МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ВОЛОДИМИРА ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ

КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

**Пояснювальна записка**

**до дипломної роботи**

**\_\_\_\_\_\_\_\_бакалавр \_\_\_\_\_\_\_\_**

(освітньо-кваліфікаційний рівень)

**на тему «Розробка серверної частини навчальної платформи на базі Node.js»**

Виконав: студент 4 курсу, групи ІТ-651

напряму підготовки 6.050103 „Програмна інженерія”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панфьоров Ю.І.

(підпис)

Керівник,

доцент, к.т.н.\_\_\_\_\_\_\_\_\_\_\_\_ Фесенко Т.М.

(підпис)

Рецензент,

доцент каф. ПМ,

к.ф.-м.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ковальов Ю.Г.

(підпис)

СЄВЄРОДОНЕЦЬК – 2019

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

бакалаврської роботи студента гр. ІТ-651 Панфьорова Ю.І.

Науковий керівник

доцент, к.т.н. Фесенко Т.М.

ПІБ, посада

Оцінка наукового керівника:

Рецензент Ковальов Ю. Г., к.ф.-м.н., доцент каф. ПМ СНУ ім. В. Даля

ПІБ, місто роботи, посада

Оцінка рецензента:

Кінцева оцінка за результатами захисту:

Голова ЕК,

зав. кафедри ПМ

д.т.н., доцент Лифар В.О.

підпис

**СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ВОЛОДИМИРА ДАЛЯ**

Факультет інформаційних технологій та електроніки

Кафедра програмування та математики

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 „Програмна інженерія”

|  |
| --- |
| ЗАТВЕРДЖУЮ  Завідувач кафедри ПМ,  д.т.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_ Лифар В.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019 р. |

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Панфьорова Юрія Ігоровича

**1. Тема роботи Розробка серверної частини навчальної платформи на базі Node.js.**

**керівник роботи к.т.н., доцент Фесенко Тетяна Миколаївна**

затверджені наказом вищого навчального закладу від «23» квітня 2019 року № 68/14.04

2. Строк подання студентом роботи 6 червня 2019 р.

3. Вихідні дані до роботи

Об'єктом досліджень є клієнт-серверні технології на основі веб-додатку навчальної платформи.

3.1 Літературні джерела:

Express in Action. Writing, building, and testing Node.js applications. Evan M. Hahn. – 2016, 256 с.

The road to GraphQL. Robin Wieruch. – 2018, 340 с.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ

4.2 Аналіз предметної галузі (огляд літератури), з висвітленням наступних питань:

Поняття навчальної платформи.

Аналіз існуючих навчальних платформ.

Вимоги до навчальних платформ.

4.3 Основна частина, в якої висвітлити:

Визначення веб-додатку.

Архітектура клієнт-серверної взаємодії.

Вимоги до серверної частини веб-додатку.

Аналіз технологій реалізації серверної частини.

Розробка серверної частини веб-додатку.

4.4 Висновки

4.4 Перелік використаних джерел

5. Перелік графічного матеріалу немає

6. Дата видачі завдання 23 квітня 2019 року.

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів дипломної роботи | Строк виконання етапів роботи | Примітка |
| 1 | Одержання завдання на виконання роботи | 23.04.19 |  |
| 2 | Укладання і погодження з керівником плану і етапів виконання роботи | 23.04.19 |  |
| 3 | Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі» | 26.04.19 |  |
| 4 | Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху | 02.05.19 |  |
| 5 | Проектування інфологічної моделі задачі що реалізується. | 09.05.19 |  |
| 6 | Укладання програмного продукту | 11.05.19 |  |
| 7 | Укладання, оформлення та погодження пояснювальної записки з керівником | 25.05.19 |  |
| 8 | Здача готової пояснювальної записки на кафедру | 06.06.19 |  |
| 9 | Укладання доповіді і презентації | 07.06.19 |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Панфьоров Ю. І.

(підпис)

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ Фесенко Т. М.

(підпис)

РЕФЕРАТ

Текст – 68 с., рис. – 11, табл. – 2, додатків – 0, літературних джерел – 23

У ході виконання дипломної роботи були розглянуті актуальність розробки навчальних платформ, існуючі рішення та їх недоліки, висунуті вимоги до навчальної платформи та серверної частини. Була освітлена клієнт-серверна архітектура веб-додатку, побудована інформаційна модель, спроектована бізнес-логіка та проаналізовані технології реалізації серверу. Була розроблена база даних, сервер та сформовані механізми взаємодії з клієнтом.

Кінцевим результатом є серверна частина веб-додатку навчальної платформи. Всі поставлені задачі виконані та всі цілі досягнуті.

Ключові слова: МОДЕЛЬ, СУТНІСТЬ, ЗВ'ЯЗОК, ВЕБ-ДОДАТОК, Node.js, GraphQL, Apollo Server, Express.js, MongoDB, Mongoose.

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БД – база даних.

СКБД – система керування базою даних.

HTML – HyperText Markup Language, мова гіпертекстової розмітки.

JSON – JavaScript Object Notation, об’єктна нотація JavaScript.

XML – Extensible Markup Language, розширювана мова розмітки.

URL - Uniform Resource Locator, уніфікований покажчик інформаційного ресурсу.

API – Application Programming Interface, інтерфейс прикладного програмування.

HTTP – HyperText Transfer Protocol, протокол передачі HyperText.

IP – Internet Protocol, міжмережевий протокол

ЗМІСТ

[ВСТУП 9](#_Toc11400362)

[РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 11](#_Toc11400363)

[1.1. Поняття навчальної платформи 11](#_Toc11400364)

[1.2. Аналіз існуючих навчальних платформ з курсами з Node.js 11](#_Toc11400365)

[1.3. Недоліки сучасних сервісів і шляхи вирішення проблем 14](#_Toc11400366)

[1.4. Вимоги до навчальної платформи 15](#_Toc11400367)

[РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ 16](#_Toc11400368)

[2.1 Визначення веб-додатку 16](#_Toc11400369)

[2.2 Архітектура клієнт-серверної взаємодії веб-додатку 16](#_Toc11400370)

[2.3 Вимоги до серверної частини веб-додатку 19](#_Toc11400371)

[2.4 Інформаційна модель даних 19](#_Toc11400372)

[2.5 Бізнес-логіка серверної частини 25](#_Toc11400373)

[2.6 Аналіз та вибір технологій реалізації серверної частини веб-додатку 28](#_Toc11400374)

[*2.6.1 Node.js* 29](#_Toc11400375)

[*2.6.2 Express.js* 32](#_Toc11400376)

[*2.6.3 GraphQL* 34](#_Toc11400377)

[*2.6.4 Apollo Server* 40](#_Toc11400378)

[*2.6.5 MongoDB* 41](#_Toc11400379)

[*2.6.6 Mongoose* 42](#_Toc11400380)

[РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ НАВЧАЛЬНОЇ ПЛАТФОРМИ 45](#_Toc11400381)

[3.1. Реалізація бази даних 49](#_Toc11400382)

[3.2. Ініціалізація сервера 53](#_Toc11400383)

[3.3. Реалізація взаємодії з клієнтом 55](#_Toc11400384)

[ВИСНОВКИ 65](#_Toc11400385)

[СПИСОК ЛІТЕРАТУРИ 66](#_Toc11400386)

# ВСТУП

Актуальність досліджень.

У сучасному світі все більш поширюються ідеї отримання освіти у сфері програмування. Класична форма навчання не встигає за сучасними темпами розвитку інтернет технологій. Саме тому збільшується роль навчальних онлайн платформ, що надають найбільш сучасні та необхідні для фахівця даної галузі знання.

В дипломній роботі не будуть розглядатися переваги та недоліки онлайн освіти порівняно з традиційною. Відзначимо лише, що на даний момент онлайн освіта – особливий та перспективний вид освіти, що може доповнювати традиційну освіту, а може існувати абсолютно самостійно.

Сучасна навчальна онлайн платформа повинна мати актуальну інформацію, бути доступною для всіх, незалежно від достатку, віку та часу. Рішення, що існують на даний момент, не завжди відповідають критеріям, що перелічені вище.

У сучасному програмуванні існує багато веб-технологій, серед яких особливо виділяється Node.js, яка швидко розвивається, має відмінну продуктивність та є простою в освоєнні.

Наявні онлайн курси з технології Node.js є або обмеженими за часом, або не мають української локалізації, або не є фінансово доступними. В українському сегменті мережі Інтернет не вистачає актуальних даних з цієї технології.

Розробка навчальної платформи, яка відповідає заданим критеріям та не має перелічених недоліків є актуальною задачею.

Об’єкт досліджень: Клієнт-серверні технології на основі веб-додатку навчальної платформи.

Предмет досліджень: Серверна частина веб-додатку.

Мета дослідження: Розробка серверної частини веб-додатку навчальної платформи.

Завдання дослідження:

а) Проаналізувати існуючі рішення;

б) Провести дослідження технологій і механізмів взаємодії клієнт-серверної архітектури;

в) Розробити серверну частину веб-додатка.

Методи дослідження: Технології які використовуються для створення веб-додатку.

# РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

## 1.1. Поняття навчальної платформи

Навчальну платформу можна розглядати як середовище, яке надає учням широкий набір інструментів для навчання, в тому числі дистанційного. Дистанційна властивість навчальної платформи дозволяє користувачам отримувати доступ у будь-який час, у будь якому місці та з будь-якого пристрою, що є підключеним до мережі Інтернет.

Головною відмінністю навчальної платформи від традиційної освіти є її доступність незалежно від місцезнаходження, часу та віку користувача. Їх використовують вдома, у школах, у вищих навчальних закладах і в компаніях, для швидкого підвищення кваліфікації.[1]

## 1.2. Аналіз існуючих навчальних платформ з курсами з Node.js

Професія програміста є однією зі списку найбільш затребуваних на сучасному ринку праці. Вимоги до таких спеціалістів підвищуються кожного дня. Щоб стати фахівцем у цій галузі потрібно приділяти увагу та час самоосвіті, бо сучасні тенденції розвитку мов програмування потребують постійного оновлення знань фахівця. Особливо сильно постає питання актуальності знань у сфері веб-програмування, тому що мови програмування, що дозволяють будувати повноцінні веб-проекти, та їх стандарти оновлюються дуже швидко. Однією з найважливіших технологій у даній сфері є платформа для виконання високопродуктивних мережевих за стосунків мови JavaScript - Node.js, засновником якої є Раян Дал. Поширення цієї платформи обумовлюється тим, що вона швидко розвивається, є дуже гнучкою та простою у засвоєнні та має велику спільноту розробників.

Навчальні платформи, що мають курси з програмування, стають все більш популярними. Найбільш відомими є:

* 1. Codecademy,
  2. w3schools.com,
  3. Udemy.

Codecademy є інтерактивною онлайн-платформою з навчання 12 мов програмування: Python, PHP, JavaScript, Ruby, Java та ін. Курси проходять на англійській мові. В онлайн-навчання імплементована система заохочувальних досягнень за виконання різних вправ, які можуть бачити інші користувачі. Також на сайті є форум, де новачки можуть поставити свої запитання та отримати зворотній зв'язок від експертів в нішевих областях. Для деяких курсів створена «пісочниця», де користувачі можуть тестувати свої програмні коди. Станом на січень 2014 року, 24 мільйони користувачів виконали понад 100 мільйонів вправ. Базове навчання безкоштовне, курси Pro і Pro Intensive дають більше можливостей для навчання на платній основі.

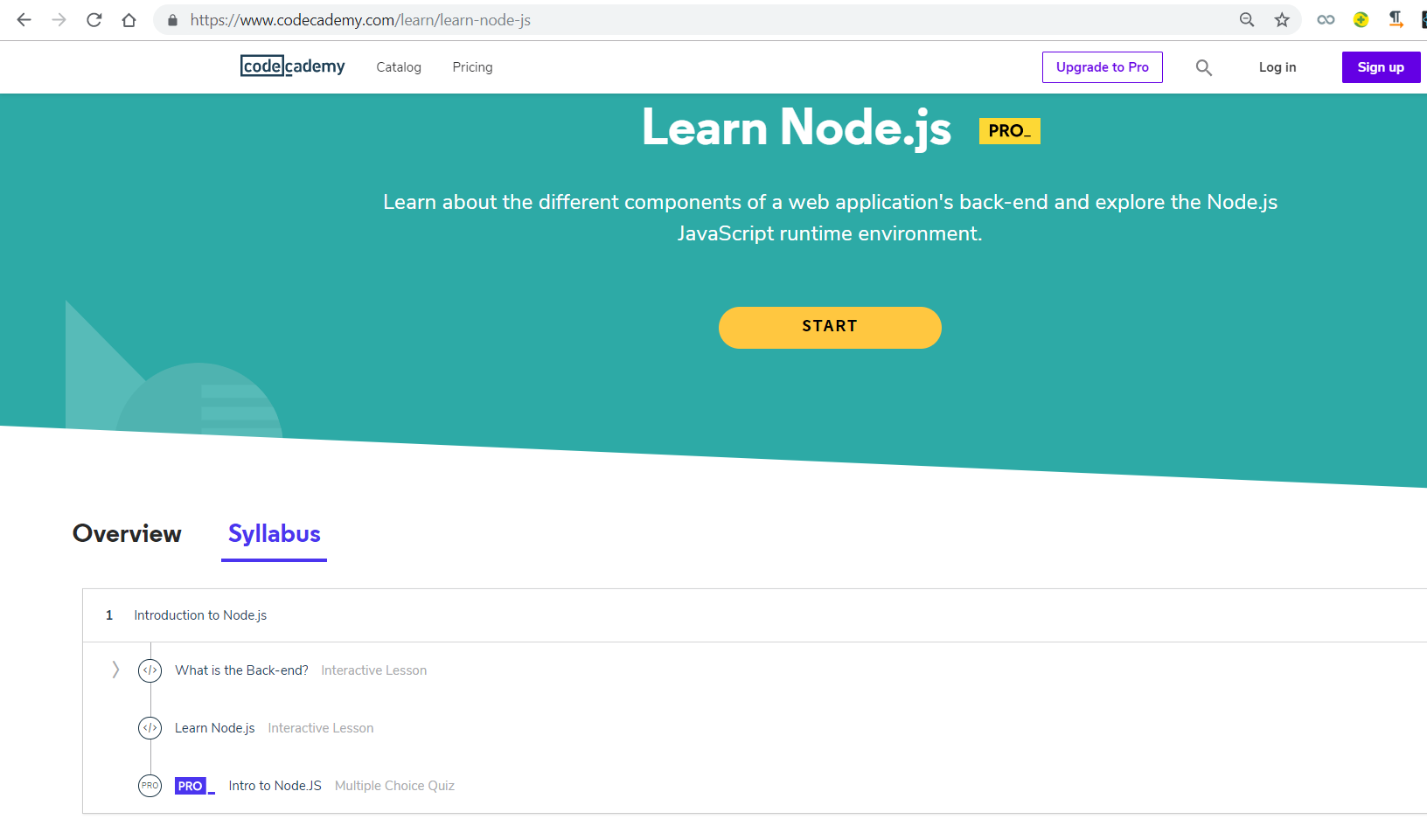


Рисунок 1.1. - Приклад курсу сайту Codecademy

W3schools.com - це веб-сайт для вивчення веб-технологій. Є можливість запускати приклади коду у обмеженому середовищі розробки – пісочниці. Можна навчатись як інтерактивно, так і знайти підручники та приклади різних задач. Всі курси зручно посортовані по розділах, можна перейти саме до того, який цікавить. Після завершення курсу можна отримати платний сертифікат.

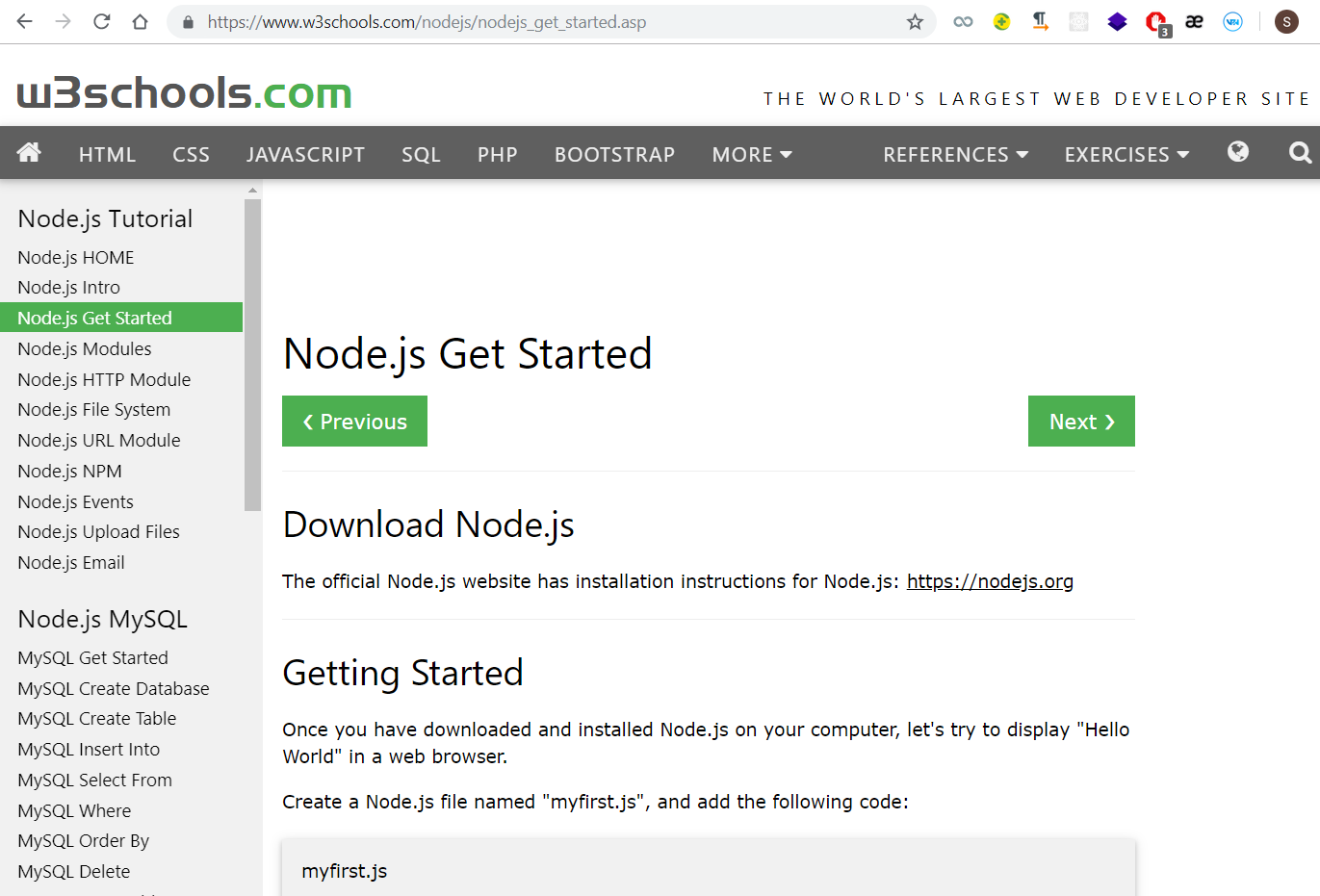


Рисунок 1.2. - Приклад курсу сайту w3schools

На платформі Udemy представлені найрізноманітніші курси, серед яких продуктивність, стиль життя, музика. Також можна знайти курси по програмуванню. Є як безкоштовні курси, так і платні. Навчальні матеріали представлені у вигляді відео, аудіо, презентаціями та текстом. Udemy також пропонує можливість організаціям створювати власні навчальні проекти для корпоративного навчання. Є багато безкоштовних вступних курсів, але більшість детальних курсів – платні.

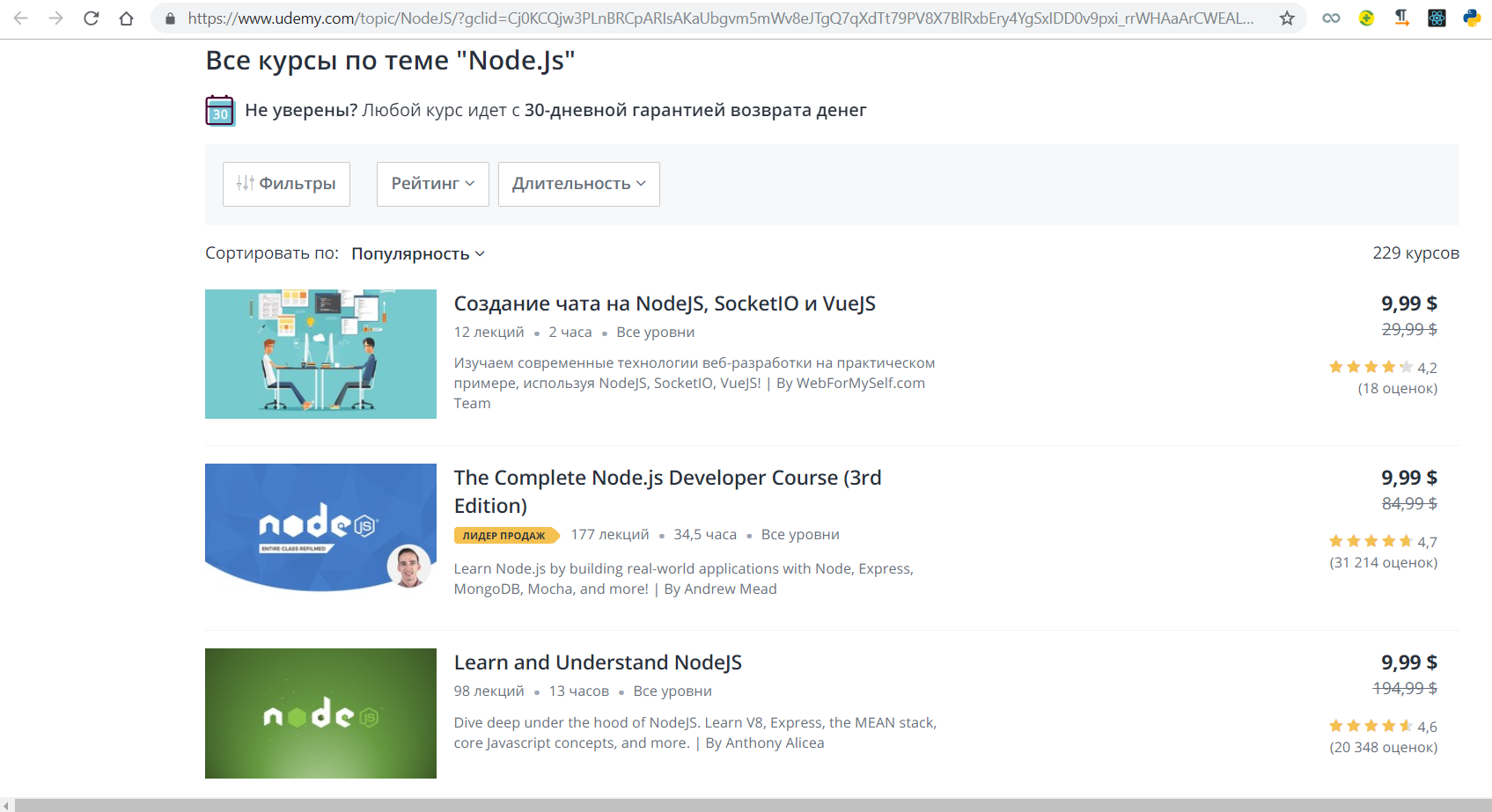


Рисунок 1.3. - Приклад курсів сайту Udemy

## 1.3. Недоліки сучасних сервісів і шляхи вирішення проблем

Підводячи підсумок щодо вищеописаного, перелічені навчальні платформи мають в тій чи іншій мірі декілька істотних недоліків:

* 1. Дані платформи не належать до українського сегменту мережі Інтернет та не мають інформації, що представлена на українській мові.
  2. Курси з Node.js на даних платформах є платними, по підписці або єдиноразовою покупкою.
  3. Безкоштовні курси з Node.js на даних платформах не мають необхідної кількості інформації з технології. Більш вагома інформація подається в окремих платних частинах курсу.

Для вирішення проблеми сучасних навчальних платформ щодо доступності знань з технології Node.js було прийнято рішення:

* 1. сформувати вимоги до навчальної платформи;
  2. побудувати інформаційну модель даних;
  3. сформувати бізнес-логіку навчальної платформи;
  4. реалізувати навчальну платформу у вигляді веб-додатку

## 1.4. Вимоги до навчальної платформи

При обміркуванні проблем, які існують у сучасних навчальних платформах, до проектованої навчальної платформи були сформовані наступні вимоги:

* 1. надавати інформацію безкоштовно;
  2. бути локалізованою на українську мову;
  3. інформація з курсу повинна бути актуальною;
  4. бути захищеною від несанкціонованого доступу;
  5. надійно зберігати дані користувача;
  6. надавати можливість оцінки знань;

Всі перелічені вимоги до навчальної платформи є фундаментом для більш конкретних вимог до серверної частини веб-додатку.

# РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

## 2.1 Визначення веб-додатку

**Веб-додаток** – це програмне забезпечення, що виконується на сервері, що є постачальником даних та послуг і в якості якого зазвичай виступає веб-сервер, та на клієнті, який зазвичай представлений веб-браузером.[2]

Веб-додатки мають ряд переваг перед звичайним програмним забезпеченням. Оскільки вони працюють у веб-браузерах, розробникам не потрібно розробляти веб-додатки для декількох платформ. Наприклад, одна програма, яка запускається в браузері Chrome, працюватиме як на операційній системі Windows, так і на OS X. Розробникам не треба поширювати оновлення програм для користувачів, коли оновлено веб-додаток. Коли оновлюється програма на сервері, всі користувачі автоматично отримують доступ до оновленої версії.

## 2.2 Архітектура клієнт-серверної взаємодії веб-додатку

Клієнт-серверна модель веб-додатку являє собою структуру програми, яка розподіляє завдання і навантаження між серверами і клієнтами. По суті і клієнт, і сервер - це програмне забезпечення. Як правило вони розташовані на різних обчислювальних машинах, і обмінюються даними через мережу за допомогою мережевих протоколів. Клієнтська частина такого веб-додатка відображає інформацію, що отримується з сервера. Хост сервера запускає одну або кілька серверних програм, які розподіляють свої ресурси між клієнтами. Клієнт запитує вміст сервера. Сервер очікує запити, а клієнт ініціює сеанс зв'язку з ним. Однією з переваг такого підходу є той факт, що клієнтська частина не залежить від операційної системи користувача, завдяки чому веб-додаток є міжплатформовим сервісом.[3]

Запити клієнта приймає та обробляє сервер, який взаємодіє з базою даних (далі БД) завдяки системі управління базою даних (СУБД). Це дає перевагу у відсутності пересилання великих обсягів даних. Оптимізовані запити потребують набагато менше часу, що в свою чергу:

1. знижує час очікування результату запиту;
2. підвищує швидкодію системи.

При виконанні запитів на сервері суттєво підвищується ступінь безпеки даних, оскільки правила цілісності даних визначаються в базі даних на сервері і є єдиними для всіх додатків, що використовують цю БД.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов’язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

* 1. рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
  2. прикладний рівень, який реалізує основну логіку веб-додатку і на якому здійснюється необхідна обробка інформації;
  3. рівень управління даними, який забезпечує зберігання даних та доступ до них.[4]

Дволанкова клієнт-серверна архітектура – це архітектура, що передбачає взаємодію двох компонентів  —  клієнта та сервера. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

* 1. модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
  2. модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта.

Трьохланкова клієнт-серверна архітектура, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка веб-додатку. Програми проміжного рівня можуть функціювати під управлінням спеціальних серверів веб-додатку, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Дволанкова архітектура є простішою в реалізації, тому що всі запити обслуговуються одним сервером.



Рисунок 2.1. - Схема дволанкової архітектури клієнт-сервер

Прикладом клієнт-серверної взаємодії є мережа Інтернет. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Основна ідея архітектури «клієнт-сервер» полягає в поділі мережевого додатка на кілька компонентів, кожен з яких реалізує специфічний набір сервісів. Компоненти такого додатку можуть виконуватися на різних комп’ютерах, виконуючи серверні і/або клієнтські функції. Це дозволяє підвищити надійність, безпеку і продуктивність мережевих додатків і мережі в цілому.

## 2.3 Вимоги до серверної частини веб-додатку

Серверну частину було вирішено розробляти як окремий компонент, щоб забезпечити стійкість, автономність та масштабованість системи та швидкість розробки.

Дані зберігаються на сервері, що містить наступні елементи:

* + 1. систему управління базами даних (СУБД)
    2. програмні модулі, що забезпечують функціонування бізнес-логіки

Основними функціями сервера у веб-додатку є:

* + 1. прийом запитів від клієнта;
    2. обробка запитів;
    3. виконання запитів до БД;
    4. відправка результатів запитів до клієнта;
    5. забезпечення системи безпеки;
    6. забезпечення стабільності режиму роботи;

Виходячи з вимог до навчальної платформи, що були розглянуті у Розділі 1, були сформовані наступні вимоги до серверної частини веб-додатку:

* 1. здійснювати взаємодію з базою даних, стосовно операцій:
     1. створення даних;
     2. зчитування даних;
     3. оновлення даних;
     4. видалення даних;
  2. структурувати поведінку запитів з клієнта та відправку результатів запиту з серверу;
  3. забезпечити механізм автентифікації;
  4. забезпечити механізм авторизації;
  5. організувати структуроване збереження даних.

## 2.4 Інформаційна модель даних

Інформаційна система - організаційно впорядкована сукупність документів (масивів документів) та інформаційних технологій, в тому числі з використанням технічних засобів, що реалізують інформаційні процеси та призначені для зберігання, обробки, пошуку, розповсюдження, передачі та надання інформації.[5]

Головними функціями інформаційної системи є:

* + 1. обробка інформації;
    2. збереження інформації;
    3. пошук інформації;
    4. надання інформації за запитом;
    5. розповсюдження інформації.

Після аналізу завдання, проектування системи починається з розробки бази даних, яка є головним компонентом інформаційної системи навчальної платформи.

Проектування бази даних полягає у правильній побудові комплексу взаємозв’язаних моделей даних.

Інформаційна модель — це абстракт­ний об’єкт, який замінює об’єкт оригінальний із метою його дослідження, зберігаючи при цьому типові риси та властивості оригіналу, важливі для дослідження. При створенні моделі треба визначити основні характеристики об’єкта та допустиму погрішність цих характеристик, вхідні характеристики, взаємовідносини характеристик.

Створення інформаційної моделі важливе, щоб зрозуміти структуру, основні властивості, закони взаємодії складових об’єкта, який аналізується, навчитися керувати цим об’єктом та прогнозувати наслідки реалізації керування.[6]

Модель «сутність-зв’язок» (ER-модель) (з англійської - Entity-relationship model або entity-relationship diagram) — модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель — це мета-модель даних, тобто засіб опису моделей даних. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення, що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою.[7]

Дана модель являє собою результат процесу, що описує та визначає предметну галузь. Інформація у моделі представлена у вигляді компонентів (сутностей), що пов’язані між собою зв’язками. Дані зв’язки виражають залежності між окремими сутностями. Кожна сутність може мати різні властивості (атрибути), які надають характеристику сутності з певної сторони.

Сутності поділяються на три види:

* + 1. Стрижнева (сильна сутність)
    2. Асоціативна сутність
    3. Характеристична сутність (слабка сутність)

Сильна сутність – це сутність що є незалежною від інших сутностей.

Асоціативна сутність є зв’язком «багато до багатьох» між двома сутностями.

Характеристична сутність – це сутність, що пов’язана з більш сильною сутністю зв’язками «один до багатьох» та «один до одного». Дана сутність описує іншу сутність і повністю залежить від неї. Якщо зникає сутність, яку описує характеристична сутність, то характеристична сутність також зникає. Характеристична сутність не може існувати без сутності, яку вона характеризує.

Кожен тип сутності має набір атрибутів, які призначені для опису сутності.

Види атрибутів:

* + 1. прості – можуть бути частиною складених атрибутів, складаються з одного компонента;
    2. складені – складаються з декількох простих атрибутів;
    3. однозначні – містять тільки одне єдине значення;
    4. багатозначні – містять декілька значень;
    5. довільні – містять значення, які формуються на основі інших значень.[8]

Зв’язок – це деяке відношення між двома типами сутностей.

Розрізняють 3 типи зв’зків між сутностями:

* + 1. «один до одного» або 1:1. Це означає, що одному екземпляру сутності може відповідати тільки один екземпляр іншої сутності;
    2. «один до багатьох» або 1:M. Це означає, що одному екземпляру сутності може відповідати будь-яка кількість (M) екземплярів іншої сутності. Якщо відоме значення максимальної кількості екземплярів, то це значення вказується замість символу M;
    3. «багато до багатьох» або M:N. Це означає, що декільком екземплярам однієї сутності може відповідати декілька екземплярів іншої сутності.

На підставі аналізу вимог до навчальної платформи можна виділити сутності, інформація про які буде зберігатися у базі даних:

* + 1. Сутність Користувач(User);
    2. Сутність Тест(Test);
    3. Сутність Питання(Question);
    4. Сутність Відповідь(Answer);
    5. Сутність Результат тесту(TestResult);
    6. Сутність Стаття(Chapter).

Кожна з перелічених вище сутностей має свої атрибути, які наведені у таблиці 2.1.

Таблиця 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Сутність | Атрибут | Тип | Значення |
| User | UserID | ID | Унікальний ключ користувача |
| username | String | Ім’я користувача |

Продовження таблиці 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| User | password | String | Пароль користувача |
| email | String | Адреса електронної пошти користувача |
| joinDate | Date | Дата реєстрації користувача |
| role | String | Права доступу користувача |
| testResults | Array of testResult | Результати тестів користувача |
| TestResult | testID | ID | Унікальний ключ результату тесту |
| testName | String | Назва тесту |
| results | Array of Boolean | Результати правильності відповідей результату тесту |
| score | Float | Успішність проходження тесту |
| Chapter | chapterID | ID | Унікальний ключ статті |
| title | String | Назва статті |
| uri | String | Уніфікований ідентифікатор ресурсу |
| Test | testID | ID | Унікальний ключ тесту |
| title | String | Назва тесту |
| questions | Array of Question | Питання тесту |
| Question | questionID | ID | Унікальний ключ питання |
| questionText | String | Текст питання |

Продовження таблиці 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Question | answers | Array of Answer | Можливі відповіді питання |
| Answer | answerID | ID | Унікальний ключ відповіді |
| answerText | String | Текст відповіді |
| isValid | Boolean | Правильність відповіді |

На підставі опису предметної галузі визначається зв’язок між наступними сутностями:

* 1. Тест та Питання: кожен Тест має багато Питань, але кожне Питання стосується тільки певного Тесту. Тому зв’язок між сутностями Тест та Питання - 1:M («один до багатьох»).
  2. Питання та Відповідь: у кожного Питання є багато варіантів Відповіді незалежно від їх правильності, проте для кожної відповіді існує лише одне Питання. Отже зв’язок між сутностями Питання та Відповідь - 1:M («один до багатьох»).
  3. Користувач та Результат тесту: Користувач отримує Результати тестів після проходження тесту. Користувач може мати багато Результатів тестів, проте кожен такий Результат тесту чітко закріплений за певним Користувачем. В цьому випадку зв’язок між сутностями Користувач та Результат тесту – 1:M («один до багатьох»).
  4. Тест та Результат тесту: кожному Результату тесту відповідає конкретний Тест. Тому зв'язок між сутностями Тест та Результат Тесту – 1:1 («один до одного»).

Варто відзначити що сутності Питання, Відповідь та Результат тесту є слабкими сутностями, або характеристичними. Вони не можуть існувати окремо від сильних сутностей, які вони характеризують.

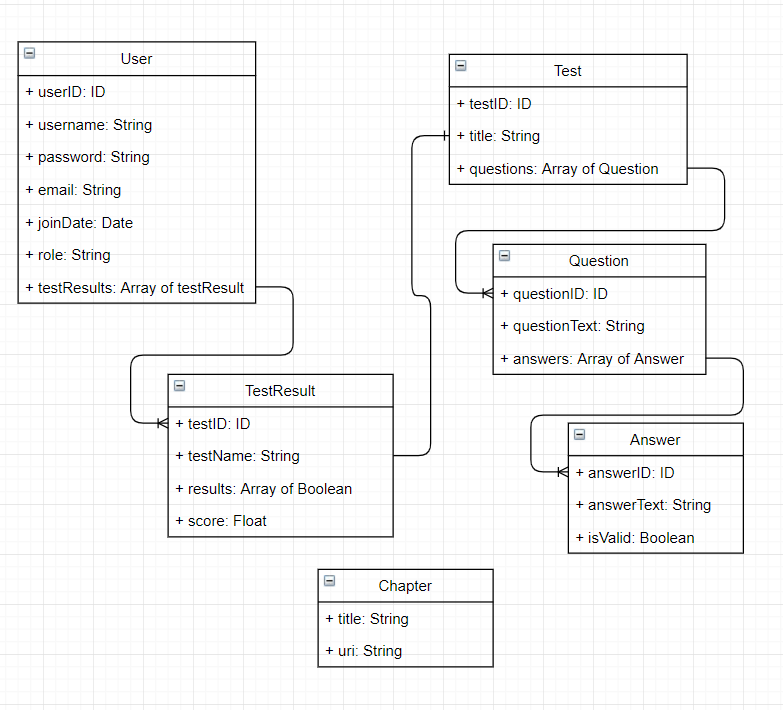


Рисунок 2.2. – ER діаграма бази даних навчальної платформи

## 2.5 Бізнес-логіка серверної частини

Бізнес-логіка – сукупність правил, принципів, залежностей поведінки об’єктів предметної галузі. Інакше кажучи бізнес-логіка – це реалізація правил та обмежень операцій, що автоматизуються.[9]

Аналізуючи вимоги та інформаційну модель, що були описані вище, можна виділити наступні варіанти бізнес-логіки:

* 1. Реєстрація нового користувача;
  2. Авторизація існуючого користувача;
  3. Створення тесту;
  4. Видалення тесту;
  5. Додавання статті;
  6. Перевірка пройденого користувачем тесту;
  7. Відправлення списку статей;
  8. Відправлення статті;
  9. Відправлення списку тестів;
  10. Відправлення тесту.

Розглянемо кожний з процесів більш детально.

Процес реєстрації нового користувача:

1. сервер отримує від клієнта запит на реєстрацію нового користувача та дані нового користувача;
2. сервер обробляє отримані дані, звіряючи їх зі збереженими в базі даних:
   1. якщо сервер знаходить даного користувача у базі даних, запит на створення нового користувача відхиляється;
   2. якщо сервер не знаходить даного користувача, новий користувач записується у базу даних і генерується та повертається тимчасовий ключ доступу.

Процес авторизація існуючого користувача:

1. сервер отримує від клієнта запит на авторизацію користувача та дані користувача;
2. сервер обробляє отримані дані, звіряючи їх зі збереженими в базі даних:
   1. якщо сервер не знаходить даного користувача у базі даних, запит на отримання доступу відхиляється;
   2. якщо сервер знаходить даного користувача, перевіряється правильність отриманого пароля:
      1. якщо пароль не відповідає отриманому з бази даних, запит на отримання доступу відхиляється;
      2. якщо пароль відповідає отриманому з бази даних, генерується та повертається тимчасовий ключ доступу.

Процес створення тесту:

1. сервер отримує від клієнта запит на створення нового тесту та дані тесту;
2. тест записується до бази даних;
3. записаний тест повертається до клієнту.

Процес видалення тесту:

1. сервер отримує від клієнта запит на видалення тесту та дані тесту;
2. сервер обробляє отримані дані, звіряючи їх зі збереженими в базі даних:
   1. якщо сервер не знаходить даний тест за унікальним ключем, запит на видалення тесту відхиляється;
   2. якщо сервер знаходить даний тест, даний тест видаляється з бази даних та повертається до клієнту.

Процес додавання статті:

1. сервер отримує від клієнта запит на додавання статті та дані статті;
2. стаття записується до бази даних;
3. записана стаття повертається до клієнту.

Процес перевірки пройденого користувачем тесту:

1. сервер отримує від клієнта запит на перевірку пройденого тесту та дані пройденого тесту:
   1. дані тесту;
   2. унікальний ключ користувача;
2. сервер звіряє обрані користувачем відповіді з правильними та формує результат тесту;
3. сервер отримує дані користувача за допомогою унікального ключа користувача;
4. сервер звіряє поточний результат тесту з попередніми результатами тестів:
   1. якщо ім’я поточного результату тесту збігається з одним з попередніх результатів тесту, поточний результат тесту замінює попередній;
   2. якщо ім’я поточного результату тесту не збігається з жодним з попередніх результатів тесту, поточний результат тесту записується до попередніх результатів тесту;
5. сформований результат тесту повертається до клієнту.

Процес відправлення списку статей:

1. сервер отримує від клієнта запит на отримання списку статей;
2. сервер звертається до бази даних та повертає з неї список статей;
3. список статей повертається до клієнту.

Процес відправлення статті:

1. сервер отримує від клієнта запит на отримання статті та унікальний ключ статті;
2. сервер виконує пошук за ключем та повертає з бази даних статтю, унікальний ключ якої співпадає з унікальним ключем статті, що розшукується;
3. отримана стаття повертається до клієнту.

Процес відправлення списку тестів:

1. сервер отримує від клієнта запит на отримання списку тестів;
2. сервер звертається до бази даних та повертає з неї список тестів;
3. список тестів повертається до клієнту.

Процес відправлення тесту:

1. сервер отримує від клієнта запит на отримання тесту та унікальний ключ тесту;
2. сервер виконує пошук за ключем та повертає з бази даних тест, унікальний ключ якого співпадає з унікальним ключем тесту, що розшукується;
3. отриманий тест повертається до клієнту.

## 2.6 Аналіз та вибір технологій реалізації серверної частини веб-додатку

Після постановки задачі, аналізу вимог, проектування структур даних та опису бізнес логіки серверної частини веб-додатку постало питання вибору технології її реалізації. Було вирішено обрати наступні технології:

* 1. Node.js;
  2. Express.js;
  3. GraphQL;
  4. Apollo Server;
  5. MongoDB;
  6. Mongoose.

### *2.6.1 Node.js*

PHP і Node.js є потужними бекендами для динамічних веб-сайтів. Обидві вони підпадають під одну й ту ж категорію, однак їх особливості дуже різні.

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері на сторонні користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості:

* + 1. асинхронна однопотокова модель виконання запитів;
    2. неблокуючий ввід/вивід;
    3. система модулів CommonJS;
    4. рушій JavaScript Google V8;

Для керування модулями використовується пакетний менеджер npm (node package manager).[10]

PHP (гіпертекстовий препроцесор) - це мова сценаріїв загального призначення, яка була розроблена для цілей веб-розробки. Вона була випущена в 1995 році, і з тих пір використовується як перший вибір мовних переваг для систем управління контентом, таких як WordPress, Drupal і Joomla, а також ряд сучасних фреймворків, таких як Laravel і Symphony.

Хоча і Node.js, і PHP є мовами сценаріїв на стороні сервера, вони, безсумнівно, матимуть різні подібності, однак між двома мовами сценаріїв існують різні подібності.

Дані технології мають наступні подібності:

Інтерпретовані мови - як Node.js, так і PHP можна запускати в середовищі виконання. І PHP, і Node.js є інтерпретованими мовами.

Життєздатний вибір - для створення веб-сайтів використовуються як PHP, так і Node.js, і обидві запускаються на стороні сервера. Node.js є більш обширним і швидшим в порівнянні з PHP, що може зробити його більш прийнятним вибором.[11]

Відмінності між PHP та Node.js наведені у таблиці 2.2

Таблиця 2.2

|  |  |  |
| --- | --- | --- |
| Особливості | PHP | Node.js |
| Робоче середовище | PHP має функцію, готову до встановлення, щоб використовувати його на стороні сервера | Node.js є середовищем виконання для Javascript на стороні сервера. |
| Двигуни | PHP працює на двигунах Zend | Node.js працює на основі механізму Google V8 javascript |
| Складність використання | PHP набагато простіше використовувати у порівнянні з Node.js. | Node.js не надто складний для використання, але вимагає більшої кількості ліній кодування та базового розуміння функцій замикання та зворотного виклику |

Продовження таблиці 2.2

|  |  |  |
| --- | --- | --- |
| JSON | PHP використовує його менше, ніж Node.js PHP використовує функції json\_encode () і json\_decode () | JSON працює краще за Node.js, ніж PHP. Node.js використовує JSON.stringify () і JSON.parse () |
| Виконання | PHP синхронний | Node.js асинхронний |
| Швидкість виконання | PHP є більш повільним, ніж Node.js | Швидше, ніж PHP, а також легкий у порівнянні з PHP. |
| Веб-сервер | PHP працює на веб-сервері Apache. Він також може працювати на веб-сервері IIS у випадку віконної машини. | Node.js не потребує веб-сервера, він працює у своєму середовищі часу виконання. |
| Менеджер пакетів | Широко використовується менеджер пакета Composer. | Менеджер пакетів вузлів (NPM) широко використовується. |

Підсумовуючи наведену вище інформацію стосовно цих двох технологій, можна відзначити що Node.js має декілька великих переваг, таких як швидкість виконання, краща робота з JSON-даними (що є доволі популярним форматом даних на сьогоднішній час), відсутність потреби в окремому веб-сервері, можливість використовувати такий же самий синтаксис як на сервері, так і на клієнті, інтенсивний розвиток та велика спільнота розробників.

З точки зору розвитку веб-сервера Node має ряд переваг:

* + 1. Чудова продуктивність. Вузол був розроблений для оптимізації пропускної здатності та масштабованості веб-додатків і є гарним рішенням для багатьох поширених проблем веб-розробки (наприклад, веб-додатків у реальному часі).
    2. Код написаний на "звичайному JavaScript", що означає, що витрачається менше часу на "зміщення контексту" між мовами, коли ви пишете як клієнтський, так і серверний код.
    3. JavaScript є відносно новою мовою програмування і має переваги від вдосконалення мовного дизайну в порівнянні з іншими традиційними мовами веб-серверів (наприклад, Python, PHP тощо). Багато інших нових і популярних мов компілюють / конвертують у JavaScript, тому ви також можете використовувати TypeScript, CoffeeScript, ClojureScript, Scala, LiveScript тощо.
    4. Менеджер пакетів вузлів (NPM) надає доступ до сотень тисяч багаторазових пакетів. Вона також має найкраще у своєму класі розрізнення залежностей, а також може використовуватися для автоматизації більшої частини інструменту побудови.
    5. Node.js портативний. Він доступний на Microsoft Windows, MacOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS і NonStop OS. Крім того, він добре підтримується багатьма провайдерами веб-хостингу, які часто надають спеціальну інфраструктуру та документацію для розміщення сайтів Node.
    6. Вона має дуже активну екосистему третіх сторін і спільноту розробників, з великою кількістю людей, які готові допомогти.[12]

Саме тому при постановці питання щодо вибору платформи для виконання веб-додатку навчальної платформи було віддано перевагу Node.js.

### *2.6.2 Express.js*

Веб-фреймворки на сервері ("фреймворки веб-додатків") - це програмні фреймворки, які полегшують написання, підтримку та масштабування веб-додатків. Вони надають інструменти та бібліотеки, які спрощують загальні завдання веб-розробки, включаючи маршрутизацію URL-адрес до відповідних обробників, взаємодіючи з базами даних, підтримуючи сеанси та авторизацію користувачів, виводячи файли форматування (наприклад, HTML, JSON, XML), і покращуючи безпеку від веб-атак.[13]

Express - мінімальний і гнучкий фрейморк веб-додатків Node.js, який забезпечує надійний набір функцій для веб- і мобільних додатків. З безліччю методів утиліти HTTP і проміжного програмного забезпечення, створення надійного API відбувається швидко і легко. Express надає тонкий шар основних функцій веб-додатків, не затушуючи функцій Node.js, які ви знаєте і любите. Багато популярних фреймворків базуються на Express.

Кілька популярних фреймворків Node.js побудовано на Express:

* + 1. ItemsAPI: пошук бекенду для веб- і мобільних додатків, побудованих на Express та Elasticsearch.
    2. KeystoneJS: веб-сайт та API Application Framework / CMS з автоматично створеним інтерфейсом адміністратора React.js.
    3. Kraken: безпечний і масштабований шар, який розширює Express, надаючи структуру і правила.
    4. LoopBack: високо-розширюваний, відкритий фреймворк Node.js для швидкого створення динамічних API-інтерфейсів REST.
    5. MEAN: фреймворк JavaScript Fullnack, що спрощує і прискорює розробку веб-додатків.
    6. Hydra-Express: Hydra-Express - це легка бібліотека, яка полегшує створення Node.js мікросервісів за допомогою ExpressJS.
    7. Locomotive: потужний веб-фреймворк MVC для Node.js від виробника Passport.js
    8. graphql-yoga: повнофункціональний, але простий і легкий GraphQL-сервер
    9. Express Gateway: повнофункціональний і розширюваний шлюз API з використанням Express як основи
    10. Kites: фреймворк веб-додатків на основі шаблонів
    11. NestJs: прогресивнний фреймворк Node.js для створення ефективних, масштабованих і корпоративних серверних додатків поверх TypeScript & JavaScript (ES6, ES7, ES8)

Express є найпопулярнішим веб-фреймворком Node і є базовою бібліотекою для ряду інших популярних веб-фреймворків Node. Він забезпечує механізми для:

* + 1. Запису обробників для запитів з різними HTTP-дієсловами на різних шляхах URL-адрес (маршрутів).
    2. Інтегрування з движками "view", щоб генерувати відповіді, вставляючи дані в шаблони.
    3. Встановлення загальних налаштуваннь веб-додатків, таких як порт для підключення, та розташування шаблонів, які використовуються для відображення відповіді.
    4. Додання додаткової обробки запиту (проміжного програмного забезпечення, з англійської middleware) в будь-якій точці конвеєра обробки запитів.

Хоча сам Express є досить мінімалістичним, розробники створили сумісні пакети програмного забезпечення для вирішення практично будь-якої проблеми веб-розробки. Є бібліотеки для роботи з файлами cookie, сеансами, входами користувачів, параметрами URL, даними POST, заголовками безпеки та багатьма іншими. Можна знайти список пакетів проміжного програмного забезпечення, які підтримує команда Express на http://expressjs.com/en/resources/middleware.html (разом зі списком деяких популярних пакетів третьої сторони. [14]

### *2.6.3 GraphQL*

GraphQL - це мова запитів для API і середовище виконання сервера для виконання запитів за допомогою системи типів, яка визначається для даних. GraphQL не прив'язаний до будь-якої конкретної бази даних або механізму зберігання даних, а підтримується існуючим кодом і даними.

Сервіс GraphQL створюється шляхом визначення типів і полів на цих типах, а потім надає функції для кожного поля кожного типу.

Як тільки служба GraphQL запущена (як правило, на URL-адресу веб-служби), її можна надіслати запити GraphQL для перевірки та виконання. Отриманий запит спочатку перевіряється, щоб переконатися, що він стосується тільки визначених типів і полів, а потім запускає надані функції для отримання результату.[15]

Запит має ту саму форму, що й результат. Це дуже важливо для GraphQL, тому що завжди повертається те, що очікується, і сервер точно знає, які поля вимагає клієнт.

Запити GraphQL можуть перетинати пов'язані об'єкти та їхні поля, дозволяючи клієнтам вибирати багато пов'язаних даних в одному запиті, замість того, щоб робити кілька кругових пропусків, як потрібно в класичній REST архітектурі.

Якщо єдине, що можна було б зробити, це перетнути об'єкти та їх поля, GraphQL вже був би дуже корисною мовою для отримання даних. Але коли додається можливість передавати аргументи полям, речі стають набагато цікавішими.

У такій системі, як REST, можна передати лише один набір аргументів - параметри запиту та сегменти URL-адреси у запиті. Але в GraphQL, кожне поле і вкладений об'єкт можуть отримати свій власний набір аргументів, що робить GraphQL повною заміною для створення декількох вибірок API. Можна навіть передати аргументи в скалярні поля, здійснити перетворення даних один раз на сервер, а не на кожний клієнт окремо.

Аргументи можуть бути різних типів. GraphQL поставляється з типовим набором типів, але сервер GraphQL може також оголосити свої власні типи, якщо вони можуть бути серіалізовані у транспортний формат.

GraphQL постачається з набором типів скалярних типів за замовчуванням:

* + 1. Int: Підписане 32-бітове ціле число.
    2. Float: підписане значення з подвійною точністю з плаваючою точкою.
    3. String: послідовність символів UTF-8.
    4. Boolean: true або false.
    5. ID: ID скалярного типу являє собою унікальний ідентифікатор, який часто використовується для видобування об'єкта або як ключ для кешу. Тип ідентифікатора серіалізується так само, як і рядок; однак, визначення його як ідентифікатора означає, що воно не призначене для читання з людиною.

У більшості реалізацій сервісу GraphQL існує також спосіб задати власні скалярні типи. Реалізація вирішує, як цей тип повинен бути серіалізований, десериализованний і перевірений. Наприклад, можна вказати, що тип дати завжди повинен бути серіалізований у цілочисельну мітку часу, і ваш клієнт повинен знати, що очікувати від цього формату для будь-яких полів дати.[16]

Більшість обговорень GraphQL зосереджується на витягуванні даних, але будь-яка повна платформа даних також потребує способу зміни даних на сервері.

У REST будь-який запит може викликати деякі побічні ефекти на сервері, але за умовами передбачено, що не потрібно використовувати GET-запити для зміни даних. GraphQL схожий - технічно будь-який запит може бути реалізований, щоб викликати запис даних. Однак корисно встановити умову, що будь-які операції, які викликають запис, повинні бути відправлені явно через мутацію.

Як і у запитах, якщо поле мутації повертає тип об'єкта, ви можете запитати вкладені поля. Це може бути корисним для отримання нового стану об'єкта після оновлення. Це особливо корисно при мутації існуючих даних, наприклад, при збільшенні поля, оскільки можна мутувати і запитувати нове значення поля за допомогою одного запиту.[17]

Наступний список показує основні переваги використання GraphQL у додатку:

* + 1. Отримання декларативних даних - GraphQL охоплює декларативні дані, які витягують його запити. Клієнт обирає дані разом зі своїми сутностями з полями через відносини в одному запиті. GraphQL вирішує, які поля потрібні для його інтерфейсу, і він практично діє як витягування даних з інтерфейсом користувача. Щоб отримати всі дані в одному запиті, запит GraphQL, який обирає лише частину даних для інтерфейсу користувача, має сенс. Він пропонує велике розмежування проблем: клієнт знає про вимоги до даних; сервер знає про структуру даних і про те, як повернути дані з джерела даних (наприклад, бази даних, мікросервісу, стороннього API).
    2. Немає перенавантаження - перевантаження в GraphQL не відбувається. Мобільний клієнт зазвичай перевантажує дані, коли існує ідентичний API, як веб-клієнт з RESTful API. За допомогою GraphQL мобільний клієнт може вибрати інший набір полів, щоб він міг отримувати тільки інформацію, необхідну для екрану.
    3. Самостійність - GraphQL цікавий не тільки для розробників React. У той час як Facebook демонстрував GraphQL на клієнтському додатку з React, він відокремлений від будь-якого інтерфейсу або серверного рішення. Довідкова реалізація GraphQL написана на JavaScript, тому використання GraphQL у Angular, Vue, Express, Hapi, Koa та інших бібліотеках JavaScript на стороні клієнта та сервера можливе, і це лише екосистема JavaScript. GraphQL імітує інтерфейс програмування мовою-агностиком REST між двома об'єктами, такими як клієнт або сервер.
    4. Популярність серед провідних компаній - Facebook є провідною компанією, що стоїть за специфікацією GraphQL та реалізацією посилання в JavaScript, але інші відомі компанії також використовують її для своїх додатків. Вони інвестуються в екосистему GraphQL завдяки величезному попиту на сучасні додатки. Крім Facebook, GraphQL також використовують ці відомі компанії:
       1. GitHub
       2. Shopify
       3. Twitter
       4. Coursera
       5. Yelp
       6. Wordpress
       7. The New York Times
       8. Samsara
       9. та ін.
    5. Єдине джерело істини - схема GraphQL є єдиним джерелом істини в додатках GraphQL. Він забезпечує центральне розташування, де описані всі доступні дані. Схема GraphQL зазвичай визначається на стороні сервера, але клієнти можуть читати (query) і записувати (mutation) дані на основі схеми. По суті, сервер пропонує всю інформацію про те, що доступне на його стороні, а клієнт запитує її частину, відправляючи запити GraphQL, або змінює її частину за допомогою мутацій GraphQL.
    6. Охоплює сучасні тенденції - GraphQL охоплює сучасні тенденції щодо побудови програм. Можна мати лише одну серверну програму, але декілька клієнтів у мережі, телефони та смарт-годинники залежно від їх даних. GraphQL може використовуватися для виконання вимог кожного клієнта - вимог до використання мережі, вкладених відносин даних, вилучення лише необхідних даних - без виділеного API для кожного клієнта. На стороні сервера може бути один сервер, а також група мікросервісів, які пропонують свої специфічні функціональні можливості. Це визначає ідеальне використання зшивання схем GraphQL, що дозволяє об'єднати всі функціональні можливості в одну схему GraphQL.
    7. Зшивання схем GraphQL - Зшивання схеми дозволяє створити одну схему з декількох схем. Подумайте про архітектуру мікросервісів для вашого бекенда, де кожна мікросервіс обробляє бізнес-логіку та дані для певного домену. У цьому випадку кожна мікросервіс може визначити свою власну схему GraphQL, після чого ви будете використовувати схеми зшивання, щоб переплетення їх у той, який доступний клієнту. Кожна мікросервіс може мати свою власну кінцеву точку GraphQL, де один шлюз API GraphQL об'єднує всі схеми в одну глобальну схему.
    8. Інтроспекція GraphQL - інтроспекція GraphQL дозволяє отримати схему GraphQL з API GraphQL. Оскільки схема має всю інформацію про дані, доступні за допомогою API GraphQL, вона ідеально підходить для автоматичного створення документації API. Вона також може бути використана для макету клієнта на схемі GraphQL для тестування або отримання схем з декількох мікросервісів під час зшивання схеми.
    9. Сильно типізований GraphQL - GraphQL - це сильно типізована мова запитів, оскільки вона написана у виразній мові визначень схем GraphQL (SDL). Будучи сильно типізованим, GraphQL менше може бути схильним до помилок, може бути перевірений під час компіляції і може бути використаний для підтримки підтримуваних інтеграцій IDE / редактора, таких як автозаповнення та перевірка.
    10. Версії GraphQL - у GraphQL не існує версій API, як це було в REST. У REST звичайно пропонують кілька версій API (наприклад, api.domain.com/v1/, api.domain.com/v2/), оскільки ресурси або структура ресурсів можуть змінюватися з часом. У GraphQL можна знехтувати API на рівні поля. Таким чином, клієнт отримує попередження про відхилення, коли запитує застаріле поле. Через деякий час застаріле поле може бути вилучено з схеми, коли його не використовують багато клієнтів. Це дає можливість з часом розвивати API GraphQL без необхідності версій.
    11. Зростаюча екосистема GraphQL - екосистема GraphQL зростає. Існують не тільки інтеграції для сильно типізованого характеру GraphQL для редакторів і IDE, але й автономні програми для самого GraphQL. Розвиваються не тільки технічні аспекти: існують конференції, зустрічі та спільноти для GraphQL, а також інформаційні бюлетені та подкасти.[18]

### *2.6.4 Apollo Server*

Apollo Server - це найкращий спосіб швидко створити готовий до виробництва, самодокументаційний API для клієнтів GraphQL, використовуючи дані з будь-якого джерела.

Він з відкритим вихідним кодом і прекрасно працює як окремий сервер, аддон до існуючого HTTP-сервера Node.js, або в середовищі без сервера.

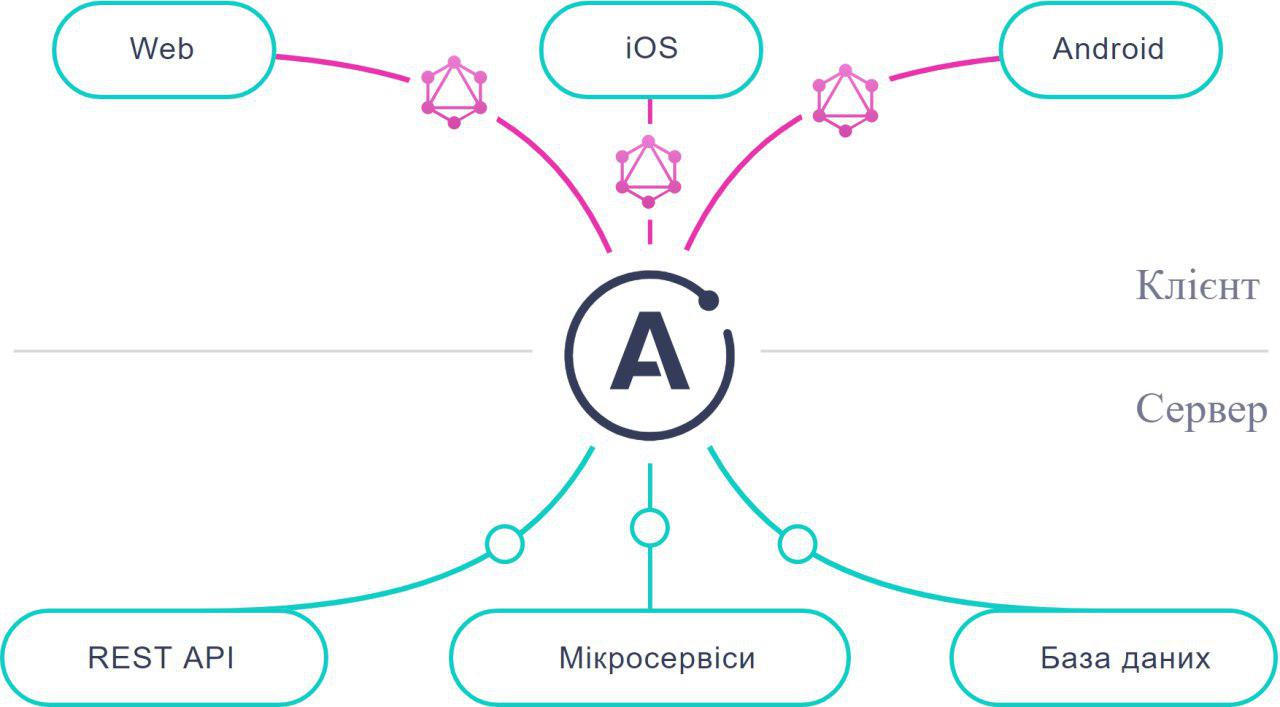


Рисунок 2.3. – Структура взаємозв’язку серверу та клієнту з використанням GraphQL

Apollo Server реалізує специфічний сервер GraphQL, який можна запитувати з будь-якого клієнта GraphQL, включаючи клієнта Apollo, що дозволяє:

* + 1. Легкий запуск, тому розробники інтерфейсу та інтерфейсу можуть швидко отримувати дані.
    2. Інкрементне прийняття, що дозволяє додавати додаткові функції, коли вони потрібні.
    3. Універсальна сумісність з будь-яким джерелом даних, будь-яким інструментом збирання та будь-яким клієнтом GraphQL.
    4. Готовність виробництва, і те, що будується в розробці, чудово працює у виробництві.[19]

Apollo Server забезпечує простий спосіб для нових або існуючих додатків швидко працювати. Існуючі програми можуть скористатися проміжним програмним забезпеченням, а нові програми можуть використовувати інтегрований веб-сервер. Обидва ці сервери можуть бути налаштовані з мінімальною конфігурацією та слідувати найкращим практикам розробки.

### *2.6.5 MongoDB*

MongoDB - документоорієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних.

Сьогодні переваги баз даних NoSQL не є таємницею, особливо коли широко використовуються хмарні обчислення.

Організації все частіше використовують бази даних NoSQL у відповідь на складність і обмеження традиційних реляційних баз даних. Бази даних NoSQL є більш масштабованими, можуть допомогти вам досягти кращої продуктивності і пропонує більш економічний спосіб розробки, впровадження та спільного використання програмного забезпечення.

Ключові переваги NoSQL включають:

* + 1. Ефективну архітектуру, яку можна масштабувати замість монолітної архітектури;
    2. Можливість обробки великих обсягів структурованих, напівструктурованих і неструктурованих даних;
    3. Краще узгодження з об'єктно-орієнтованим програмуванням;
    4. Гарну співпрацю з сучасними методологіями розробки програмного забезпечення;

Компанії вибирають MongoDB для розробки своїх сучасних додатків, оскільки це єдина база даних, яка дозволяє користувачам підтримувати переваги реляційних баз даних, включаючи інновації баз даних NoSQL. Бази даних NoSQL також мають тенденцію бути з відкритим вихідним кодом, а це означає відносно дешевий спосіб розробки, впровадження та спільного використання програмного забезпечення.[20]

### *2.6.6 Mongoose*

Mongoose - бібліотека моделювання об'єктів (ODM) для MongoDB і Node.js. Вона керує відносинами між даними, забезпечує перевірку схеми і використовується для перекладу між об'єктами в коді і поданням цих об'єктів в MongoDB.

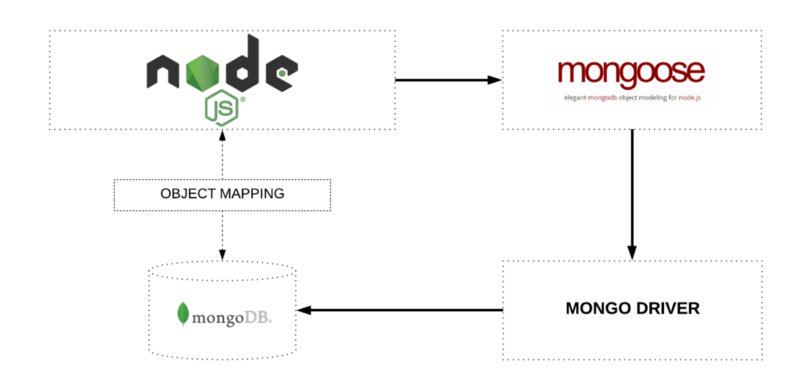


Рисунок 2.4. – Об'єктне об'єднання між Node і MongoDB, керованим через Mongoose

MongoDB - це база даних документів NoSQL. Це означає, що ви можете зберігати в ній документи JSON, а структура цих документів може змінюватися, оскільки вона не застосовується, як бази даних SQL. Це одна з переваг використання NoSQL, оскільки вона прискорює розробку додатків і зменшує складність розгортання.

Нижче наведено приклад того, як дані зберігаються в базі даних Mongo проти SQL:

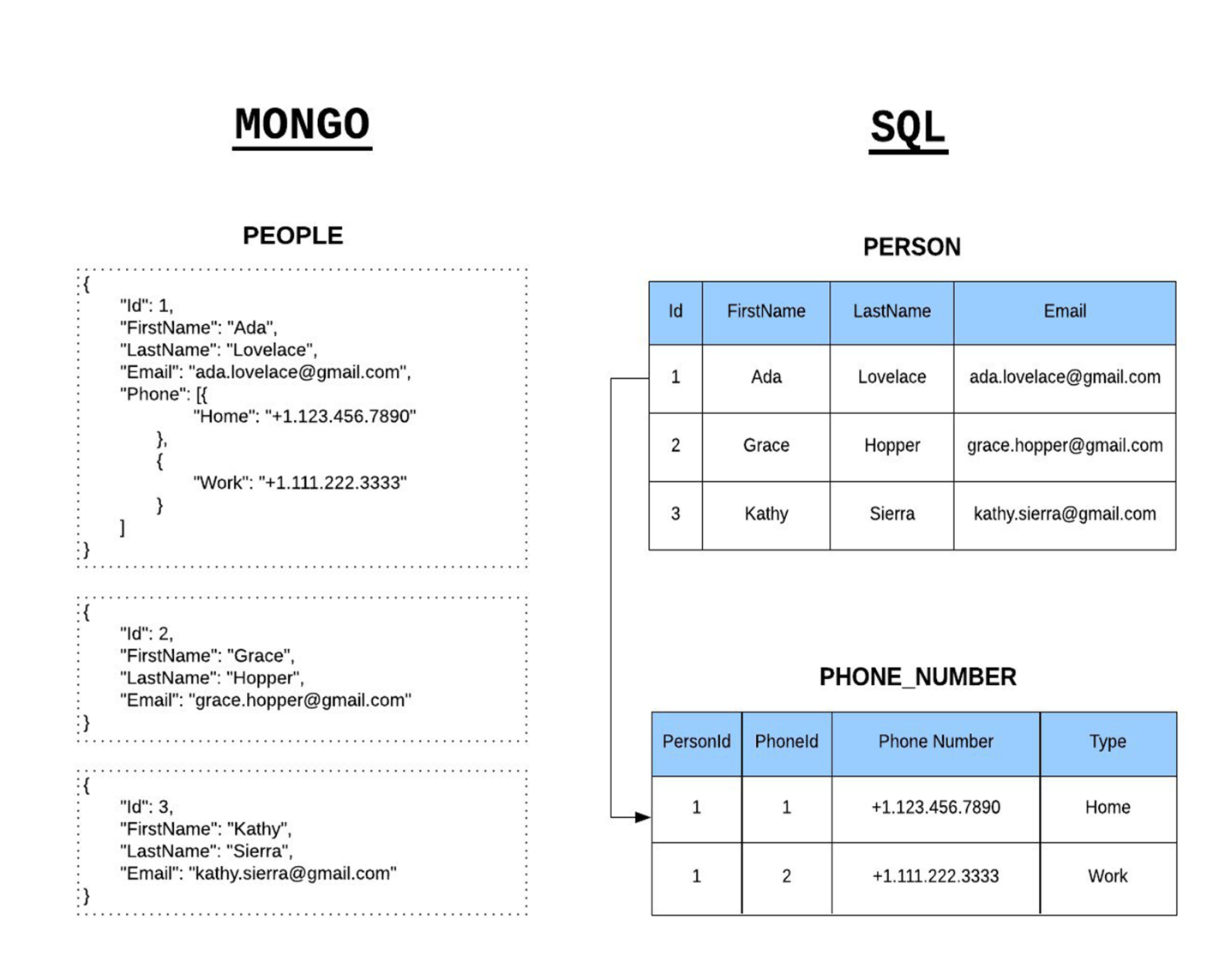


Рисунок 2.5. – Документи NoSQL проти реляційних таблиць в SQL

«Колекції» в MongoDB еквівалентні таблицям у реляційних базах даних. Вони можуть містити кілька документів JSON. «Документи» еквівалентні записам або рядкам даних у SQL. Хоча рядок SQL може посилатися на дані в інших таблицях, документи Mongo зазвичай поєднують це в документі. «Поля» або атрибути подібні до стовпців у таблиці SQL.

Хоча Mongo є схематичним, SQL визначає схему за допомогою визначення таблиці. «Схема» Mongoose - це структура даних документу (або форма документа), яка застосовується через прикладний рівень. «Моделі» - це конструктори вищого порядку, які беруть схему і створюють екземпляр документа, еквівалентний записам у реляційній базі даних.

Mongoose модель - обгортка на Mongoose схемі. Схема Mongoose визначає структуру документа, значення за замовчуванням, валідатори і т.д., тоді як модель Mongoose надає інтерфейс до бази даних для створення, запитів, оновлення, видалення записів тощо.[21]

Три головні переваги використання Mongoose проти рідного MongoDB:

* + 1. MongooseJS надає шар абстракції поверх MongoDB, що виключає необхідність використання іменних колекцій.
    2. Моделі в Mongoose виконують основну частину роботи зі встановлення значень за замовчуванням для властивостей документа і перевірки даних.
    3. Функції можуть бути прикріплені до моделей у MongooseJS. Це дає можливість безперешкодного впровадження нових функціональних можливостей.

Запити використовують функціональне прив'язування, а не вбудовані мнемоніки, які призводять до коду, який є більш гнучким і читабельним, а отже, і більш придатним до можливих змін.[22]

# РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ НАВЧАЛЬНОЇ ПЛАТФОРМИ

В теоретичній частині були проаналізовані існуючі навчальні платформи та вказані їх функціонал та недоліки. Було сформовано список вимог як до веб-додатку навчальної платформи, так і до його серверної частини. Були створені сутності, що відображають структуру даних, та розглянута бізнес-логіка серверу веб-доданку. Був представлений список технологій реалізації серверної частини веб-додатків та приведені їх переваги над аналогами.

Практична частина присвячена розробці серверної частини веб-додатку «NodeUA».

Розробка серверної частини веб-додатку розпочалась зі створення репозиторію на сайті GitHub.[23]

Для розробки серверу веб-доданку було обране середовище розробки Visual Studio Code, яке є дуже зручним у використанні та має плагіни до багатьох мов програмування і вбудовану систему контролю версій.

Інтерфейс програми Visual Studio Code зображений на рисунку 3.1.

Код розроблявся на мові JavaScript. В якості менеджера пакетів був обраний npm(Node Package Manager). Серверна частина була ініціалізована за допомогою команди в терміналі:

npm init – після чого був створений файл package.json, в якому вказані назва проекту, опис, версія, автор, ліцензія, прописуються залежності (встановлені за допомогою npm пакети), а також скрипти, що запускають сервер.

Після ініціалізації серверної частини проекту були встановлені наступні залежності:

* + 1. apollo-server-express – пакет для роботи з Apollo Server;
    2. bcrypt – пакет для хешування паролів;
    3. dotenv – пакет для використання змінних оточення;
    4. express – пакет для роботи з Express.js;
    5. graphql – пакет для роботи з GraphQL;
    6. jsonwebtoken – пакет для роботи з автентифікацією та секретними ключами;
    7. mongoose – пакет для роботи з Mongoose.

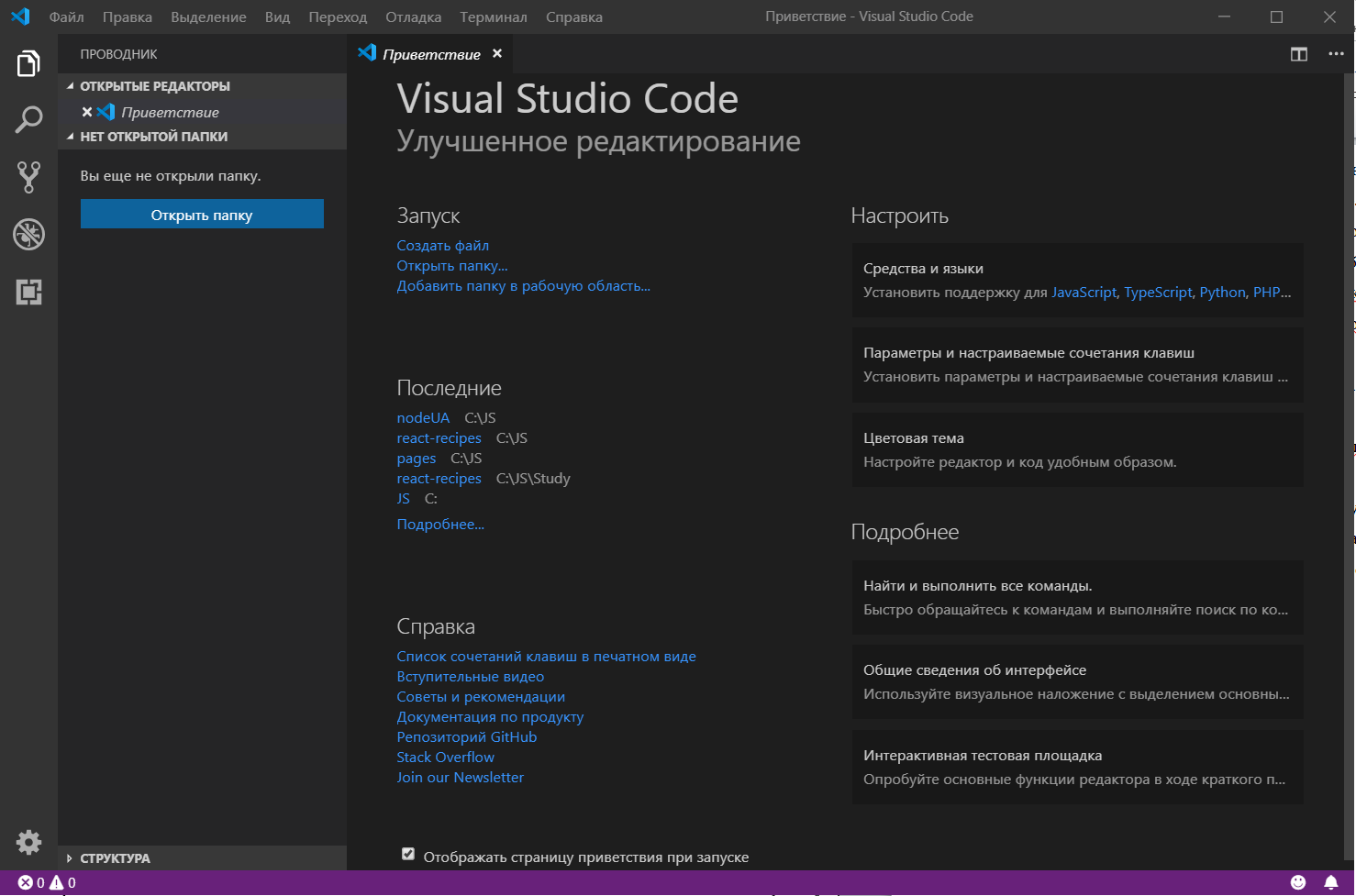


Рисунок 3.1. – інтерфейс середовища розробки Visual Studio Code

Для реалізації бази даних було вирішено обрати хмарний варіант MongoDB – MongoDB Atlas, від розробників MongoDB.

Даний варіант має істотну перевагу – доступ до бази даних можна отримати з будь-якого комп’ютера що відповідає наступним вимогам:

* + 1. Має доступ до мережі Інтернет;
    2. Змінні оточення проекту мають посилання на базу даних з відповідним логіном та паролем;
    3. IP-адреса комп’ютера внесена до білого списку IP-адрес бази даних.
    4. Інтерфейс сервісу MongoDB Atlas зображений на рисунку 3.2.

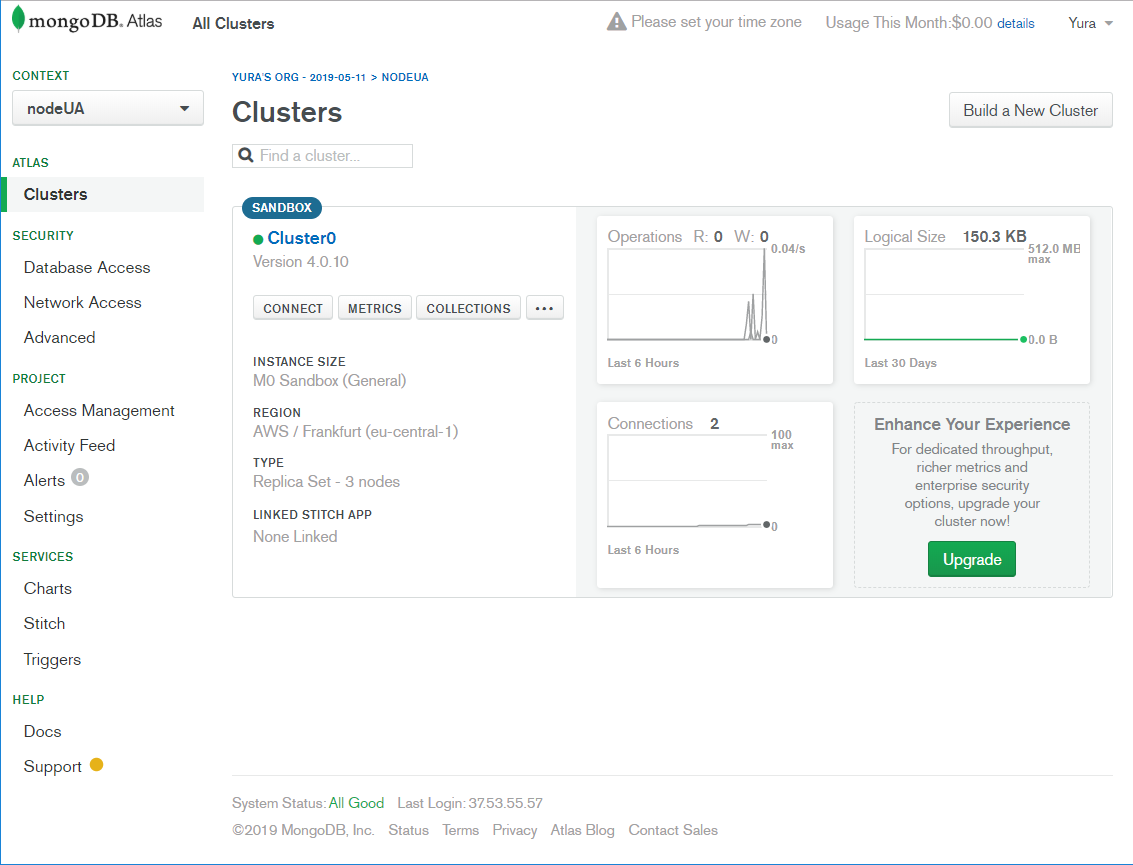


Рисунок 3.2. – Інтерфейс сервісу MongoDB Atlas.

Структура серверної частини веб-додатку зображена на рисунку 3.3.

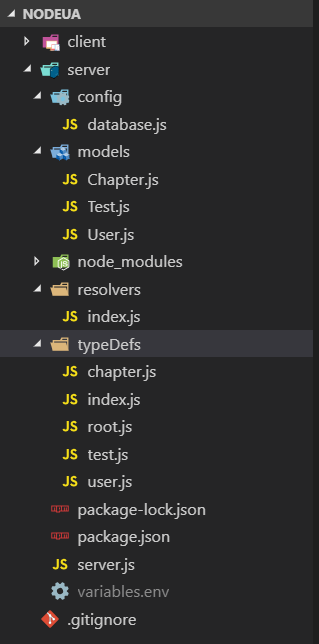


Рисунок 3.3. – Структура серверної частини веб-додатку

Окрім файла package.json, директорія server, що є кореневою для серверної частини, містить наступні файли та директорії:

* + 1. variables.env - містить усі необхідні для сервера змінні оточення;
    2. server.js - є точкою входу для сервера і де розташована головна логіка роботи серверної частини веб-додатка;
    3. node\_modules – директорія, що містить пакети, які були встановлені npm;
    4. config – директорія для конфігуруючи файлів;
    5. models – директорія для об’єктних моделей Mongoose бази даних MongoDB;
    6. resolvers – директорія для GraphQL резолверів;
    7. typeDefs – директорія для схем GraphQL.

Директорія config містить файл database.js, що відповідає за підключення бібліотеки моделювання об'єктів Mongoose до бази даних MongoDB.

Директорія models містить наступні файли:

* + 1. Chapter.js – містить Mongoose схему статті;
    2. Test.js - містить Mongoose схему тесту;
    3. User.js - містить Mongoose схему користувача.

Директорія resolvers містить файл index.js, у якому зосереджена реалізація бізнес логіки серверу.

Директорія typeDefs містить наступні файли:

* + 1. root.js - містить корені типів запитів;
    2. chapter.js – містить GraphQL схему статті та пов’язані з нею типи запитів;
    3. test.js – містить GraphQL схему тесту та пов’язані з нею типи запитів;
    4. user.js – містить GraphQL схему користувача та пов’язані з нею типи запитів;
    5. index.js – слугує для збору усіх GraphQL схем.

### 3.1. Реалізація бази даних

Пакет Mongoose був використаний для створення Mongoose схем, що відповідають сутностям спроектованої бази даних. Важливо вказати, що Mongoose самостійно надає збереженим об’єктам унікальний ключ (id), що значно спрощує реалізацію спроектованих раніше сутностей.

Mongoose схема ChapterSchema реалізує поля сутності Стаття, а саме її назву та уніфікований ідентифікатор ресурсу завдяки полям title та uri;

Реалізація схеми ChapterSchema (файл Chapter.js):

const Schema = mongoose.Schema;

// #2 Instantiate a schema using mongoose Schema

const ChapterSchema = new Schema({

title: {

type: String,

required: true

},

uri: {

type: String,

required: true

}

});

Аналогічно до схеми Chapter схема UserSchema реалізує поля сутності Користувач.

Реалізація схеми UserSchema (файл User.js):

const UserSchema = new Schema({

username: {

type: String,

required: true,

unique: true

},

password: {

type: String,

required: true

},

email: {

type: String,

required: true

},

joinDate: {

type: Date,

default: Date.now

},

role: {

type: String

},

testResults: {

type: [TestResultSchema],

}

});

Варто відмітити, що поле testResults даної схеми представляє собою масив схем TestResultSchema. Такий тип схем називається піддокументами. Піддокументи схожі зі звичайними документами, але вони не можуть бути збереженими у базі даних без збереження батьківського документу. Даний тип документів зберігається тільки якщо був викликаний метод збереження батьківського документу.

Сама схема TestResultSchema має поля, які відповідають характеристичній сутності Результат тесту:

const TestResultSchema = new Schema({

testName: {

type: String

},

results: {

type: [Boolean]

},

score: {

type: Number

}

})

Схема UserSchema має проміжну функцію (middleware), що виконується перед записом екземпляра користувача до бази даних:

UserSchema.pre('save', function(next) {

if (!this.isModified('password')) {

return next();

}

bcrypt.genSalt(10, (err, salt) => {

if (err) return next(err);

bcrypt.hash(this.password, salt, (err, hash) => {

if(err) return next(err);

this.password = hash;

next();

})

})

})

Дана функція викликається кожний раз перед збереженням даних користувача. Вона перевіряє чи було змінено поле паролю і якщо так, то за допомогою пакету bcrypt генерується сіль, яка приймає участь у хешуванні паролю і передає контекст виконання наступній проміжній функції.

Схема TestSchema реалізує поля сутності Тест:

const TestSchema = new Schema({

title: {

type: String,

required: true

},

questions: [QuestionSchema]

});

Поле questions є масивом схем QuestionSchema, а схема QuestionSchema є піддокументом для TestSchema. QuestionSchema являє собою реалізацію сутності Питання:

const QuestionSchema = new Schema({

questionText: {

type: String,

required: true

},

answers: [AnswerSchema]

});

Поле answers є масивом схем AnswerSchema, яка є піддокументом для схеми QuestionSchema. Схема AnswerSchema відповідає спроектованій сутності Відповідь:

const AnswerSchema = new Schema({

answerText: {

type: String,

required: true

},

isValid: {

type: Boolean,

required: true

},

isChecked: {

type: Boolean,

required: true,

default: false

}

});

Таким чином, всі сутності та взаємодії між ними були успішно реалізовані у базі даних.

### 3.2. Ініціалізація сервера

Після створення моделей даних, всі моделі та необхідні пакети були підключені до головного файлу сервера:

// #1 Import Express and Apollo Server

const express = require("express");

require("dotenv").config({ path: "variables.env" });

const { ApolloServer } = require("apollo-server-express");

const jwt = require('jsonwebtoken');

// #2 Import mongoose schemas

const mongoose = require("./config/database");

const Chapter = require('./models/Chapter');

const User = require('./models/User');

const Test = require('./models/Test');

// #3 Import GraphQL type definitions

const typeDefs = require("./typeDefs");

Наступним завданням була ініціалізація Apollo Server:

const server = new ApolloServer({

typeDefs,

resolvers,

context: ({ req: { currentUser } }) => {

return { Chapter, User, Test, currentUser }

}

});

Конструктор ApolloServer приймає наступні аргументи:

* + 1. typeDefs – graphQL схеми;
    2. resolvers – функції-резолвери запитів;
    3. context – контекст виконання.

context відповідає за передачу контексту виконання для функцій-резолверів запитів. Вказані в даному полі моделі можуть використовуватися у функціях обробки запитів. В даному випадку в поле context повертаються Mongoose моделі та поточний користувач currentUser.

Після ініціалізації сервера Apollo було ініціалізовано сервер express, який є основою серверної частини веб-додатку:

const app = express();

Наступним кроком були ініціалізовані опції для зв’язку клієнта та сервера, де в поле origin записується адреса клієнтської частини веб-додатк:

const corsOptions = {

origin: "http://localhost:3000",

credentials: true

};

Після цього була написана проміжна функція автентифікації користувача за допомогою пакета jsonwebtoken:

app.use(async (req, res, next) => {

const token = req.headers.authorization;

//console.log(token, typeof token);

if (token !== 'null' && token !== undefined) {

try {

const currentUser = await jwt.verify(token, process.env.SECRET);

req.currentUser = currentUser;

} catch (err) {

console.error(err);

}

}

next();

})

Дана функція приймає запит, з заголовку якого отримується тимчасовий ключ. Якщо значення ключа порожнє або ключ взагалі не існує, функція повертає помилку. Якщо ключ є, він проходить верифікацію за допомогою методу verify пакета jsonwebtoken, після чого дані поточного користувача передаються далі.

Останніми кроками перед запуском серверу були додавання опцій для зв’язку з клієнтом, підключення express сервера як проміжного програмного забезпечення, встановлення порту та запуску прослуховування порту:

server.applyMiddleware({ app, cors: corsOptions });

const port = process.env.PORT || 4444;

app.listen(port, () => {

console.log(

`Server running on http://localhost:${port}${server.graphqlPath}`

);

});

### 3.3. Реалізація взаємодії з клієнтом

Для взаємодії з клієнтською частиною веб-додатку був використаний GraphQL.

Згідно з бізнес-логікою був реалізований наступний функціонал:

* + 1. Реєстрація нового користувача;
    2. Авторизація існуючого користувача;
    3. Створення тесту;
    4. Видалення тесту;
    5. Додавання статті;
    6. Перевірка пройденого користувачем тесту;
    7. Відправлення списку статей;
    8. Відправлення статті;
    9. Відправлення списку тестів;
    10. Відправлення тесту.

Для роботи з запитами були реалізовані наступні graphQL типи даних:

* + 1. Type Chapter
    2. Type User
    3. Type Test
    4. Type Question
    5. Type Answer

GraphQL розмежує звичайні типи даних та типи введення, що дає можливість спростити схему. В даній реалізації використовуються наступні типи введення:

* + 1. input AnswerInput
    2. input QuestionInput

Типи введення є дуже корисними у випадках великої вкладеності аргументів запиту.

Всі типи GraphQL схем відповідають їх аналогам з Mongoose.

GraphQL типи з файлу typeDefs/test.js:

type Answer {

\_id: ID

answerText: String!

isValid: Boolean!

isChecked: Boolean!

}

input AnswerInput {

answerText: String!

isValid: Boolean!

isChecked: Boolean!

}

type Question {

\_id: ID

questionText: String!

answers: [Answer]!

}

input QuestionInput {

questionText: String!

answers: [AnswerInput]!

}

type Test {

\_id: ID

title: String!

questions: [Question]!

}

GraphQL типи з файлу typeDefs/chapter.js:

type Chapter {

\_id: ID

title: String

uri: String

}

GraphQL типи з файлу typeDefs/user.js:

type TestResult {

\_id: ID

testName: String

results: [Boolean]

score: Float

}

type User {

\_id: ID

username: String!

password: String!

email: String!

joinDate: String

role: String

testResults: [TestResult]

}

type Token {

token: String!

}

Тип Token – це тимчасовий ключ доступу, що видається при успішній реєстрації або авторизації.

Реєстрація нового користувача реалізована GraphQL мутацією signupUser, що при успішному завершенні повертає тимчасовий ключ доступу (token).

Файл typeDefs/user.js:

signupUser(username: String!, email: String!, password: String!): Token

Файл resolvers/index.js:

signupUser: async (root, { username, email, password }, { User }) => {

const user = await User.findOne({ username: username });

if (user) {

throw new Error('User already exists');

}

const newUser = await new User({

username,

email,

password,

role: 'user',

testResults: []

}).save();

return { token: createToken(newUser, process.env.SECRET, '24hr') }

},

Даний резолвер мутації signupUser отримує три аргументи: ім’я користувача, електронну пошту користувача та пароль. Функція звертається до Mongoose моделі User та виконує за допомогою методу моделі пошук у базі даних користувача з відповідним ім’ям. Якщо користувач знаходиться – функція повертає помилку, що користувач с таким ім’ям вже існує. Якщо користувача з таким ім’ям немає, функція створює нового користувача з відповідними полями та викликає функцію createToken, результат виконання якої відправляє до клієнту.

Функція createToken приймає три аргументи: об’єкт з даними користувача, змінну оточення, за допомогою якої відбувається генерація токена, та дату закінчення дії токену:

const createToken = (user, secret, expiresIn) => {

const { username, email } = user;

return jwt.sign({ username, email }, secret, { expiresIn });

}

Авторизація існуючого користувача реалізована GraphQL мутацією signinUser, що при успішному завершенні повертає тимчасовий ключ доступу (token).

Файл typeDefs/user.js:

signinUser(username: String!, password: String!): Token

Файл resolvers/index.js:

signinUser: async (root, { username, password }, { User }) => {

const user = await User.findOne({ username: username });

if (!user) {

throw new Error('User not found');

}

const isValidPassword = await bcrypt.compare(password, user.password);

if (!isValidPassword) {

throw new Error('Invalid password');

}

return { token: createToken(user, process.env.SECRET, '24hr') }

},

Даний резолвер мутації signinUser отримує два аргументи: ім’я користувача та пароль. Функція звертається до Mongoose моделі User та виконує за допомогою методу моделі пошук у базі даних користувача з відповідним ім’ям. Якщо користувач не знаходиться – функція повертає помилку, що користувач не знайдений. Якщо користувач з таким ім’ям є, функція перевіряє пароль на правильність за допомогою методу compare пакета bcrypt. Якщо пароль невірний функція повертає помилку з текстом, що пароль невірний. Якщо пароль – вірний, функція викликає функцію createToken, результат виконання якої відправляє до клієнту.

Створення тесту реалізоване мутацією addTest, що при успішному завершенні повертає до клієнту новий тест.

Файл typeDefs/test.js:

addTest(questions: [QuestionInput]!, title: String!): Test

Файл resolvers/index.js:

addTest: async (root, { questions, title }, { Test }) => {

const newTest = await new Test({ title, questions }).save();

return newTest;

}

Даний резолвер отримує два аргументи: масив відповідей та назву тесту, після чого звертається до Mongoose моделі Test та виконує за допомогою методу моделі запис тесту у базу даних. Після успішного запису функція повертає записаний тест до клієнту.

Додавання статті відбувається аналогічним шляхом:

Файл typeDefs/chapter.js:

addChapter(title: String!, uri: String!): Chapter

Файл resolvers/index.js:

addChapter: async (root, { title, uri }, { Chapter }) => {

const newChapter = await new Chapter({ title: title, uri: uri }).save();

return newChapter;

}

Видалення тесту реалізоване мутацією deleteTest, що при успішному виконанні повертає видалений тест.

Файл typeDefs/test.js:

addChapter(title: String!, uri: String!): Chapter

Файл resolvers/index.js:

deleteTest: async (root, {\_id}, {Test}) => {

const test = await Test.findOne({\_id: \_id});

if(!test) {

throw new Error('Test not found');

}

const deletedTest = await Test.findByIdAndDelete(\_id);

// console.log(deletedTest);

return deletedTest;

}

Даний резолвер отримує унікальний ключ тесту, після чого звертається до Mongoose моделі Test та виконує пошук тесту. Якщо тест не знайдений, функція повертає помилку, що тест не знайдено. Якщо тест знайдений, він видаляється з бази даних та повертається до клієнту.

Перевірка пройденого тесту реалізована мутацією checkTest, яка отримує унікальний ключ користувача, назву тесту та питання з відміченими відповідями. Після цього в новий масив записуються результати перевірки на правильність обраних відповідей. Далі, якщо тест не був виконаний даним користувачем раніше, результат тесту записується до інших. Якщо тест вже був виконаний раніше, новий результат тесту замінює старий, після чого поточні результати повертаються до клієнта.

Мутація checkTest

checkTest: async (root, { \_id, title, questions }, { User }) => {

const results = [];

questions.map(question => {

question.answers.map(answer => {

if (answer.isValid) {

if (answer.isChecked) {

return results.push(true);

} else {

return results.push(false);

}

}

})

});

let isNewResult = true;

let id;

if (testResults.length === 0) {

isNewResult = true;

} else {

testResults.map(testResult => {

if (testResult.testName === title) {

isNewResult = false;

id = testResult.\_id;

}

});

}

let testResult = {

"results": results,

"testName": title,

};

if (isNewResult) {

const updatedUser = await User.findByIdAndUpdate({ \_id }, { "$addToSet": { "testResults": testResult } }, { "new": true }, function (err) {

if (err) throw new Error(err)

});

console.log("user after update(isNewResult=true):", updatedUser);

} else if (!isNewResult) {

const updatedUser = await User.findOneAndUpdate({ 'testResults.\_id': id }, { '$set': { 'testResults.$.results': results, 'testResults.$.score': score } }, { "new": true }, function (err) {

if (err) throw new Error(err)

console.log("Successfully saved!");

});

console.log("user after update(isNewResult=false):", updatedUser);

}

return { results };

}

# ВИСНОВКИ

В наш час веб-додатки з клієнт-серверною архітектурою дуже поширенимі. Їх популярність обумовлюється мультиплатформністю та легкою масштабованістю.

У ході виконання дипломної роботи були зроблені наступні задачі:

В першому розділі була розглянута актуальність навчальних платформ, проаналізовані існуючі рішення та їх недоліки, висунуті вимоги до навчальної платформи.

В другому розділі була освітлена клієнт-серверна архітектура веб-додатку, сформовані вимоги до серверної частини, побудована інформаційна модель, спроектована бізнес-логіка та проаналізовані технології реалізації серверу.

В третьому розділі була освітлена реалізація бази даних, ініціалізований сервер та створені механізми взаємодії з клієнтом.

В результаті отримана функціонуюча серверна частина веб-додатку навчальної платформи та реалізовані усі вимоги, поставлені до розробки серверу.

# СПИСОК ЛІТЕРАТУРИ

1. Онлайн-образование: 7 ключевых трендов 2018 года [Електронний ресурс]. - Режим доступу: https://geniusmarketing.me/lab/onlajn-obrazovanie-7-klyuchevyx-trendov-2018-goda/. - Дата звернення: 18.05.2019 р.

2. Web Application. [Електронний ресурс]. - Режим доступу: https://techterms.com/definition/web\_application. - Дата звернення: 19.05.2019 р.

3. Веб-застосунок [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Веб-застосунок. - Дата звернення: 19.05.2019 р.

4. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс]. - Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів -9893d8048229. - Дата звернення: 19.05.2019 р.

5. Термін «Інформаційна система» [Електронний ресурс]. - Режим доступу: https://zakon.rada.gov.ua/laws/term/11471:56668. - Дата звернення: 20.05.2019 р.

6. Модель «сутність — зв'язок» [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Модель\_«сутність\_—\_зв'язок». - Дата звернення: 20.05.2019 р.

7. Інформатика [Електронний ресурс]. - Режим доступу: http://schoollib.com.ua/dovidnyk\_shkolyara/informatyka/7.html. - Дата звернення: 21.05.2019 р.

8. Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів [Електронний ресурс]. - Режим доступу: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/#q03. - Дата звернення: 21.05.2019 р.

9. Бизнес-логика [Електронний ресурс]. - Режим доступу: https://ru.wikipedia.org/wiki/Бизнес-логика. - Дата звернення: 21.05.2019 р.

10. Node.js [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Node.js. - Дата звернення: 23.05.2019 р.

11. What is the difference between PHP and Node.js? [Електронний ресурс]. - Режим доступу: https://www.websoptimization.com/blog/what-is-the-difference-between-php-and-node-js/. - Дата звернення: 24.05.2019 р.

12. Express/Node introduction [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\_Nodejs/Introduction. - Дата звернення: 24.05.2019 р.

13. Server-side web frameworks [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\_steps/Web\_frameworks. - Дата звернення: 25.05.2019 р.

14. Express/Node introduction [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\_Nodejs/Introduction. - Дата звернення: 27.05.2019 р.

15. Introduction to GraphQL [Електронний ресурс]. - Режим доступу: https://graphql.org/learn/. - Дата звернення: 27.05.2019 р.

16. Schemas and Types [Електронний ресурс]. - Режим доступу: https://graphql.org/learn/. - Дата звернення: 27.05.2019 р.

17. Queries and Mutations [Електронний ресурс]. - Режим доступу: https://graphql.org/learn/queries/. - Дата звернення: 27.05.2019 р.

18. Why GraphQL: Advantages, Disadvantages & Alternatives [Електронний ресурс]. - Режим доступу: https://www.robinwieruch.de/why-graphql-advantages-disadvantages-alternatives/. - Дата звернення: 30.05.2019 р.

19. Introduction. What is Apollo Server and what does it do? [Електронний ресурс]. - Режим доступу: https://www.apollographql.com/docs/apollo-server/. - Дата звернення: 30.05.2019 р.

20. Benefits Of NoSQL [Електронний ресурс]. - Режим доступу: https://www.mongodb.com/scale/benefits-of-nosql. - Дата звернення: 01.06.2019 р.

21. Introduction to Mongoose for MongoDB [Електронний ресурс]. - Режим доступу: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/. - Дата звернення: 01.06.2019 р.

22. An Overview of MongoDB & Mongoose [Електронний ресурс]. - Режим доступу: https://medium.com/chingu/an-overview-of-mongodb-mongoose-b980858a8994. - Дата звернення: 01.06.2019 р.

23. NodeUA-Info/nodeUA [Електронний ресурс]. - Режим доступу: https://github.com/NodeUA-Info/nodeUA. - Дата звернення: 03.06.2019 р.