Лабораторна робота №5: Розробка серверної частини додатку та її розгортання

Мета: Навчитись розробляти серверну частину веб-додатків за допомогою Node.js та розгортати веб-додатки за допомогою хмарних сервісів.

Теоретичні відомості

Вступ до клієнт-серверної взаємодії

У сучасних веб-додатках **клієнт-серверний зв'язок** є основною концепцією, де клієнт (зазвичай браузер користувача або мобільний додаток) взаємодіє з сервером за допомогою запитів даних, виконання дій і відображення контенту. Ця взаємодія відбувається через мережу, а зв'язок між клієнтом і сервером зазвичай відбувається через **API (інтерфейси прикладного програмування)**.

Поширеною архітектурою побудови веб-додатків є **клієнт-серверна архітектура**, де клієнтська частина відповідає за користувацький інтерфейс, а сервер - за обробку, зберігання та пошук даних. Добре спроектована система типу «клієнт-сервер» забезпечує безперебійну взаємодію між інтерфейсом і сервером, дозволяючи користувачам отримувати, відображати та керувати даними.

Клієнт-серверний зв'язок за допомогою API

API - це набір правил і протоколів, які визначають, як повинні взаємодіяти різні програмні компоненти. У контексті розробки веб-додатків API зазвичай надає доступ до даних і сервісів на стороні сервера, доазволяючи інтерфейсу запитувати і отримувати дані динамічно, без необхідності перезавантажувати сторінку.

Найпоширенішою формою спілкування між клієнтом і сервером є **HTTP-запити** (GET, POST, PUT, DELETE тощо), які виконуються за допомогою RESTful API.

REST (Representational State Transfer - передача представницького стану) - це архітектурний стиль проектування мережевих додатків, в основі якого лежить комунікація без стану, де кожен запит до API містить всю інформацію, необхідну для обробки запиту.

Наприклад, коли користувач взаємодіє з веб-додатком, клієнту може знадобитися отримати такі дані, як список продуктів, інформацію про профіль користувача або оновлення в режимі реального часу. Цього можна досягти за допомогою HTTP-запитів до API на стороні сервера, який відповідає запитуваними даними у форматі **JSON** (JavaScript Object Notation), що полегшує обробку та відображення даних на клієнтській частині.

Розробка серверної чатини з використанням Node.js та Express

Node.js - це **середовище виконання JavaScript**, яке дозволяє розробникам запускати код JavaScript на стороні сервера, забезпечуючи повноцінну розробку JavaScript.

Express - фреймворк веб-додатків для Node.js, що спрощує процес створення API та обробки HTTP-запитів.

Разом Node.js та Express зазвичай використовуються для створення **серверів**, які обробляють вхідні API-запити з клієнтської частини додатків. Ці сервери можуть обробляти дані, взаємодіяти з базою даних і надавати відповідну інформацію.

Зазвичай сервер з використанням Node.js та Express працює наступним чином:

1. **Створення сервера Node.js**: Використовуючи середовище виконання Node.js, створюється сервер, який прослуховує вхідні HTTP-запити на певному порті.
2. **Визначення маршрутів**: У Express маршрути використовуються для зіставлення певних кінцевих точок з діями, такими як отримання даних або оновлення інформації. Наприклад, маршрут GET /products може повернути список продуктів з бази даних.
3. **Обробка запитів**: При отриманні запиту Express може обробляти дані, взаємодіяти з базою даних (наприклад, MongoDB, MySQL) і надсилати відповідь клієнту із запитуваними даними у форматі JSON.

Такі можливості дозволяють розробникам створювати API для управління даними та бізнес-логікою, які можуть бути використані інтерфейсними додатками, створеними за допомогою таких фреймворків, як React, Angular або Vue.

Інтеграція клієнтської та серверної частини додатку

У типовому додатку **клієнтська частина** (наприклад, React, Angular або Vue) робить HTTP-запити **до серверної частини** (наприклад, Node.js/Express або Firebase), щоб отримати або відправити дані.

Зазвичай інтеграція клієнтської та серверної частини виглядає наступним чином:

1. **Клієнтська частина додатку надсилає запити**: Фронтенд надсилає HTTP-запити (наприклад, GET, POST, PUT) до сервера або бази даних Firebase. Наприклад, коли користувач входить в систему або надсилає форму, дані надсилаються на сервер для обробки.
2. **Серверна частина додатку обробляє дані**: Сервер отримує запит, обробляє дані (наприклад, робить запит до бази даних або виконує бізнес-логіку) і формує відповідь.
3. **Серверна частина надсилає відповідь**: Серверна частина надсилає відповідь назад клієнту, зазвичай у форматі JSON, який фронтенд може легко проаналізувати та відобразити.
4. **Оновлення даних у клієнтській частині**: Інтерфейс користувача динамічно оновлюється для відображення нових даних, наприклад, для відображення списку елементів або підтвердження того, що форму було надіслано.

Взаємодія між клієнтом і сервером є основою сучасних веб-додатків. Використовуючи API для отримання даних, інтегруючи клієнтські та серверні частини, розробники можуть створювати адаптивні та динамічні веб-додатки.

Розгортання веб-додатків

Розгортання - надання додатку публічного доступу для користувачів. Розгортання передбачає розміщення додатку на хмарному сервісі або сервері, забезпечення його ефективної роботи та масштабування за потреби.

Хмарні сервіси спрощують розгортання веб-додатків, надаючи інструменти для автоматизації, масштабованості та безпеки. Для розгортання фронтенду та бекенду використовуються різні сервіси.

Хостинг клієнтської частини

1. **GitHub Pages** – безкоштовний сервіс для розгортання статичних сайтів (HTML, CSS, JS). Підходить для простих веб-додатків без бекенду.
    - Вимагає, щоб код був у публічному або приватному репозиторії GitHub.
    - Підтримує автоматичне оновлення при змінах у гілці main або gh-pages.
2. **Netlify** – більш гнучкий сервіс для хостингу статичних сайтів із можливістю автоматичних оновлень та інтеграції з API.
    - Підтримує розгортання додатків розташованиз у репозиторіях GitHub, GitLab, Bitbucket.
    - Дозволяє використовувати серверні функції для обробки запитів.
    - Проста інтеграція з різними доменними іменами сайтів та сертифікатами SSL.i

Розгортання Node.js сервера

1. **Render** – популярний сервіс для хостингу Node.js-бекенду з автоматичним процесом розгортання.
    - Підтримує автоматичне оновлення з репозиторію GitHub.
    - Дозволяє працювати з базами даних, середовищем змінних та логами.
    - Є безкоштовний тариф із певними обмеженнями.

Render зручний для розгортання Express.js API або повноцінного Node.js сервера.

Хостинг статичних файлів на сервері

Хостинг статичних файлів на сервері означає зберігання та обслуговування файлів, які не змінюються на сервері перед відправкою користувачу. До таких файлів належать HTML-документи, таблиці стилів CSS, JavaScript-код, зображення, шрифти та інші ресурси. Основна перевага статичного хостингу — це висока швидкість завантаження, оскільки сервер просто передає файли без обробки логіки.

Node.js є популярним вибором для розгортання статичних сайтів, особливо з використанням фреймворку Express. Express надає метод express.static(), який дозволяє зробити будь-яку папку доступною для веб-запитів. Це дає змогу організувати зручну файлову структуру та обслуговувати контент без зайвих витрат ресурсів сервера.

Окрім локального хостингу, такий підхід застосовується у розгортанні сайтів на хмарних платформах, таких як AWS S3, Vercel або Netlify. Використання CDN (Content Delivery Network) також підвищує продуктивність, кешуючи файли на серверах, розташованих ближче до користувача.

Крім того, статичні сайти можна поєднувати з API для отримання динамічних даних, що створює гібридний підхід між статичним і динамічним рендерингом. Безпека таких сайтів вища, оскільки немає серверної логіки, що зменшує ризик атак типу SQL-ін'єкцій або зламу серверних скриптів.

Порядок виконання роботи

Завдання 1: Створення серверної частини за допомогою Node.js

Опис:

Створити серверну частину за допомогою Node.js з Express. Налаштувати обробку запитів API від фронтенду та забезпечити взаємодію з базою даних для зберігання та отримання даних.

Кроки:

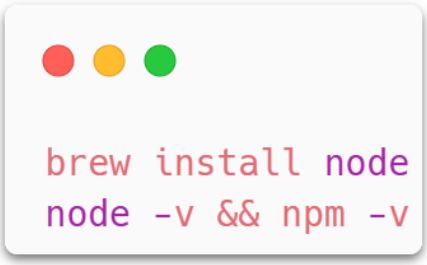
Встановлення Node.js та npm

Windows

1. Завантажити останню версію з nodejs.org
2. Запустити .msi файл інсталятора і провести встановлення
3. Ввести у командній строці (cmd) наступні команди:

MacOS

1. Через Homebrew (рекомендовано)



2. Через офіційний інсталятор

Завантажити .pkg з nodejs.org і встановити.

Linux (Ubuntu/Debian)

1. Ввести у Терміналі наступні команди:

```
curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -
sudo apt install -y nodejs
node -v && npm -v
```

Ініціалізація проекту Node.js

Відкрити термінал і створити нову папку проекту.

Перейти до папки і запустити наступну команду:

```
mkdir my-backend && cd my-backend
npm init -y
```

Встановити необхідні залежності:

```
npm install express cors
```

Налаштування Express-сервера

Створити новий файл server.js в корені проекту.

Додати наступний код для ініціалізації Express-сервера:

```
const express = require("express");
const cors = require("cors");

const app = express();
app.use(cors());
app.use(express.json());

app.listen(5000, () => {
    console.log("Server is running on port 5000");
});
```

Визначення основних маршрутів API

Всередині файлу server.js додати:

```
app.get("/api/message", (req, res) => {
    res.json({ message: "Hello from the backend!" });
});
```

Перевірити роботу сервера, запустивши його:

```
node server.js
```

Відкрити браузер і перейти за адресою http://localhost:5000/api/message. Має відобразитись відповідь:

{ "message": "Hello from the backend!" }

Інтеграція клієнтської та серверної частини

У клієнтській частині (фронтенді) виконати запит на отримання даних з серверної частини, вставивши наступний приклад в код одного із раніше створених компонентів фронтенд-фреймворку:

```
fetch("http://localhost:5000/api/message")
  .then(response => response.json())
  .then(data => console.log(data.message));
```

Даний приклад реалізує успішну взаємодію клієнтської та серверної частини.

Хостинг статичних файлів у Node.js з Express

Приклад реалізації хостингу статичних файлів у Node.js з Express наведений нижче:

```
const express = require('express');
const app = express();

app.use(express.static('public'));

app.listen(3000, () => {
    console.log('Server is running on http://localhost:3000');
});
```

У наведеному прикладі всі файли, що знаходяться у папці public, стають доступними за HTTP-запитами. Наприклад, файл public/index.html можна відкрити за адресою http://localhost:3000/index.html. Такий підхід дозволяє швидко налаштувати статичний хостинг без складної конфігурації.

Завдання 2: Робота з базою даних (Firebase Firestore)

Опис:

Інтеграція Firebase Firestore з серверною частиною для динамічного зберігання та отримання даних.

Кроки:

Налаштування Firebase Firestore

1. Зареєструватись/Увійти на Firebase (https://firebase.google.com/)
2. Перейти до консолі Firebase Console.
3. Вибрати проект (або створити новий, якщо ще не створили).
4. Натиснути на пункт Налаштування проекту → Облікові записи служб
5. Натиснути кнопку «Згенерувати новий приватний» ключ і завантажити JSON-файл.
6. Розмістити файл у директорії проекту як serviceAccountKey.json
7. Переконтись, що serviceAccountKey.json завантажено з Firebase і розміщено в корені проекту.
8. У розділі База даних Firestore створити нову базу даних у тестовому режимі.
9. Створити колекцію та деякі об'єкти даних (документи).

Встановлення Firebase SDK для роботи з аутентифікацією та базою даних

```
npm install firebase-admin
```

Налаштування Firebase на серверній частині

Створити файл firebaseConfig.js в корені проекту і додати у нього наступний код, змінивши значення полів на відповідні параметри, використані для конфігурації клієнтської частини:

```
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "your-project-id.firebaseapp.com",
  projectId: "your-project-id",
  storageBucket: "your-project-id.appspot.com",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID",
  measurementId: "YOUR_MEASUREMENT_ID"
};

export default firebaseConfig;
```

Всередині файлу server.js провести підключення бази даних за наступним прикладом коду:

```
const express = require("express");
const cors = require("cors");
const admin = require("firebase-admin"); // Initialize Firebase Admin SDK
const serviceAccount = require("./serviceAccountKey.json");
admin.initializeApp({ credential: admin.credential.cert(serviceAccount) });
const db = admin.firestore();
const app = express(); app.use(cors());
app.use(express.json());
app.listen(5000, () => { console.log("Server is running on port 5000"); });
```

Створення маршрутів API для отримання даних з бази даних

Оновити файл server.js:

```
// Fetch data from Firestore
app.get("/api/users", async (req, res) => {
    const snapshot = await
db.collection("users").get();
    const users = [];
    snapshot.forEach(doc => {
        users.push({ id: doc.id, ...doc.data() });
    });
    res.json(users);
});
```

Тестування API

Запустити серверну частину і використати GET і POST HTTP-запити для отримання даних за маршрутом /api/users, який можна використовувати для функціональності клієнтської частини.

Завдання 3: Створення захищеного маршруту серверної частини

У файлі server.js з реалізацією серверної частини додати новий захищений маршрут і код, пов'язаний з верифікацією користувача:

```
// Verify Firebase Auth Token (Middleware)
const verifyToken = async (req, res, next) => {
const token = req.headers.authorization?.split("Bearer ")[1];
 if (!token) {
return res.status(401).json({ message: "Unauthorized" });
}
const decodedToken = await admin.auth().verifyIdToken(token);
req.user = decodedToken;
next();
// Protected route (Only accessible with a valid token)
app.get("/api/protected", verifyToken, (req, res) => {
res.json({ message: "You have accessed a protected route!", user: req.user });
 });
```

Відповідь від сервера за маршрутом /api/protected отримають тільки автентифіковані користувачі.

Отримання даних використовуючи автентифікацію API

Реалізація запиту до захищеного маршруту, що включає інформацію про автентифікацію користувача виглядає наступним чином на клієнтській частині:

```
// Login
// Fetch Protected API Data
async function getProtectedData() {
    const user = auth.currentUser;
    if (!user) {
        alert("Please log in first.");
        return;
    }
    try {
        const token = await getIdToken(user);
        const response = await fetch("http://localhost:5000/api/protected", {
            headers: {
                Authorization: `Bearer ${token}`
            }
        });
        const data = await response.json();
        alert(JSON.stringify(data));
    } catch (error) {
        alert("Error fetching protected data: " + error.message);
    }
}
```

Завдання 4: Хостинг та розгортання

Розгортання клієнтської частини веб-додатку

Варіант 1. Сторінки GitHub

1. Відкрити GitHub (https://github.com/) і створити новий публічний репозиторій з назвою username.github.io, де username - це ваше ім'я користувача (нікнейм) на GitHub. Помістити всі файли створеного сайту в цю папку.

2. Відкрити браузер і перейти за адресою https://username.github.io, де username - це ваше ім'я користувача. Сайт доступний в Інтернеті після успішного розгортання.

Варіант 2. Netlify

1. Завантажити створений сайт у репозиторій GitHub
2. Зареєструйтесь на сайті https://www.netlify.com/ та перейти на панель управління
3. Натиснути кнопку "Додати новий сайт" та "Імпортувати існуючий проект"

4. Перейти до свого GitHub, авторизуватись у Netlify та вибрати сховище зі створеним веб-сайтом
5. Налаштувати параметри, обовёязково необхідно ввести назву сайту
6. Натиснути кнопку Розгорнути
7. Сайт доступний в Інтернеті після успішного розгортання

Розгортання серверної чатини веб-додатку

Варіант 1. Render

1. Зареєструватись на сайті https://render.com/ та перейти на сторінку дашборду
2. Відкрити розділ "Проекти" та натиснути кнопку «Додати новий»
3. Виберати опцію «Веб-сервіс»
4. Підключити репозиторій GitHub з кодом на стороні сервера як джерело, натиснути Підключити
5. Встановити такі параметри проекту, як команда збірки (npm i) та команда запуску (node server.js)
6. Вибрати тип пакету послуг - Безкоштовний
7. Натиснути кнопку Розгорнути веб-службу
8. Зачекати на процес розгортання
9. Клікнути на пункт «Розгорнутий веб-сервіс» і знайти посилання на розгорнутий API біля заголовка веб-сервісу. (https://<ID>.onrender.com)
10. Серверна частина доступна через API в Інтернеті після успішного розгортання

Лабораторне завдання

Завдання 1: Створення серверної частини за допомогою Node.js

- **Опис:**
  Розробити серверну частину додатку, створивши сервер використовуючи **Node.js** з **Express**. Цей сервер має обробляти запити API від клієнтської частини (фронтенду) та взаємодіятиме з базою даних для зберігання та отримання даних.
- **Кроки:**
  1. Ініціалізувати новий проект **Node.js** та встановити необхідні залежності (наприклад, Express, CORS тощо).
  2. Налаштувати **Express-сервер** для обробки вхідних запитів з клієнтської частини.
  3. Визначити основні маршрути (наприклад, GET для отримання даних і POST для відправки даних).

4. Налаштувати маршрути автентифікації:
    - POST /register: Створення нового користувача.
    - POST /login: Автентифікація користувача та повернення токену JWT.
    - GET /profile: Отримання даних користувача за допомогою токена.
5. Захистити приватні маршрути за допомогою проміжного програмного рішення для доступу тільки автентифікованих користувачів.
6. Протестувати сервер, налаштувавши прості маршрути, які повертають статичні дані або підтверджують успішні запити.

Завдання 2: Робота з базою даних (зберігання та пошук даних)

- **Опис:**
  Інтегрувати **хмарну базу даних** (наприклад, **Firebase Firestore**) зі розробленою у завданні 1 серверною частиною для зберігання та отримання даних. Дані будуть використовуватися клієнтською частиною для динамічного відображення контенту.
- **Кроки:**
    1. Налаштувати **хмарну базу даних** (**Firebase Firestore**) і підключити її до розробленого сервера Node.js за допомогою відповідного SDK або клієнтської бібліотеки.
    2. Створити моделі даних (наприклад, дописи в блозі, інформацію про продукти, деталі подій) і визначити схему їх зберігання в базі даних.
    3. Реалізувати маршрути **API** для зберігання даних у базі даних (POST) та їх отримання (GET).
    4. Реалізувати завантаження клієнтською частиною збережених даних і динамічно відображати їх на веб-сторінці (наприклад, показувати список продуктів, дописів або деталі подій).

Завдання 3: Розгортання веб-додатку на хмарному хостингу

- **Опис:**
  Розгорнути веб-додаток на обраній хмарній хостинговій платформі.
- **Кроки:**
    1. Обрати хостинг-провайдера
    2. Налаштувати хмарне середовище:
        - Розгорнути **клієнтську частину додатка** на статичному хостингу.
        - Розгорнути **серверну частину додатка** за допомогою хмарної функції або сервера.

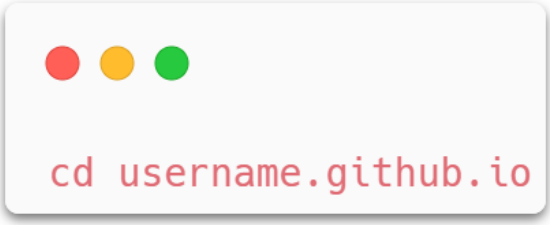
3. Налаштувати комунікацію між **клієнтською та серверною частиною додатка**:
    - Переконатись, що маршрути API працюють з розгорнутою серверною частиною.
    - Налаштувати CORS, якщо це необхідно.

Контрольні запитання

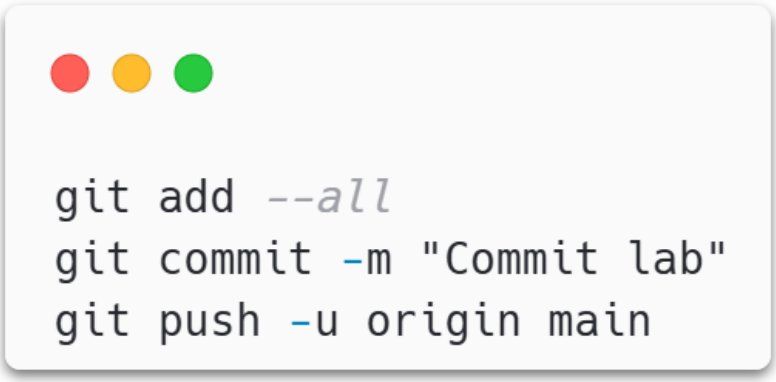
1. Що таке RESTful API, і які основні HTTP-запити використовуються для взаємодії між клієнтською та серверною частинами?
2. Яку роль виконує Express у створенні серверної частини веб-додатку, і як він спрощує обробку HTTP-запитів?
3. Як налаштувати підключення до бази даних Firebase Firestore у Node.js і які переваги цього підходу?

Вимоги до оформлення звіту

3. Увійти в папку проекту



4. Створити нову папку з назвою **lab5** і помістити всі файли створені під час виконання лабораторної роботи в дану папку.
5. Завантажити зміни у репозиторій



Варіанти завдань

Варіант 1

Веб-сайт ресторану

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для зберігання відгуків про ресторан. Налаштувати з'єднання сервера з базою даних відгуків про ресторан.
3. На сервері створити маршрут (HTTP GET) для отримання всіх відгуків про ресторан з бази даних. Використати даний маршрут у клієнтській частині для отримання всіх відгуків про ресторан та подальшого їх відображення на сторінці Про нас.
4. На сервері створити маршрут (HTTP POST) для додавання нового відгуку. Реалізувати перевірку використання назв конкурентів у відгуку. Реалізувати збереження відгуку у базі даних при відсутності використання назв конкурентів у відгуку. Використати створений новий маршрут у клієнтській частині веб-додатку для збереження відгуку при заповненні форми відгуку користувачем на сторінці Про нас.
5. Розгорнути веб-сайт ресторану на хостингу

Варіант 2

Веб-сайт організації онлайн-ігор

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.

2. Створити хмарну базу даних для збереження оцінок ігор залишених користувачами.

3. На сервері створити маршрут (HTTP GET) для отримання середнього рейтингу конкретної гри. Використати його у клієнтській частині для відображення рейтингу біля кожної гри.

4. Реалізувати на сервері маршрут (HTTP POST) для оновлення рейтингу гри після оцінки користувачем. Використати його у клієнтській частині, щоб рейтинг оновлювався після оцінювання гри користувачем.

5. Розгорнути веб-сайт організації онлайн-ігор на хостингу.

Варіант 3

Веб-сайт платформи для навчальних курсів

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.

2. Створити хмарну базу даних для збереження відгуків про курси.

3. На сервері створити маршрут (HTTP GET) для отримання всіх відгуків про конкретний курс. Використати його у клієнтській частині для відображення відгуків під кожним курсом. Перед передачею даних на клієнтську частину реалізувати на сервері трансформацію: додати до

кожного відгуку поле dateFormatted, яке містить відформатовану дату у вигляді "день.місяць.рік" (наприклад, "02.03.2025"), та відсортувати відгуки за датою у порядку спадання.

4. Реалізувати на сервері маршрут (HTTP POST) для додавання нового відгуку про курс. Використати його у клієнтській частині, щоб після написання відгуку він одразу з'являвся на сторінці. Перед збереженням у базі даних додати на сервері перевірку довжини тексту відгуку (мінімум 10 символів, максимум 500 символів) та автоматично додавати до кожного нового відгуку поле часу створення відгуку createdAt із поточною датою у форматі ISO 8601.

5. Розгорнути веб-сайт платформи для навчальних курсів на хостингу.

Варіант 4

Веб-сайт платформи для бронювання турів

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.

2. Створити хмарну базу даних для збереження відгуків про тури.

3. На сервері створити маршрут (HTTP GET) для отримання всіх відгуків про конкретний тур. Використати його у клієнтській частині для відображення відгуків під кожним туром. Перед передачею даних на клієнтську частину реалізувати на сервері трансформацію: для кожного туру додати поле averageRating, що містить середнє значення рейтингу туру на основі оцінок користувачів, та сортувати відгуки за рейтингом від кращих до гірших оцінок.

4. Реалізувати на сервері маршрут (HTTP POST) для додавання нового відгуку про тур. Використати його у клієнтській частині, щоб після написання відгуку він одразу з'являвся на сторінці. Перед збереженням у базі даних додати на сервері перевірку наявності в тексті відгуку заборонених слів (чорний список слів).

5. Розгорнути веб-сайт платформи для бронювання турів на хостингу.

Варіант 5

Веб-сайт онлайн-магазину книг

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для зберігання оформлених замовлень.
3. На сервері створити маршрут (HTTP GET) для отримання всіх оформлених замовлень користувача. Використати створений маршрут на клієнтській частині для відображення оформлених замовлень на сторінці "Мій акаунт". Додатково реалізувати сортування отриманих замовлень за датою оформлення (від найновіших до найстаріших) перед передачею їх на клієнтську частину.

4. Реалізувати на сервері маршрут (HTTP POST) для збереження оформленого нового замовлення. Перед збереженням у базі даних перевіряти наявність товарів у кошику (запобігти збереження пустого списку товарів).
5. Розгорнути веб-сайт онлайн-магазину книг на хостингу.

Варіант 6

Веб-сайт платформи для оренди автомобілів

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для зберігання інформації про бронювання.
3. На сервері створити маршрут (HTTP GET) для отримання всіх бронювань користувача. Використати його у клієнтській частині для відображення бронювань на сторінці "Мої бронювання".
4. Реалізувати на сервері маршрут (HTTP POST) для зберігання нового оформленого бронювання. Використати створений маршрут у клієнтській частині для підтвердження бронювання після заповнення форми користувачем. Перед збереженням бронювання в базі перевіряти доступність автомобіля на вибрані дати та автоматично зменшувати кількість доступних одиниць. Якщо авто недоступне, надсилати відповідне повідомлення користувачу.
5. Розгорнути веб-сайт платформи для оренди автомобілів на хостингу.

Варіант 7

Веб-сайт платформи для оренди житла

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для зберігання відгуків про квартири.
3. На сервері створити маршрут (HTTP GET) для отримання всіх відгуків про конкретну квартиру. Використати створений маршрут у клієнтській частині для відображення відгуків під кожною квартирою. Реалізувати пагінацію на клієнтській та серверній частині, якщо кількість відгуків більша десяти, щоб уникнути високого навантаження на клієнтську частину.
4. Реалізувати на сервері маршрут (HTTP POST) для додавання нового відгуку про квартиру. Використати створений маршрут у клієнтській частині, щоб відгук з’являвся на сторінці після його додавання користувачем.
5. Розгорнути веб-сайт платформи для оренди житла на хостингу.

Варіант 8

Веб-сайт платформи для онлайн-бронювання квитків на заходи

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження оцінок подій.
3. На сервері створити маршрут (HTTP GET) для отримання всіх оцінок події. Використати створений маршрут для відображення середньої оцінки під кожною подією на сторінці. Реалізувати пагінацію на клієнтській та серверній частині, якщо кількість відгуків більша десяти, щоб уникнути високого навантаження на клієнтську частину.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження нової оцінки події. Використати створений маршрут на клієнтській частині, щоб після оцінки користувачем середня оцінка одразу оновлювалась на сторінці події.
5. Розгорнути веб-сайт платформи для онлайн-бронювання квитків на заходи на хостингу.

Варіант 9

Веб-сайт платформи для організації волонтерських ініціатив

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження оцінок ініціатив.
3. На сервері реалізувати маршрут (HTTP GET) для отримання всіх оцінок конкретної ініціативи та трансформувати отримані дані перед передачею клієнтській частині для відображення середнього значення оцінок з округленням до двох знаків після коми.
4. Реалізувати на сервері маршрут (HTTP POST) для додавання нової оцінки ініціативи. Використати ствоерний маршрут на клієнтській частині, щоб після додавання нової оцінки, загальна середня оцінка оновлювалась на сторінці ініціативи.
5. Розгорнути веб-сайт платформи для організації волонтерських ініціатив на хостингу.

Варіант 10

Веб-сайт онлайн-магазину спортивних товарів

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження персональних списків бажаних товарів.
3. На сервері створити маршрут (HTTP GET) для персональних списків бажаних товарів. Використати створений маршрут у клієнтській частині для відображення персональних списків бажаних товарів у профілі користувача.

4. На сервері створити маршрут (HTTP POST) для зберігання нових персональних списків бажаних товарів у базі даних, при цьому здійснити валідацію даних перед їх збереженням і уникнути дублювання товарів у списках користувачів.
5. Розгорнути веб-сайт онлайн-магазину спортивних товарів на хостингу.

Варіант 11

Веб-сайт платформи для пошуку робочих вакансій

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження історії поданих заявок.
3. На сервері створити маршрут (HTTP GET) для отримання всіх поданих користувачем заявок. Використати створений маршрут у клієнтській частині для відображення списку поданих заявок у профілі користувача.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження подачі нової заявки на вакансію у базі даних, при цьому валідувати дані перед збереженням, щоб запобігти дублюванню заявок на одну й ту саму вакансію користувачем.
5. Розгорнути веб-сайт платформи для пошуку робочих вакансій на хостингу.

Варіант 12

Веб-сайт платформи для замовлення їжі

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження історії замовлень користувача.
3. На сервері створити маршрут (HTTP GET) для отримання всіх створених користувачем замовлень. Використати створений маршрут у клієнтській частині для відображення списку створених замовлень на сторінці Мої замовлення.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження оформленого замовлення у базі даних, при цьому здійснити перевірку кількості страв у кошику перед підтвердженням замовлення (мінімум - одна страва, максимум – 10 страв). Використати створений маршрут у клієнтській частині, щоб після оформлення замовлення воно відображалось у списку замовлень.
5. Розгорнути веб-сайт платформи для замовлення їжі на хостингу.

Варіант 13

Веб-сайт блогу для подорожей

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження вподобаних статей.
3. На сервері створити маршрут (HTTP GET) для отримання всіх вподобаних користувачем статей. Використати створений маршрут у клієнтській частині для відображення кнопки "Подобається" з оновленим стелем для вподобаних статей.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження вподобаних статей користувачем у базі даних, а також додати перевірку на те, чи вже було поставлено вподобання для цієї статті, щоб уникнути дублювання. Використати створений маршрут у клієнтській частині, для збереження інформації про вродобану статтю після натиснення користувачем кнопки "Подобається".
5. Розгорнути веб-сайт блогу для подорожей на хостингу.

Варіант 14

Веб-сайт інтерактивної платформи для вивчення мов

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження пройдених уроків.
3. На сервері створити маршрут (HTTP GET) для отримання всіх пройдених користувачем уроків з фільтрацією по даті проходження. Використати створений маршрут у клієнтській частині для відображення пройдених уроків із зміненим стилем.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження пройдених користувачем уроків у базі даних з датою проходження. Використати створений маршрут у клієнтській частині, для збереження інформації про пройдений урок після натиснення користувачем кнопки "Відзначити як пройдений".
5. Розгорнути веб-сайт інтерактивної платформи для вивчення мов на хостингу.

Варіант 15

Веб-сайт платформи для вивчення історії через інтерактивні події

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження результатів проходження тестів.

3. На сервері створити маршрут (HTTP GET) для отримання середньої оцінки всіх пройдених користувачем тестів. Використати створений маршрут у клієнтській частині для відображення середньої оцінки всіх пройдених користувачем тестів зверху сторінки.
4. Реалізувати на сервері маршрут (HTTP POST) для збереження оцінок пройдених користувачем тестів та дати проходження у базі даних. Використати створений маршрут у клієнтській частині, для збереження інформації про отриману оцінку після проходження користувачем тесту.
5. Розгорнути веб-сайт платформи для вивчення історії через інтерактивні події на хостингу.

Варіант 16

Веб-сайт симулятора управління власним стартапом

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про компанію.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про створену компанію. Використати створений маршрут у клієнтській частині для відображення інформації про компанію на сторінці "Мій стартап".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про створену компанію. Використати створений маршрут у клієнтській частині, для збереження змін інформації про отриману інформації про компанію на сторінці "Мій стартап". Реалізувати валідацію у мінімум 5 знаків імені створеної компанії перед збереженням інформації у базі даних.
5. Розгорнути веб-сайт симулятора управління власним стартапом на хостингу.

Варіант 17

Веб-сайт платформи для створення та управління персональними цілями

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про виконані користувачем цілі.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про виконані користувачем цілі відфільтровані за датою виконання. Використати створений маршрут у клієнтській частині для відображення інформації про виконані користувачем цілі на сторінці "Прогрес".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про виконані користувачем цілі додавши поле часу виконання. Використати створений маршрут у клієнтській частині, для збереження змін інформації про виконані користувачем цілі на сторінці "Прогрес".

5. Розгорнути веб-сайт платформи для створення та управління персональними цілями на хостингу.

Варіант 18

Веб-сайт платформи для віртуальних хакатонів та змагань із програмування

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про подані користувачем заявки.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про подані користувачем заявки відфільтровані за датою створення заявки. Використати створений маршрут у клієнтській частині для відображення інформації про подані користувачем заявки на сторінці "Мої проєкти".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про подані користувачем заявки. Використати створений маршрут у клієнтській частині, для збереження змін інформації про подані користувачем заявки на сторінці "Мої проєкти".
5. Розгорнути веб-сайт платформи для віртуальних хакатонів та змагань із програмування на хостингу.

Варіант 19

Веб-сайт симулятора управління будівництвом міста

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про покращені будівлі та їх характеристики.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про покращені будівлі та їх характеристики. Використати створений маршрут у клієнтській частині для відображення інформації про покращені будівлі та їх характеристики на сторінці "Моє місто".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про покращені будівлі та їх характеристики з обмеженням на частоту збереження інформації один раз на хвилину. Використати створений маршрут у клієнтській частині, для збереження змін інформації про покращені будівлі та їх характеристики на сторінці "Моє місто" після натиснення кнопки "Покращити будівлю".
5. Розгорнути веб-сайт симулятора управління будівництвом міста на хостингу.

Варіант 20

Веб-сайт симулятора космічної експедиції

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про активні місії.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про активні місії відсортовані по часу початку місії. Використати створений маршрут у клієнтській частині для відображення інформації про активні місії на сторінці "Мої подорожі ".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про активні місії з полем часу початку місії. Використати створений маршрут у клієнтській частині, для збереження змін інформації про активні місії на сторінці "Мої подорожі ".
5. Розгорнути веб-сайт симулятора космічної експедиції на хостингу.

Варіант 21

Веб-сайт платформи для моніторингу здоров'я та фізичної активності

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про активні тренування.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про активні тренування, що мають бути згруповані за типами тренувань. Використати створений маршрут у клієнтській частині для відображення інформації про активні тренування на сторінці "Мій прогрес".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про активні тренування. Використати створений маршрут у клієнтській частині, для збереження змін інформації про активні тренування.
5. Розгорнути веб-сайт платформи для моніторингу здоров'я та фізичної активності на хостингу.

Варіант 22

Веб-сайт платформи для онлайн-курсу з фотографії

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про пройдені користувачем уроки.

3. На сервері створити маршрут (HTTP GET) для отримання інформації про пройдені користувачем уроки відсортовані за датою проходження. Використати створений маршрут у клієнтській частині для відображення інформації про пройдені користувачем уроки на сторінці "Мій прогрес".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про пройдені користувачем уроки з датою проходження. Використати створений маршрут у клієнтській частині, для збереження змін інформації про пройдені користувачем уроки.
5. Розгорнути веб-сайт платформи для онлайн-курсу з фотографії на хостингу.

Варіант 23

Веб-сайт платформи для оренди спортивного обладнання

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про оформлені користувачем оренди обладнання.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про оформлені користувачем оренди обладнання відфільтровані по ціні. Використати створений маршрут у клієнтській частині для відображення інформації про оформлені користувачем оренди обладнання на сторінці "Мої оренди".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про оформлені користувачем оренди обладнання та ціну оренди. Використати створений маршрут у клієнтській частині, для збереження змін інформації про оформлені користувачем оренди обладнання.
5. Розгорнути веб-сайт платформи для оренди спортивного обладнання на хостингу.

Варіант 24

Веб-сайт платформи для планування подорожей

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про список запланованих користувачем подорожей.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про список запланованих користувачем подорожей відсортованих за тривалістю подорожі. Використати створений маршрут у клієнтській частині для відображення інформації про список запланованих користувачем подорожей на сторінці "Мої подорожі".

4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про список запланованих користувачем подорожей. Використати створений маршрут у клієнтській частині, для збереження змін інформації про список запланованих користувачем подорожей.
5. Розгорнути веб-сайт платформи для планування подорожей на хостингу.

Варіант 25

Веб-сайт платформи для створення та обміну рецептами

1. Налаштувати серверну частину з використанням Node.js/Express для хостингу статичних файлів сайту.
2. Створити хмарну базу даних для збереження інформації про список створених користувачем рецептів.
3. На сервері створити маршрут (HTTP GET) для отримання інформації про список створених користувачем рецептів відсортованих за часом приготування. Використати створений маршрут у клієнтській частині для відображення інформації про список створених користувачем рецептів на сторінці "Мої рецепти".
4. Реалізувати на сервері маршрут (HTTP POST) для збереження інформації про список створених користувачем рецептів. Використати створений маршрут у клієнтській частині, для збереження змін інформації про список створених користувачем рецептів.
5. Розгорнути веб-сайт платформи для створення та обміну рецептами на хостингу.