

SQLite en Wasm

/ Pour quoi faire, et comment ?

François Mockers

PayLead

/About me

QA Lead @  PAYLEAD

*Fidélité et achats dans l'appli de votre banque,
pour une expérience client fluide*

Maintainer @  BEVY

*Un moteur de jeu simple et orienté données
en Rust !*

@FrancoisMockers@hachyderm.io

/Intro

- 1. SQLite*
- 2. Son impact sur le monde mobile*
- 3. Son arrivée en Wasm*
- 4. Les APIs navigateurs*

/SQLite

- Base de données intégrée
- Projet commencé en 2000
- Bibliothèque C, pas de processus serveur
- Support complet de SQL
 - <https://www.sqlite.org/fullsql.html>
- Transactions ACID
- 1 base de données = 1 fichier

- Estimations:
 - 1e12 bases de données
 - 2ème bibliothèque logicielle la plus déployée au monde
- macOS et Windows, iOS et Android
- Firefox, Safari, Chrome
- PHP, Python
- Airbus, GM, Nissan, Suzuki

***/ Dans les applications
mobiles***

- Facile à utiliser par les applications
 - Le système met à disposition des APIs pour la gestion des bases
- Sauvegarde des réglages, des données
 - Synchronisation entre les appareils
 - Sauvegarde et backup
- Meilleure utilisabilité en mode offline
 - L'application n'a pas besoin d'appels réseaux pour manipuler les données
- Meilleur respect de la vie privée
 - Les données utilisateurs n'ont pas besoin d'être traitées par le serveur

/SQLite en Wasm

/ Pourquoi

- Meilleure expérience utilisateur
 - Réactivité
 - Persistance
- Réduire les échanges réseaux
- Réduire la charge serveur
- Respect de la vie privée










- sql.js démarré en 2012, version 1 en novembre 2019
- SQLite Wasm disponible depuis mars 2023 (3.40)
- Compile SQLite en Wasm avec des aides pour remplacer la libc et pour la persistance
- 4 niveaux:
 - API bas niveau, proche de celle en c
 - API orientée objet
 - API communiquant avec un Web Worker par message
 - API avec promises

- Exécution depuis le thread UI
 - Facile à mettre en place, mais pas recommandé
 - Persistence par SessionStorage et LocalStorage
 - API C ou OO
- Exécution depuis un Web Worker
 - Persistence par Origin Private File System
 - Ne bloque pas le thread UI
 - API par message ou Promise

```
<script src="sqlite3.js"></script>

globalThis.sqlite3InitModule().then((sqlite3) => {
  try {
    const db = new sqlite3.oo1.DB();
    try {
      db.exec('CREATE TABLE IF NOT EXISTS t(a,b)');
      db.exec({
        sql: 'INSERT INTO t(a,b) VALUES (?,?)',
        bind: [27, 18],
      });
      db.exec({
        sql: 'SELECT a FROM t ORDER BY a LIMIT 3',
        callback: (row) => {
          console.log(row);
        },
      });
    } finally {
      db.close();
    }
  } catch (err) {
    console.log(err.name, err.message);
  }
});
```

- Scénario
 - Une requête de base pour extraire des données de Snowflake
 - L'utilisateur peut affiner ou trier les résultats

- Scénario
 - Une requête de base pour extraire des données de Snowflake
 - L'utilisateur peut affiner ou trier les résultats
- ~~Re-exécuter la requête mise à jour sur Snowflake~~  
- ~~Mettre en cache les données en backend, envoyer les données retraitées au frontend~~    
- Mettre les données dans une base SQLite en frontend, et faire les derniers traitements dans le front   

/D mo

- <https://sql.js.org/examples/GUI/>
- <https://github.com/lerocha/chinook-database/releases/tag/v1.4.5>

***/ Les APIs navigateurs
utilisées***

/Et les autres

- Shared Array Buffers
- File System API
- WebGPU / WebXR



Questions?

DEVOXX FRANCE 2024