (Users, Schools, Reviews, Ballroom) створити свій DAO з підтримкою CRUD операцій.

Розглянемо реалізовані функції для танцювальних шкіл:

1. Get - отримати школу по Id (рис. 19)
2. List - отримати всі школи, що задовольняють фільтру (рис. 20)
3. Create - створити школу (рис. 21)
4. Delete - видалити школу по Id (рис. 22)
5. Update - частково оновини дані школи (рис. 23)

```
async getSchool(id: string): Promise<School | null> {
  return this.collection
    .findOne({ id })
    .then(convertDbSchoolToSchool);
}
```

Рис. 19. Код функції GET класу SchoolDao

```
async listSchools(filter: ListSchoolsFilter): Promise<School[]> {
  return this.collection
    .find({})
    .toArray()
    .then((dbSchools) => dbSchools.map(convertDbSchoolToSchool));
}
```

Рис. 20. Код функції LIST класу SchoolDao

```
async createSchool(params: CreateSchoolParams): Promise<School> {
  return this.collection
    .create({
      id:uuid(),
      ...params
     }
    .toArray()
    .then((dbSchools) => dbSchools.map(convertDbSchoolToSchool));
  }
```

Рис. 21. Код функції CREATE класу SchoolDao

```
async deleteSchool(
  id: string
): Promise<School> {
  return this.collection
   .findOneAndDelete({ id })
   .then((result) => convertDbSchoolToSchool(result.value));
}
```

Рис. 22. Код функції DELETE класу SchoolDao

```
async updateSchool(
  params: SchoolUpdateParams
): Promise<School> {
  return this.collection
    .findOneAndUpdate(
      { id: params.id },
      { $set: { ...params } },
      { returnOriginal: false }
    )
    .then((result) => convertDbSchoolToSchool(result.value));
}
```

Рис. 23. Код функції UPDATE класу SchoolDao

DAO для інших колекції реалізовані аналогічним чином.

4.4.2 Рівень логіки додатку

На рівні логіки об’єднують запити з GraphQL та джерело даних - DAO. За це відповідають окремі класи DancerService, TrainerService, SchoolService, BallroomService, ReviewService. Для оптимізації вони використовують DataLoaders, описані в п.3.3, а окрім цього, їхній функціонал - виклик

відповідних функції потрібних DAO - якщо операція на List, викликається такий же метод в DAO.

4.5 Розробка клієнтської частини

4.5.1 Дизайн

Для створення клієнтської частини було обрано конструктор сайтів Wix та платформу Velo. На сторінці wix.com було створено пустий сайт з чистим макетом. За основу було обрано червоний, білий та бежевий кольори (рис. 24).

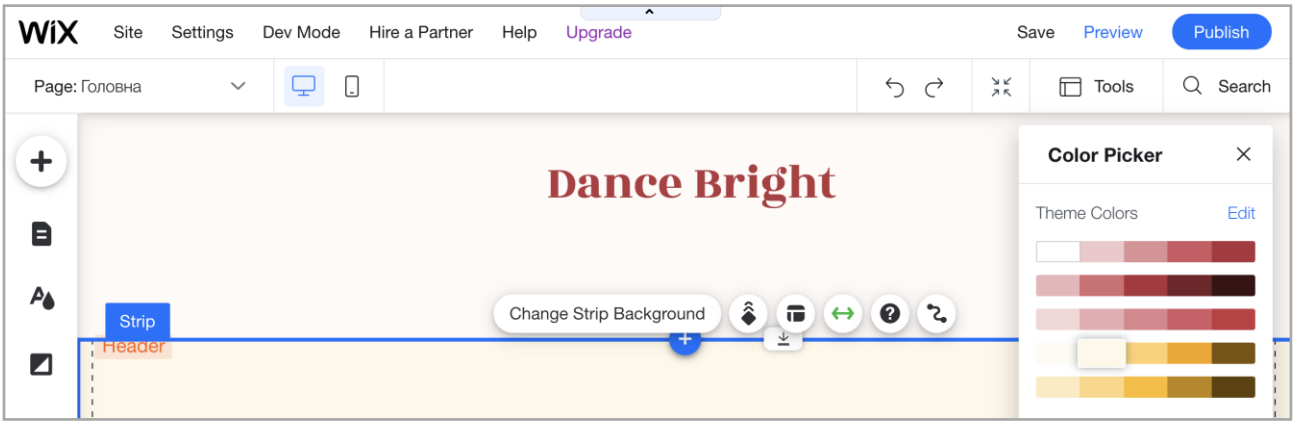

Рис. 24. Зображення обраного шаблону на палітри кольорів на wix.com

Для початку потрібно було створити “шапку” сайту, яка відображалася б на всіх сайтах і містила б у собі назву та емблему проекту, яка складається з назви та навігаційного.

Емблема танцювальної пари була взята з Wix каталогу безкоштовних картинок (рис. 25), а меню було створено з основними вкладками, визначеними в розділі 2: головна, танцівники, тренери та школи (рис. 26).

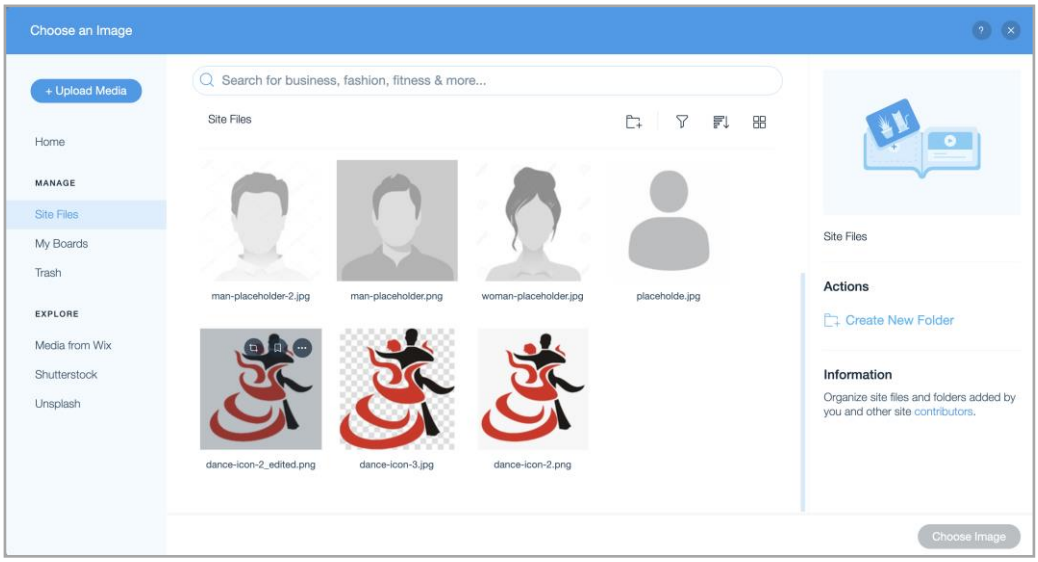

Рис. 25. Зображення файлового каталогу на wix.com

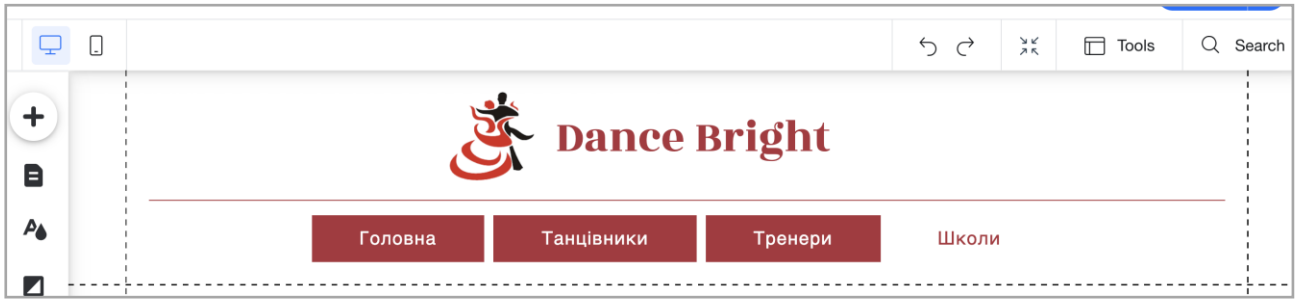

Рис. 26. Зображення “шапки” сайту на wix.com

Після цього необхідно створити сторінки танцівників, тренерів, шкіл, їх пошуку, головного меню і наповнити їх полями для даних, фільтрами, визначеними в розділі 2. Оскільки процес дизайну для них є практично однаковим, розглянемо процес створення найголовнішої сторінки: пошуку партнерів.

Для сторінки пошуку потрібно дві області - з фільтрами та результатами. Для цього було створено відповідно контейнер та таблицю (рис. 27).

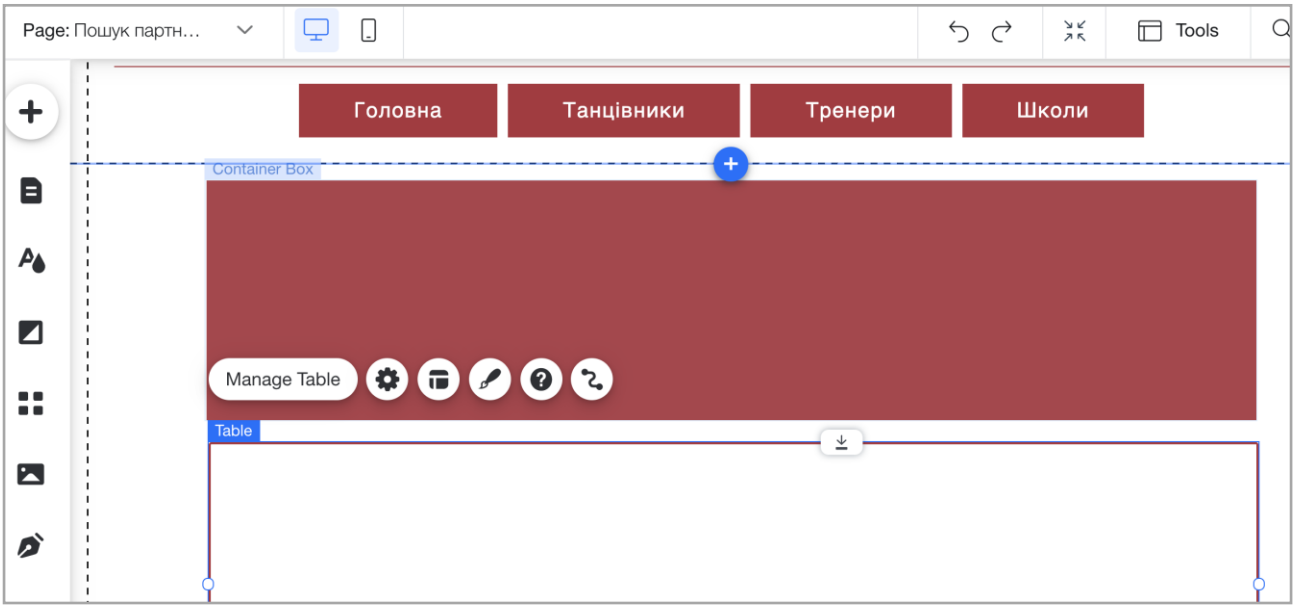

Рис. 27. Зображення контейнера та таблиці для пошуку партнерів на wix.com

Далі на панелі фільтрів були створені текстові поля для імені, міста та школи, числові поля для зросту та кількості годин в тиждень, радіо кнопки для статі та можливості переходу в іншу школу, а також списки для класів танцювання (рис. 28)

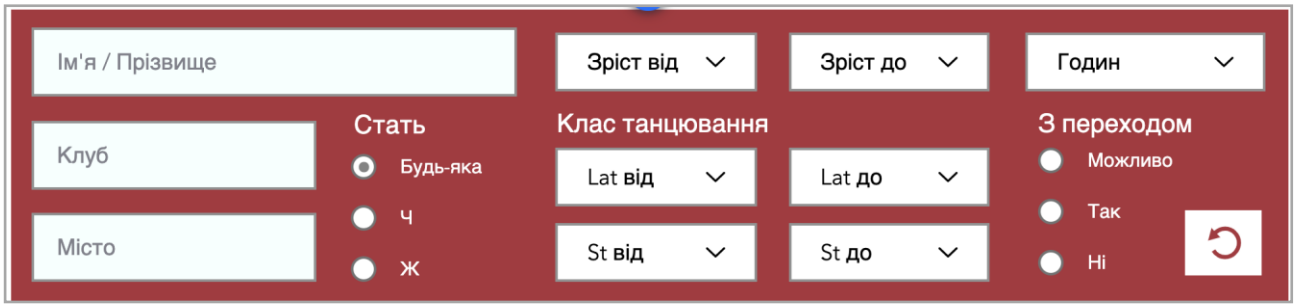

Рис. 28. Зображення готової панелі фільтрації при пошуку партнерів на wix.com

Перед заповненням таблиці з результатами, потрібно визначити типи та назви її колонок: фото, ім’я, вік, зріст, стать, школа та посилання на профіль з повною інформацією (рис. 29).

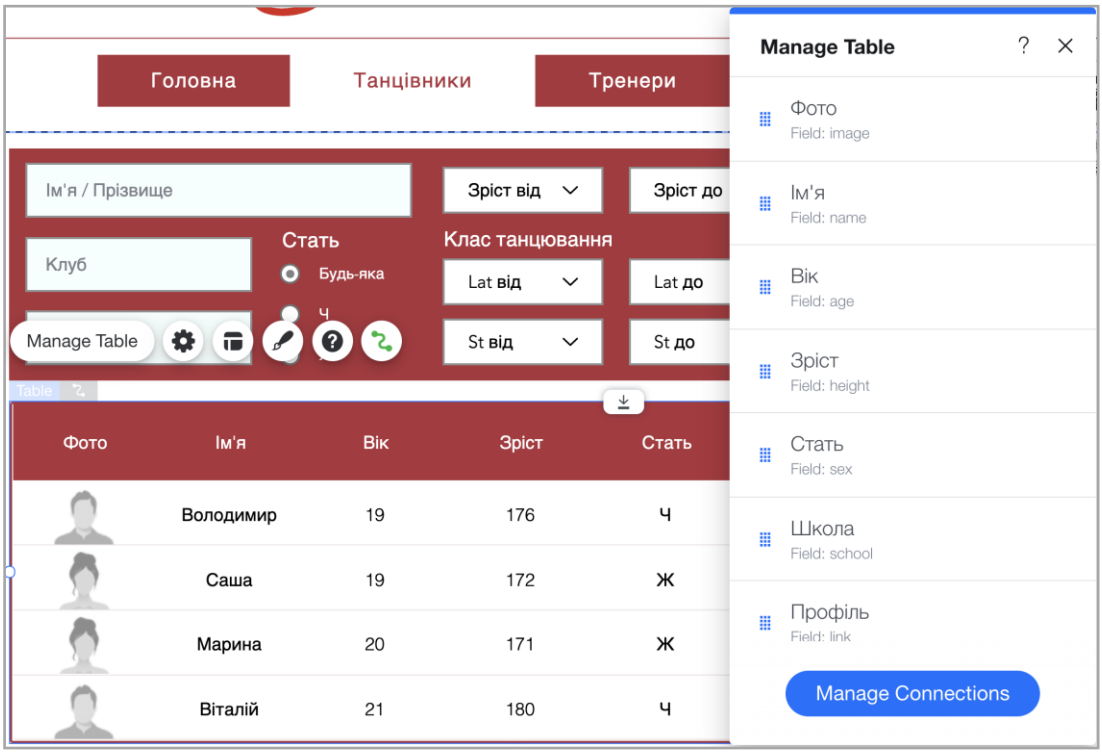

Рис. 29. Зображення налаштувань таблиці з тестовими даними на wix.com

Дизайн сторінки пошуку готовий, далі залишається налаштувати взаємодію між фільтрами та результатами за допомогою платформи Velo.

4.5.2 Взаємодія даних

Для того, щоб відобразити дані на сторінці або змінити їх, потрібно написати код, який реагував би на події на сторінці та викликав сервер по

GraphQL. Для цього нам і потрібна платформа Velo, оскільки вона дає необмежений доступ для роботи з кодом сторінки та зовнішніми запитами.

Для того, щоб відреагувати на натискання кнопки “Оновити”, було написано функцію обробки події, яка збирає дані з усіх фільтрів, конвертує в GraphQL запит, відправляє на сервер та відображає результати в таблиці танцівників (рис. 30).

```
$w("#refreshButton").onClick(async () => {
    const filter = {
        name: $w("#nameInput").value,
        school: $w("#schoolInput").value,
        city: $w("#cityInput").value,
        sex: $w("#sexRadioGroup").value,
        minHeight: $w("#minHeightDropdown").value,
        maxHeight: $w("#maxHeightDropdown").value,
        minLatClass: $w("#minLatClassDropdown").value,
        maxLatClass: $w("#maxLatClassDropdown").value,
        minStClass: $w("#minStClassDropdown").value,
        maxStClass: $w("#maxStClassDropdown").value,
        canTransfer: $w("#canTransferRadioGroup").value,
        hoursPerWeek: $w("#hoursPerWeekDropdown").value,
    };
    const query = convertPartnersFilterToGraphQLQuery(filter)
    const dancers = await graphQLClient.executeQuery(query);
    $w("#dancersTable").rows = dancers;
})
```

Рис. 30. Код обробки натискання кнопки “Оновити”

Аналогічно реалізовані обробники подій на всіх інших сторінках.

ВИСНОВКИ

У даній роботі розглядалась проблема інтернет-комунікації новачків та аматорів танцювального спорту.

Проаналізовано функціональні можливості, виявлено переваги та недоліки існуючих інтернет-ресурсів, а також розглянута найбільш типові варіанти запитів, характерних для обраної предметної області.

Розроблено систему Dance Bright, яка дозволяє виконувати розширений пошук танцювальних партнерів, тренерів та шкіл за найпопулярнішими критеріями, створювати та редагувати анкети, а також залишати відгуки про тренерів та шкіл.

Використання при реалізації системи Dance Bright технології GraphQL дозволяє забезпечити гнучкість, при потребі, розширення системи як в контексті бальних танців, так і для інших видів спорту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Великий руйнівник-реформатор: Як COVID-19 підштовхує Україну до стрімкої цифрової трансформації [Електронний ресурс] – Режим доступу до ресурсу: https://www.ua.undp.org/content/ukraine/uk/home/blog/2020/how-COVID-19-is-nudging-Ukraine-towards-digital-transformation.html.
2. Dentsu Aegis Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.dentsu.com/.
3. Frontiers | Effects of Dance interventions on the aspects of the Participants Self [Електронний ресурс] – Режим доступу до ресурсу: https://www.frontiersin.org/articles/10.3389/fpsyg.2018.01130/full.
4. Взгляд на партнера с другой стороны пары [Електронний ресурс] – Режим доступу до ресурсу: https://portal.main.tpu.ru/diamant/sovet/sovet-other/vzglyad.81.
5. Танцевальный спорт в Украине [Електронний ресурс] – Режим доступу до ресурсу: https://dancesport.net.ua/
6. Рамблер / Топ-100 / Культура и искусство / Танец [Електронний ресурс] – Режим доступу до ресурсу: https://top100.rambler.ru/navi?categoryId=1085&page=1&subcategoryId=1096&resourceId=3030464#3030464.
7. Flymark [Електронний ресурс] – Режим доступу до ресурсу:
8. Apollo GraphQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/.
9. DataLoader [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/graphql/dataloader.
10. JavaScript | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

11. Usage statistics of JavaScript as client-side programming language on websites [Електронний ресурс] – Режим доступу до ресурсу: https://w3techs.com/technologies/details/cp-javascript.
12. Node.js | Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Node.js.
13. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com.
14. Free Website Builder | Wix.com [Електронний ресурс] – Режим доступу до ресурсу: https://www.wix.com.
15. About Velo by Wix [Електронний ресурс] – Режим доступу до ресурсу: https://support.wix.com/en/article/about-velo-by-wix.
16. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура.
17. A light on Client-Server, and Communication Techniques in between [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/a-light-on-client-server-and-communication-techniques-in-between-ac40e53a4318.
18. Guys, REST APIs are not Databases [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@marinithiago/guys-rest-apis-are-not-databases-60db4e1120e4.
19. Three-Tier Architecture | IBM [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/three-tier-architecture.
20. TypeScript GraphQL Code Generator [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/blog/typescript-graphql-code-generator-generate-graphql-types-with-apollo-codegen-tutorial/.
21. GraphQL Codegen Configuration [Електронний ресурс] – Режим доступу до ресурсу: https://www.graphql-code-generator.com/docs/getting-started/codegen-config.

ДОДАТКИ

Додаток А. Таблиця сутностей GraphQL схеми

Назва сутності	Назва поля	Тип даних	Пояснення
Dancer	Id	ID!	Унікальний ідентифікатор
	Email	String!	Адреса електронної скриньки
	RegistrationDate	String!	Дата реєстрації в системі
	Auth	Auth!	Деталі авторизації в системі
	Contacts	[Contact!]!	Контактні дані
	IsVerified	Boolean!	Статус підтвердження акаунту
	DanceTypes	[DanceTypeEnum!]!	Типи танців серед існуючих в системі, які танцює користувач
	Awards	[Award!]!	Нагороди
	BirthDate	String	Дата народження
	Sex	SexEnum	Стать
	Height	Int	Зріст
	Schools	[SchoolMembership!]!	Танцювальні школи, які користувач відвідував або відвідує
	City	String	Місто, в якому переважно мешкає користувач
	Country	CountryEnum	Країна, в які переважно мешкає танцівник
	About	String	Додаткові дані про себе
	Images	[Image!]!	Фотографії танцівника
Trainer	Id	ID!	Унікальний ідентифікатор
	Email	String!	Адреса електронної скриньки

Таблиця 1. Сутності GraphQ схеми

Назва сутності	Назва поля	Тип даних	Пояснення
Trainer	RegistrationDate	String!	Дата реєстрації в системі
	Auth	Auth!	Деталі авторизації в системі
	Contacts	[Contact!]!	Контактні дані
	IsVerified	Boolean!	Статус підтвердження акаунту
	DanceTypes	[DanceTypeEnum!]!	Типи танців серед існуючих в системі, які танцює тренер
	Awards	[Award!]!	Нагороди
	BirthDate	String	Дата народження
	Sex	SexEnum	Стать
	Height	Int	Зріст
	Schools	[SchoolMembership!]!	Танцювальні школи, в яких тренер викладав або викладає
	City	String	Місто, в якому переважно мешкає тренер
	Country	CountryEnum	Країна, в які переважно мешкає тренер
	Reviews	[Review!]!	Відгуки про тренера
	Stars	Int	Середня оцінка відгуків про тренера
	About	String!	Додаткові дані про себе
	Images	[Image!]!	Фотографії танцівника
School	Id	ID!	Унікальний ідентифікатор
	Name	String!	Назва школи
	Locations	[Location!]!	Адреса однієї або кількох будівель школи
	Contacts	[Contact!]!	Контактні дані
	Reviews	[Review!]!	Відгуки про школу
	Stars	Float	Середня оцінка відгуків про школу

Продовження таблиці 1

Назва сутності	Назва поля	Тип даних	Пояснення
School	About	String!	Додаткові дані про школу
	Dancers	[SchoolMembership!]!	Танцівники, які відвідували або відвідують школу
	Trainers	[SchoolMembership!]!	Тренери, які викладали або викладають у школі
	Images	[Image!]!	Фотографії танцівника
BallroomDetails	LatClass	BallroomClassEnum	Танцювальний клас стандартної програми
	StClass	BallroomClassEnum	Танцювальний клас латино-американської програми
BallroomDetails	AgeCategory	AgeCategoryEnum	Вікова категорія
	Awards	[BalroomAward!]!	Нагороди
	Partners	[Partnership!]!	Минулі та поточні партнери
SchoolMembership	Id	ID!	Унікальний ідентифікатор
	UserId	ID!	Ідентифікатор танцівника або тренера
	StartedDate	String!	Дати початку відвідування школи
	EndedDate	String	Дата завершення відвідування школи
	SchoolId	ID!	Ідентифікатор школи
Review	Id	ID!	Унікальний ідентифікатор
	ReviewType	ReviewTypeEnum!	Тип відгуку
	Text	String!	Текст відгуку
	Stars	Int!	Оцінка
	Date	String!	Дата
	UserId	ID!	Ідентифікатор користувача
Partnership	Dancer	Dancer!	Танцівник
	Partner	Dancer!	Партнер
	Role	Role!	Роль танцівника у парі

Продовження таблиці 1

Назва сутності	Назва поля	Тип даних	Пояснення
Partnership	CreatedDate	String!	Дати початку парного танцювання
	EndedDate	String	Дата кінця парного танцювання
Award	Id	ID!	Унікальний ідентифікатор
	AwardType	AwardTypeEnum!	Тип нагороди
	UserId	ID!	Ідентифікатор танцівника, чия це нагорода
	Text	Text!	Текст нагороди
	Score	Int!	Оцінка
PartnerForm	Id	ID!	Унікальний ідентифікатор
	DancerId	ID!	Ідентифікатор танцівника
	DesiredHeight	IntRange	Обмеження по зросту потенційного партнера
	DesiredAge	IntRange	Обмеження по віку потенційного партнера
	DesiredCities	[String!]	Обмеження по місту потенційного партнера
	DesiredSchools	[ID!]	Обмеження по школам для танцювання з потенційним партнером
	DesiredStClass	[BallroomClassEnum!]	Обмеження по класу танцювання стандартної програми
	DesiredLatClass	[BallroomClassEnum!]	Обмеження по класу танцювання латино-американської програми
	DesiredHoursPerWeek	IntRange	Обмеження по кількості годин тренувань у тиждень
	CanChangeSchool	Boolean	Обмеження на перехід в іншу школу для партнера
	About	String	Додаткові дані

Кінець таблиці 1