はい、承知いたしました。Angularのヒーローチュートリアルを、バックエンドとしてローカルのExpressとSQLite3に接続する手順を説明します。

全体の流れ

1. **Express + SQLite3** のバックエンド**API**サーバーを構築する
    - ヒーローのデータを管理するためのAPI（GET, POST, PUT, DELETE）を作成します。
2. **Angular**ヒーローチュートリアルの**HeroService**を修正する
    - 既存のインメモリデータベース（in-memory-web-api）へのリクエストを、作成したローカルのExpressサーバーに向けるように変更します。

ステップ1: ExpressとSQLite3でAPIサーバーを構築する

まず、ヒーローのデータを管理するためのAPIサーバーを作成します。

1. プロジェクトのセットアップ

1. Angularのプロジェクトとは別のフォルダで作業します。
   ```
   mkdir hero-backend
   cd hero-backend
   npm init -y
   npm install express sqlite3 cors
   ```
    - **express**: Webサーバーフレームワーク
    - **sqlite3**: SQLiteデータベースを操作するライブラリ
    - **cors**: Angularアプリ（localhost:4200）からのリクエストを許可するために必要
2. server.jsというファイルを作成します。

2. データベースの初期設定

server.jsに、データベースの接続とテーブル作成のコードを追加します。

```
// server.js
const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const cors = require('cors');

const app = express();
const port = 3000; // APIサーバーのポート

app.use(cors()); // CORSを許可
app.use(express.json()); // POSTリクエストのbodyをパースするために必要

// データベースに接続（ファイルがなければ自動作成）
const db = new sqlite3.Database('./heroes.db', (err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Connected to the SQLite database.');
```

```
});

// ヒーローテーブルを作成
db.serialize(() => {
  db.run(`CREATE TABLE IF NOT EXISTS heroes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
  )`, (err) => {
    if (err) {
      return console.error(err.message);
    }
    // 初期データを投入（初回のみ）
    const stmt = db.prepare("INSERT INTO heroes (name) VALUES (?)");
    const initialHeroes = ['Dr. Nice', 'Bombasto', 'Celeritas',
'Magneta', 'RubberMan', 'Dynama', 'Dr. IQ', 'Magma', 'Tornado'];
    initialHeroes.forEach(name => {
        db.get("SELECT * FROM heroes WHERE name = ?", [name], (err,
row) => {
            if (!row) {
                stmt.run(name);
            }
        });
    });
    stmt.finalize();
  });
});
```

3. APIエンドポイントの作成

server.jsに、ヒーローを操作するためのCRUD（作成、読み取り、更新、削除）APIを追加します。

```
// server.js の続き...

// すべてのヒーローを取得 (GET /api/heroes)
app.get('/api/heroes', (req, res) => {
  const name = req.query.name;
  if (name) {
    // ヒーローを名前で検索
    db.all("SELECT * FROM heroes WHERE name LIKE ?", [`%${name}%`],
(err, rows) => {
      if (err) {
        res.status(500).json({ "error": err.message });
        return;
      }
      res.json(rows);
    });
  } else {
    // 全件取得
```

```
    db.all("SELECT * FROM heroes", [], (err, rows) => {
      if (err) {
        res.status(500).json({ "error": err.message });
        return;
      }
      res.json(rows);
    });
  }
});

// IDでヒーローを取得 (GET /api/heroes/:id)
app.get('/api/heroes/:id', (req, res) => {
  const id = req.params.id;
  db.get("SELECT * FROM heroes WHERE id = ?", [id], (err, row) => {
    if (err) {
      res.status(500).json({ "error": err.message });
      return;
    }
    res.json(row);
  });
});

// ヒーローを追加 (POST /api/heroes)
app.post('/api/heroes', (req, res) => {
  const { name } = req.body;
  db.run(`INSERT INTO heroes (name) VALUES (?)`, [name], function(err)
{
    if (err) {
      res.status(500).json({ "error": err.message });
      return;
    }
    res.json({ id: this.lastID, name: name });
  });
});

// ヒーローを更新 (PUT /api/heroes)
app.put('/api/heroes', (req, res) => {
  const { id, name } = req.body;
  db.run(`UPDATE heroes SET name = ? WHERE id = ?`, [name, id],
function(err) {
    if (err) {
      res.status(500).json({ "error": err.message });
      return;
    }
    res.status(204).send(); // No Content
  });
});
```

```
// ヒーローを削除 (DELETE /api/heroes/:id)
app.delete('/api/heroes/:id', (req, res) => {
  const id = req.params.id;
  db.run(`DELETE FROM heroes WHERE id = ?`, id, function(err) {
    if (err) {
      res.status(500).json({ "error": err.message });
      return;
    }
    res.status(204).send(); // No Content
  });
});


// サーバーを起動
app.listen(port, () => {
  console.log(`API server listening on http://localhost:${port}`);
});
```

4. APIサーバーの起動

ターミナルで以下のコマンドを実行して、APIサーバーを起動します。
```
node server.js
```

これで、http://localhost:3000/api/heroes にアクセスするとヒーローのリストがJSON形式で返ってくるようになります。

ステップ2: Angularヒーローチュートリアルの修正

次に、Angularアプリケーションがこの APIサーバーと通信するように設定を変更します。

1. in-memory-web-apiの削除

ヒーローチュートリアルでは、angular-in-memory-web-apiというライブラリが擬似的なバックエンドとして機能しています。これを削除し、実際のAPIを叩くように変更します。

1. **app.module.ts**の修正 HttpClientInMemoryWebApiModuleのインポートとimports配列からの登録を削除します。
   ```
   // src/app/app.module.ts

   // この行を削除
   // import { HttpClientInMemoryWebApiModule } from
   'angular-in-memory-web-api';
   // import { InMemoryDataService } from './in-memory-data.service';

   @NgModule({
     imports: [
       // ...
       HttpClientModule,
   ```

```
      // 以下の2行を削除またはコメントアウト
      // HttpClientInMemoryWebApiModule.forRoot(
      //   InMemoryDataService, { dataEncapsulation: false }
      // )
    ],
    // ...
  })
  export class AppModule { }
```

2. **in-memory-data.service.tsの削除** src/app/in-memory-data.service.tsファイルは不要になるので、削除してしまって構いません。

2. hero.service.tsの修正

HeroService内のAPIのURLを、先ほど作成したExpressサーバーのURLに変更します。

```
// src/app/hero.service.ts
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
// ...

@Injectable({ providedIn: 'root' })
export class HeroService {

  // Expressサーバーの URLに変更
  private heroesUrl = 'http://localhost:3000/api/heroes';

  httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
  };

  constructor(private http: HttpClient) { }

  // getHeroes()メソッド
  getHeroes(): Observable<Hero[]> {
    return this.http.get<Hero[]>(this.heroesUrl)
      .pipe(
        tap(_ => this.log('fetched heroes')),
        catchError(this.handleError<Hero[]>('getHeroes', []))
      );
  }

  // getHero()メソッド
  getHero(id: number): Observable<Hero> {
    const url = `${this.heroesUrl}/${id}`; // URLを修正
    return this.http.get<Hero>(url).pipe(
      tap(_ => this.log(`fetched hero id=${id}`)),
      catchError(this.handleError<Hero>(`getHero id=${id}`))
```

```
    );
  }

  // updateHero()メソッド
  updateHero(hero: Hero): Observable<any> {
    // PUTリクエストのURLは /api/heroes のまま
    return this.http.put(this.heroesUrl, hero, this.httpOptions).pipe(
      tap(_ => this.log(`updated hero id=${hero.id}`)),
      catchError(this.handleError<any>('updateHero'))
    );
  }

  // addHero()メソッド
  addHero(hero: Hero): Observable<Hero> {
    return this.http.post<Hero>(this.heroesUrl, hero,
this.httpOptions).pipe(
      tap((newHero: Hero) => this.log(`added hero w/
id=${newHero.id}`)),
      catchError(this.handleError<Hero>('addHero'))
    );
  }

  // deleteHero()メソッド
  deleteHero(id: number): Observable<Hero> {
    const url = `${this.heroesUrl}/${id}`; // URLを修正
    return this.http.delete<Hero>(url, this.httpOptions).pipe(
      tap(_ => this.log(`deleted hero id=${id}`)),
      catchError(this.handleError<Hero>('deleteHero'))
    );
  }

  // searchHeroes()メソッド
  searchHeroes(term: string): Observable<Hero[]> {
    if (!term.trim()) {
      return of([]);
    }
    // クエリパラメータを付与
    return
this.http.get<Hero[]>(`${this.heroesUrl}/?name=${term}`).pipe(
      tap(x => x.length ?
         this.log(`found heroes matching "${term}"`) :
         this.log(`no heroes matching "${term}"`)),
      catchError(this.handleError<Hero[]>('searchHeroes', []))
    );
  }

  // log() と handleError() は変更なし
  // ...
```

```
}
```

3. Angularアプリの起動

すべての修正が完了したら、Angularアプリケーションを起動します。
```
ng serve
```

ブラウザで http://localhost:4200 を開くと、ヒーローの一覧が表示されます。ヒーローの追加、編集、削除を行うと、Expressサーバーのコンソールにログが表示され、heroes.dbファイルにデータが永続化されるのが確認できます。