Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

JSON Tool

Виконав
студент групи ІА–22:
Щаблевський Д. Е.
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Зміст 1
Хід роботи: 3
Реалізація клієнт-серверної взаємодії 3
Взаємодія між клієнтом і сервером: 6
Висновок: 7

Тема: Різні види взаємодії: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Мета: Ознайомитися з основними принципами та шаблонами взаємодії між додатками, такими як «Client-Server», «Peer-to-Peer» та «Service-Oriented Architecture», дослідити їхні особливості та способи реалізації для створення гнучких, масштабованих і ефективних програмних систем.

Хід роботи:

Реалізація клієнт-серверної взаємодії

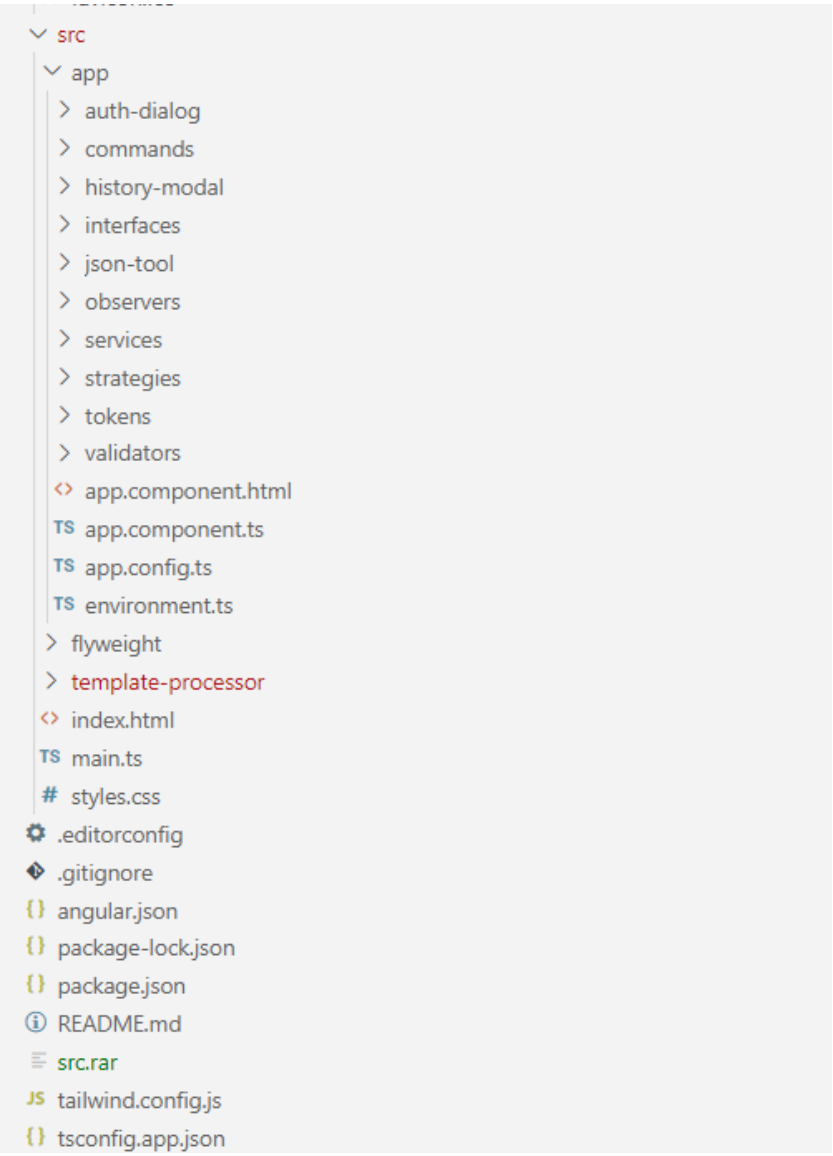

Рис. 1 — Структура проекту клієнта

```
@Injectable({ providedIn: 'root' })
export class FirestoreService {
  constructor(private firestore: Firestore) {}

  async saveDocument(type: DocumentType, content: string, userId: string) {
    const collectionRef = collection(this.firestore, type);
    const doc = {
      content,
      timestamp: new Date(),
      userId,
      version: 1
    };
    return addDoc(collectionRef, doc);
  }

  async getLastVersions(type: DocumentType, userId: string, limitQuantity = 10) {
    const q = query(
      collection(this.firestore, type),
      where('userId', '==', userId),
      orderBy('timestamp', 'desc'),
      limit(limitQuantity)
    );
    return getDocs(q);
  }

  async getJsonById(type: DocumentType, docId: string) {
    const docRef = doc(this.firestore, type, docId);
    return getDoc(docRef);
  }

  async updateJson(type: DocumentType, docId: string, content: string, userId: string) {
    const docRef = doc(this.firestore, type, docId);
    return updateDoc(docRef, {
      content,
      timestamp: new Date(),
      userId,
      version: increment(1)
    });
  }

  async deleteJson(type: DocumentType, docId: string) {
    const docRef = doc(this.firestore, type, docId);
    return deleteDoc(docRef);
  }

  async getAllUserDocuments(type: DocumentType, userId: string) {
    const q = query(
      collection(this.firestore, type),
      where('userId', '==', userId),
      orderBy('timestamp', 'desc')
    );
    return getDocs(q);
  }
}
```

Рис. 2 — Реалізація API на стороні клієнта

У цьому прикладі взаємодія розподілених частин (клієнта та сервера) реалізована з використанням Firebase, де хмарний сервіс надає кінцеві точки (endpoints) для роботи з ресурсами, а клієнт звертається до них через HTTP-запити. Детально розглянемо, як ця взаємодія працює.

Серверна частина: Firebase Authentication та Cloud Datastore

У цьому проекті серверна частина реалізована через **Firebase Authentication** для аутентифікації користувачів і **Cloud Datastore** для зберігання JSON-даних. Firebase забезпечує хмарні сервіси для управління користувачами та ефективного зберігання даних.

1. **Firebase Authentication (Аутентифікація):**
    - Використовується для реєстрації користувачів, входу до системи та управління доступом.
    - Забезпечує надійний механізм автентифікації за допомогою email/password, Google, Facebook або інших методів.
    - Кожен користувач має власний унікальний ідентифікатор (UID), що використовується для авторизації доступу до даних.
2. **Cloud Datastore (Зберігання даних):**
    - Використовується для зберігання і обробки JSON-даних у вигляді документів.
    - Cloud Datastore надає можливість зберігати структуровані дані у форматі JSON, здійснювати запити, оновлення та видалення документів.
    - Дані організовані в колекціях, де кожен документ представляє окремий JSON-об'єкт.

Основні функції серверної частини:

- **Зберігання і обробка JSON-даних:**
    - JSON-об'єкти зберігаються в Cloud Datastore, де кожен об'єкт є окремим документом.
- **Авторизація користувачів:**
    - Firebase Authentication забезпечує доступ до даних лише авторизованим користувачам.
- **Доступ до даних:**
    - Взаємодія з даними здійснюється через REST API, використовуючи Firebase SDK для доступу до Cloud Datastore.

Клієнтська частина: Angular

Клієнтська частина реалізована за допомогою Angular, яка взаємодіє з Firebase для аутентифікації користувачів та з Cloud Datastore для роботи з JSON-даними.

1. **Сервіс (FirebaseService):**
    - Сервіс **FirebaseService**відповідає за взаємодію з Cloud Datastore через Firebase SDK. Він реалізує логіку для отримання, збереження та видалення JSON-даних.
    - Основні функції сервісу:
        - **Отримання списку JSON-даних:**
            - Метод getJsonList() запитує всі документи з Cloud Datastore і отримує їх як список JSON-об'єктів.
        - **Отримання одного JSON-об'єкта:**
            - Метод getJsonById(id: string) отримує конкретний документ за ID з Cloud Datastore.
        - **Створення нового JSON-об'єкта:**
            - Метод createJson(newJson: any) додає новий документ до Cloud Datastore.
        - **Видалення JSON-об'єкта:**
            - Метод deleteJson(id: string) видаляє документ з Cloud Datastore за його ID.
2. **Компоненти Angular:**
    - **JsonEditorComponent:** Компонент для створення та редагування JSON-об'єктів, що дозволяє користувачам взаємодіяти з полями JSON і зберігати зміни.
    - **JsonListComponent:** Компонент для перегляду списку JSON-об'єктів, що отримує та відображає їх за допомогою сервісу.
3. **Аутентифікація користувачів:**
    - **AuthService:** Сервіс для роботи з Firebase Authentication, який відповідає за реєстрацію, вхід користувачів та керування їх сесіями.
    - Користувачі повинні бути автентифіковані для доступу до JSON-даних.

Взаємодія між клієнтом і сервером:

- **Аутентифікація:** Клієнт здійснює запити до Firebase Authentication для входу користувача і отримання токена доступу.
- **Доступ до даних:** Клієнтська частина Angular виконує запити до Cloud Datastore для отримання і редагування JSON-даних. Кожен запит до Cloud Datastore супроводжується токеном доступу для перевірки прав користувача.

Взаємодія розподілених частин (Firebase та Angular):

1. **API для взаємодії:**

- Клієнт (Angular) використовує AngularFire для взаємодії з Firebase, відправляючи запити до Firestore та отримуючи відповіді у форматі JSON.

2. **Асинхронність:**
   - Використовуються асинхронні методи для запитів до Firebase, що дозволяє забезпечити швидку та безперервну роботу користувача з JSON-даними без блокування інтерфейсу.
3. **Протокол HTTP:**
   - Firebase функції обробляють HTTP-запити від клієнта, використовуючи стандартні методи HTTP, що дозволяє здійснювати CRUD-операції з JSON.
4. **Розподіленість:**
   - Завдяки використанню Firebase як хмарної платформи, клієнт та сервер можуть працювати на різних пристроях або навіть географічно віддалених серверах, забезпечуючи масштабованість і незалежність системи.

Код проекту:
https://github.com/neodavis/json-tool

Висновок:
У ході лабораторної роботи було реалізовано клієнт-серверну архітектуру для взаємодії між клієнтом та сервером у застосунку. Використання **Firebase Authentication** для аутентифікації та **Cloud Datastore** для зберігання даних дозволило забезпечити безпечну авторизацію користувачів і ефективне управління даними. Це дозволило створити гнучку, масштабовану систему, яка ефективно обробляє запити від клієнта через REST API. Завдяки реалізації асинхронної роботи з даними на стороні клієнта на базі **Angular**, було досягнуто високої продуктивності та безперервної взаємодії з користувачем. Такий підхід дозволяє спростити додавання нового функціоналу та забезпечити стабільну роботу додатка навіть за значного навантаження.