CushionDB

A PWA Compliant User Data Management Tool

What is CushionDB?

CushionDB is an open source easy-to-use framework that simplifies data management when building small offline-first Progressive Web Apps.

You are a front-end web developer and you want to build an application that revolves around a single user's data - such as a personal note taking or todo app.

Sounds pretty simple right?

Next, you want user's to be able to access their data across multiple devices.

Still, not too difficult.

The application needs to be available to the user whether they are online or offline.

How do you do this?

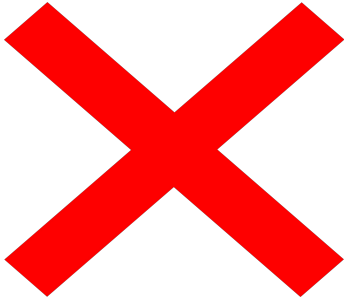
Options

Platform Specific Development

iOS

Android

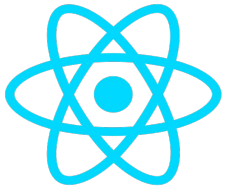
Windows



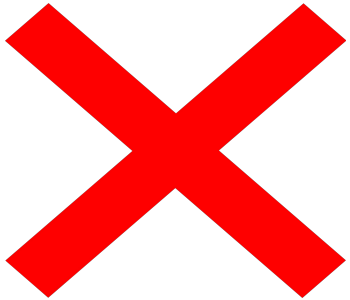
Cross-platform Libraries

Cordova

Ionic



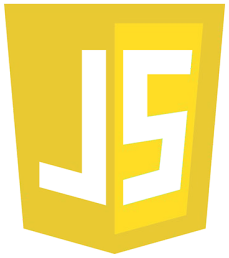
React Native



Front-End Web With PWA Tools

HTML

CSS



JavaScript

Roadmap

Section 1:

- ❑ Introduction to Progressive Web Apps
- ❑ The Progressive Web App Toolkit

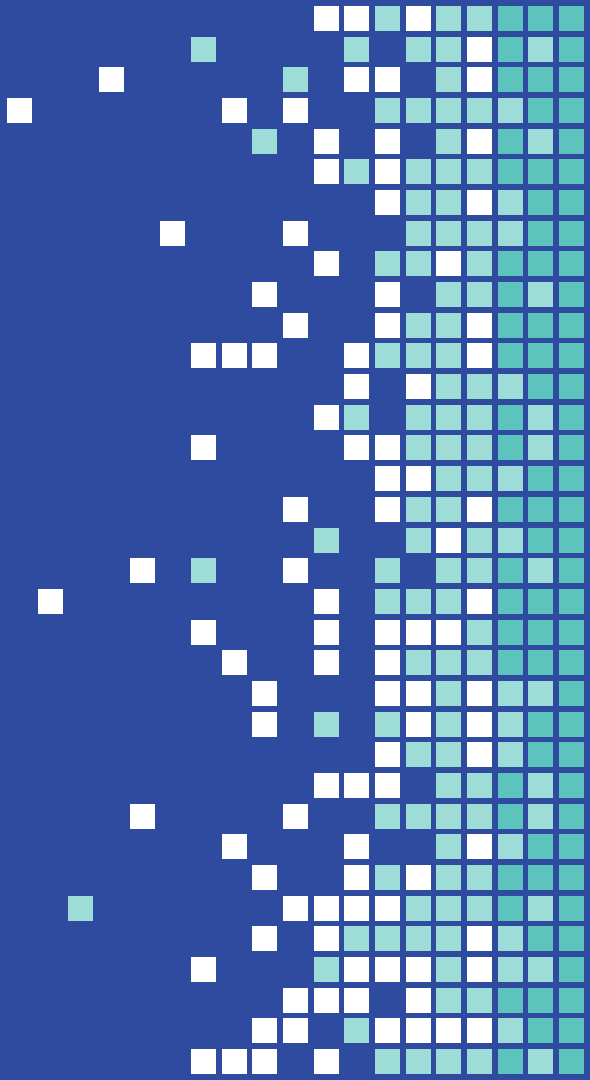
Section 2:

- ❑ Offline FIrst Data Management
- ❑ Frameworks / Solutions

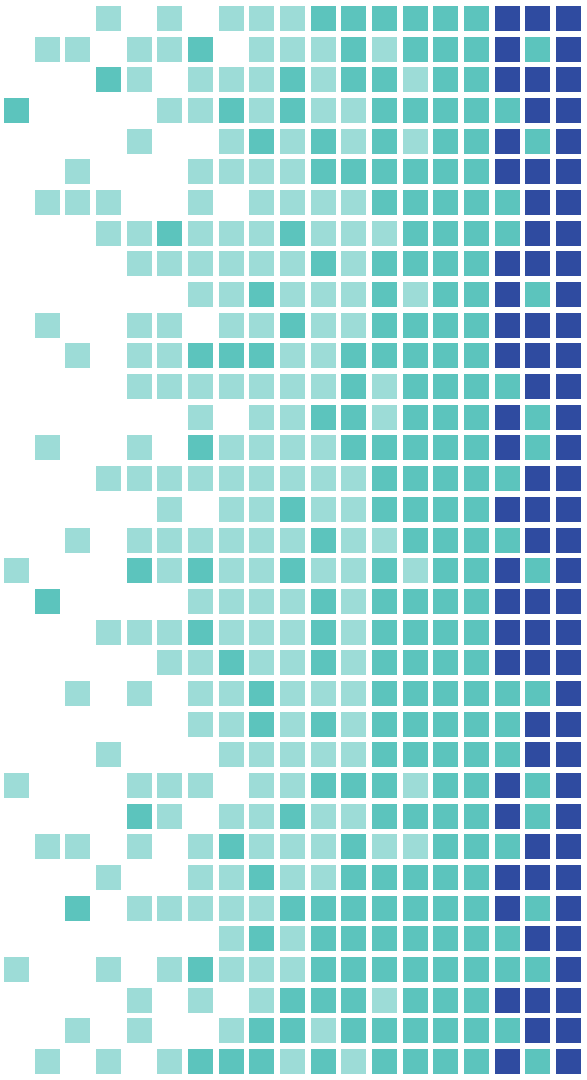
Section 3:

- ❑ CushionDB
- ❑ Challenges and Considerations

Introduction to Progressive Web Apps *(PWAs)*

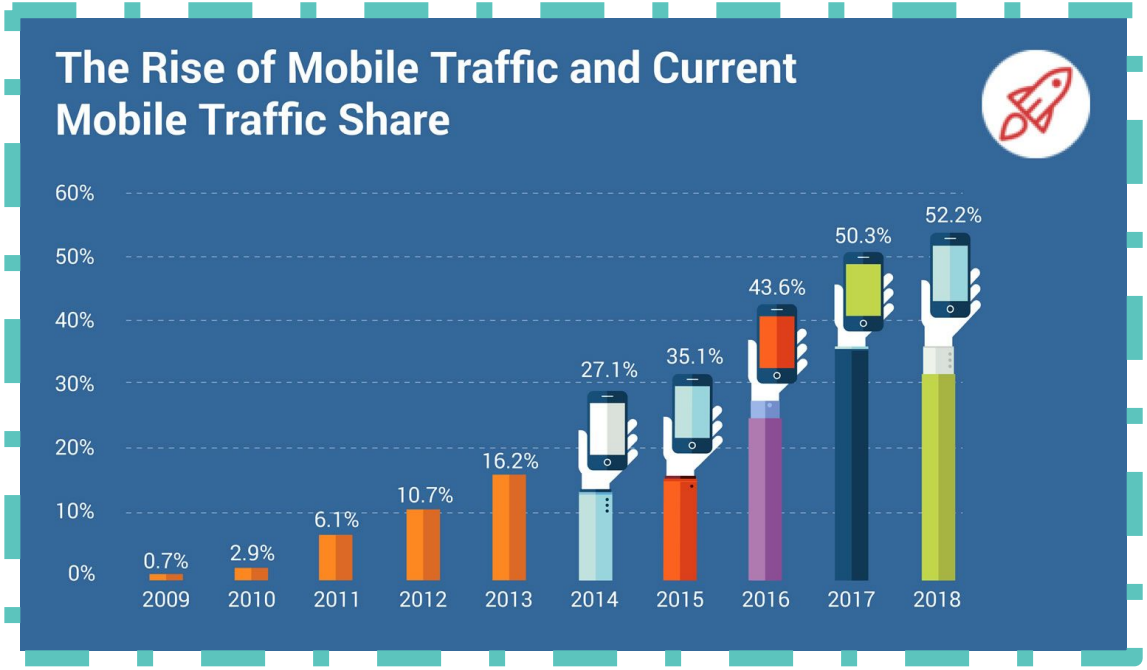


PWAs and Mobile History



It started with smartphones

Starting with the first iPhone, mobile devices have steadily taken over as the most used device for browsing the internet.



https://www.broadbandsearch.net/blog/internet-statistics

Mobile-First Design

With the realisation that web apps will most likely be viewed on devices of varying sizes and capabilities, the philosophies of Mobile-First and Responsive web design were born.

Mobile Apps

Smartphones didn’t just introduce a new way to approach web design, they also introduced mobile apps - Applications that could be downloaded and installed on a phone to give users an even better user experience.

The Native Experience

There are many features that native apps have that allow them to provide a better user experience than web apps:

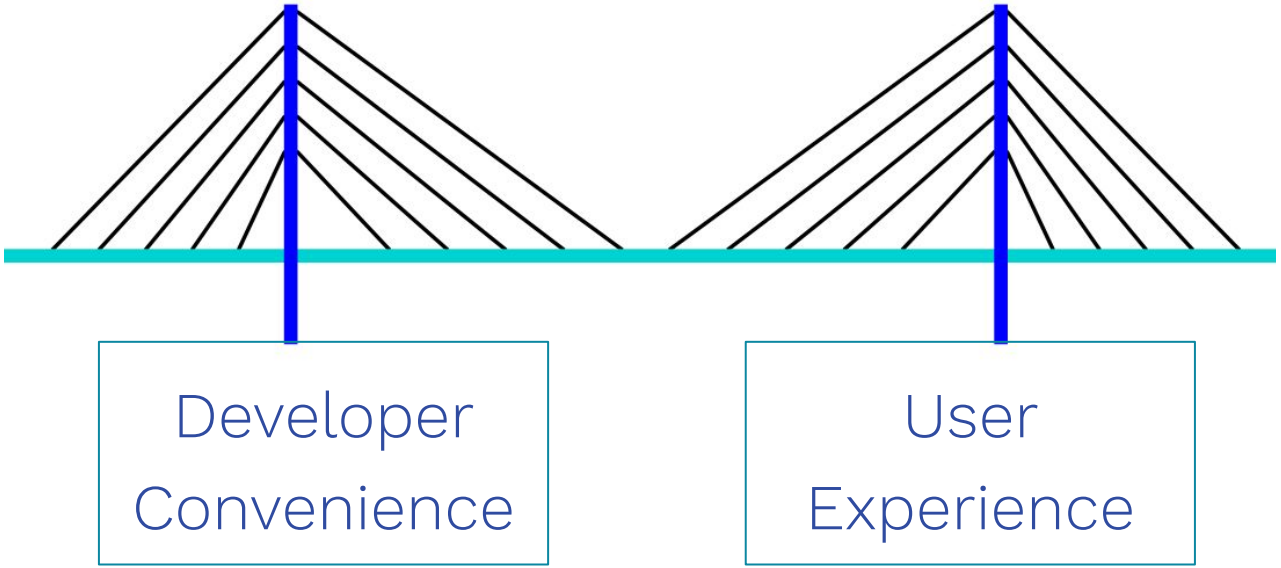
- Speed - Fast responses to user interactions.
- Launch from an icon.
- Push notifications to remind and re-engage.
- Ability to be used online or offline.
- Save data on the device and synchronize to a cloud

Downsides to Native Apps

If a wide user base is desired, apps need to be developed for multiple platforms.

- Multiple technologies to learn
- Multiple app store submissions and approvals
- Each new version of an app requires approval

A Need for a bridge



PWA to Save the Day

PWA design is a philosophy that centers around producing web applications that are widely available on all devices AND provide many of the features that Native Apps have monopolized for so long:

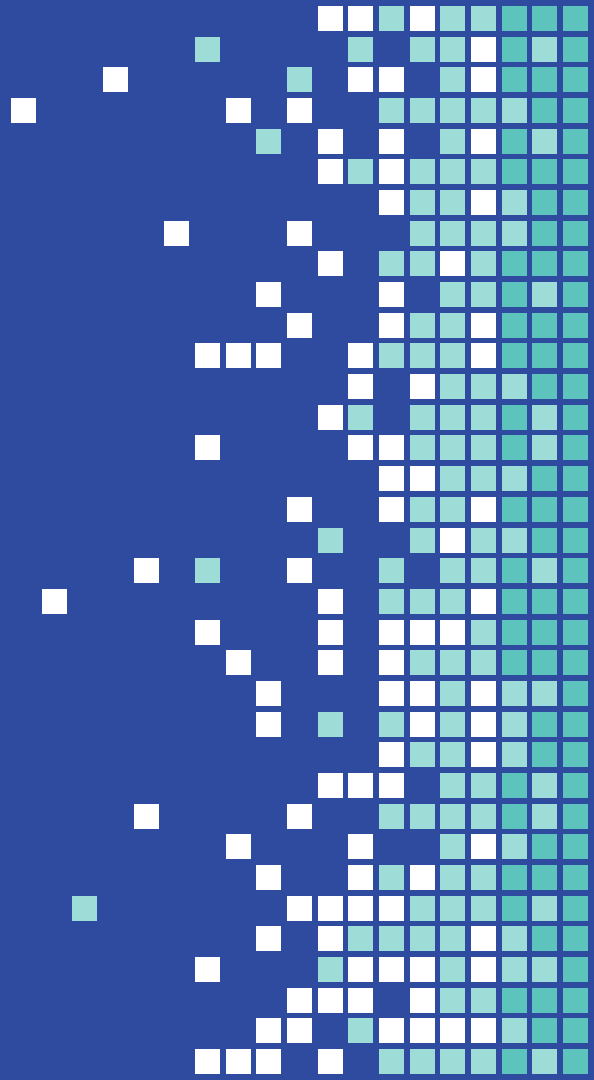
- Offline capable
- Automatic data syncing
- Push notifications
- Launch from homescreen icon
- Native look and feel
- Speed and responsiveness

Offline-First

Offline-first is an extension of mobile-first

Rather than treating a bad internet connection as an error, treat is as a legitimate state where users will still have partial, if not full, access to the app.

The PWA Toolkit



Browsers Provide PWA Tools

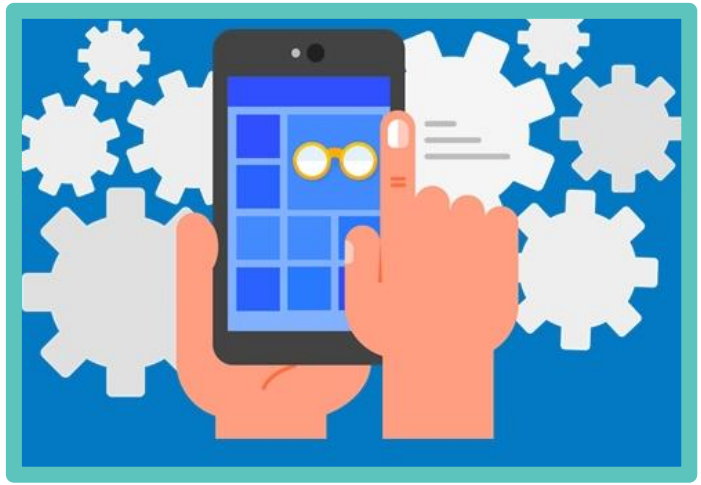
No need to download or install anything

But...

Browser support will vary for each tool

Toolkit Overview

- Web App Manifest
- Push Notifications
- Cache API
- IndexedDB
- Service Workers

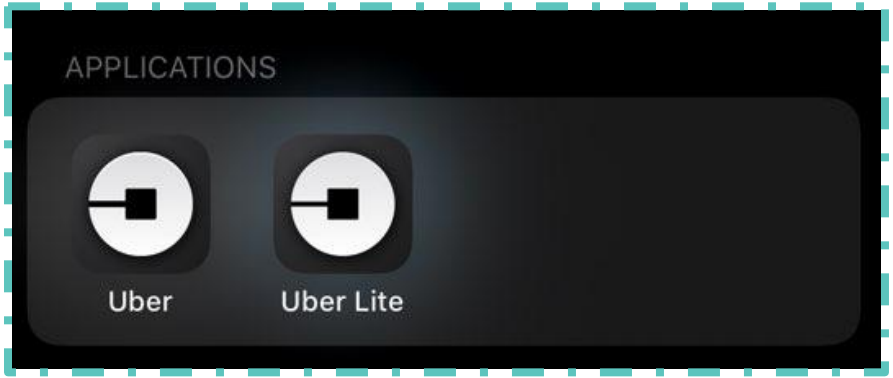


Web App Manifest

82.48%

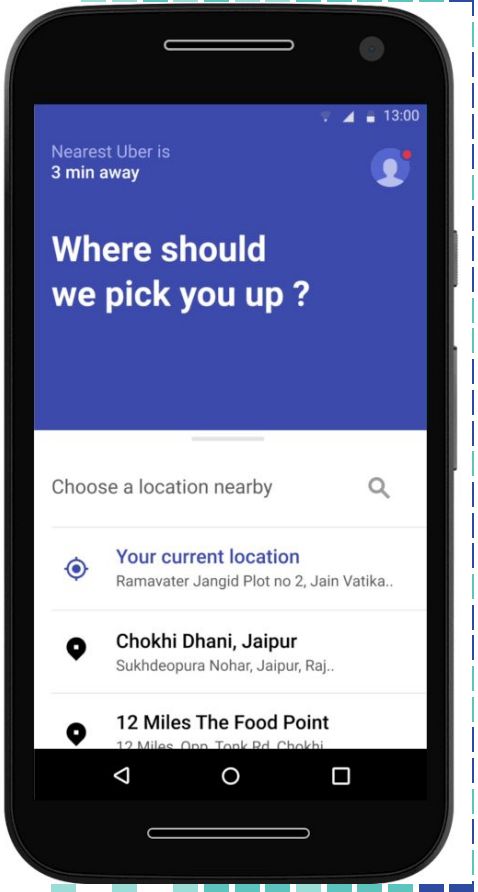
The Web App Manifest is a JSON document that provides information about an application which browsers can use to provide a better offline user experience.

```
{
  "short_name": "Todos",
  "name": "My Todos",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/images/icons-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/todos",
  "background_color": "#2F4BA0",
  "display": "stabdalone",
  "scope": "/todos/",
  "theme_color": "#5CC4BD"
}
```



Web App Manifest in use

Uber Lite



Push Notifications

Push notifications are a combination of two different tools:

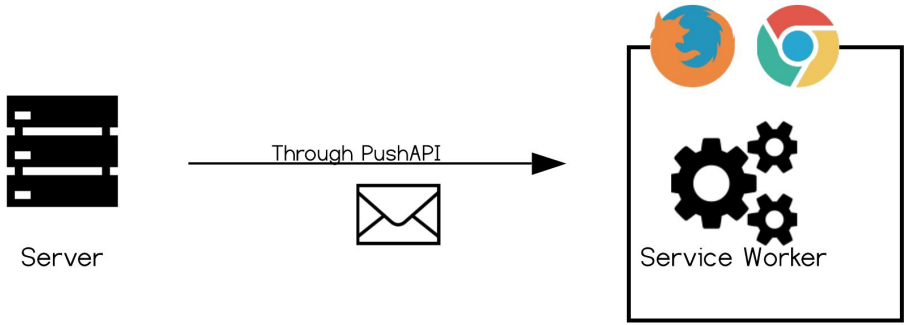
The Push API and the Notification API.

Both of these tools rely on the Service Worker

Push API

76.59%

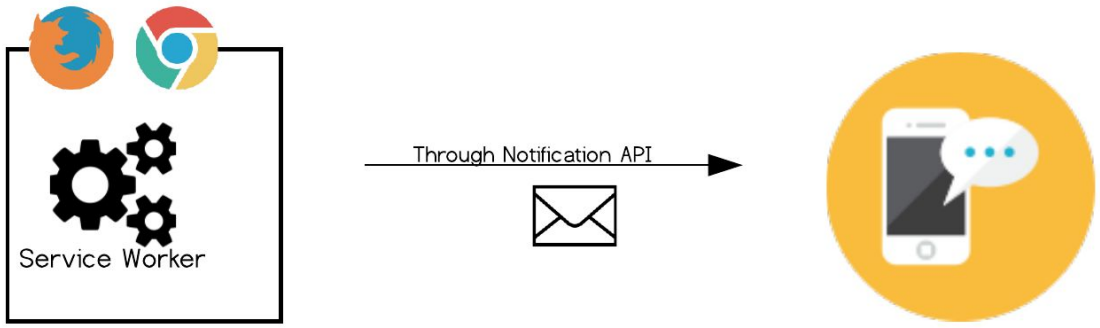
The Push API allows an application server to send messages to a service worker on a user's device without an explicit request being made. These messages can be "pushed" to a device without the app or even the browser being open.



Notification API

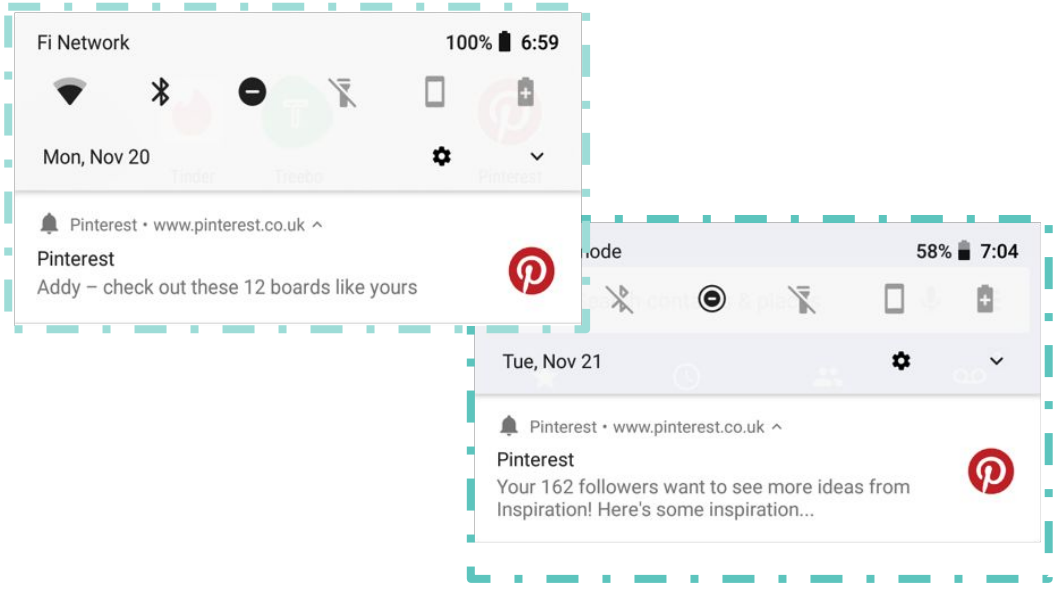
76.55%

Once the Service Worker receives the message from the Push API, it will parse through the data from the message and utilize the Notification API to interact with the operating system on the device and display a notification to the user.



Push Notifications in Use

Pinterest has been using PWA technologies since 2017



Cache API

96.64%

Designed to give developers better control over client-side caching.

Browser Cache

- Cache-control headers
- Little developer control
- Unreliable

Cache API

- Exposed API
- Full developer control
- Much more reliable

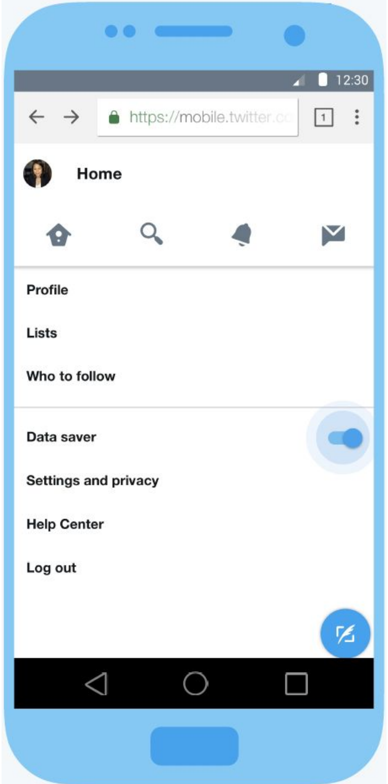
IndexedDB

82.48%

- Database that lives on the browser
- Capable of storing a lot of data - often limited
- noSQL - Stores any valid JS data type
- Uses key/value pairs for CRUD operations

Cache API and IndexedDB in use

Twitter Lite



Service Worker

88.48%

Although Service Workers (SW) are simply JS files that are installed on the browser, they are basically the glue that holds the rest of the PWA tools together. They provide many useful functions such as:

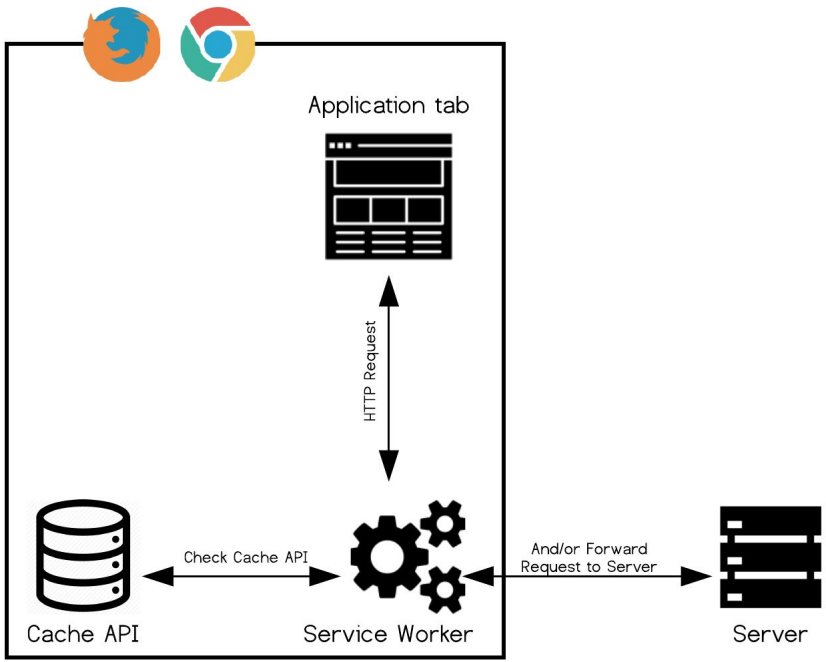
- Event handling and messaging between application code and SW
- Proxy request/response cycle
- Background sync

Service Worker Events

Active

EVENT TRIGGERED

Service Worker as a Proxy



Background Sync

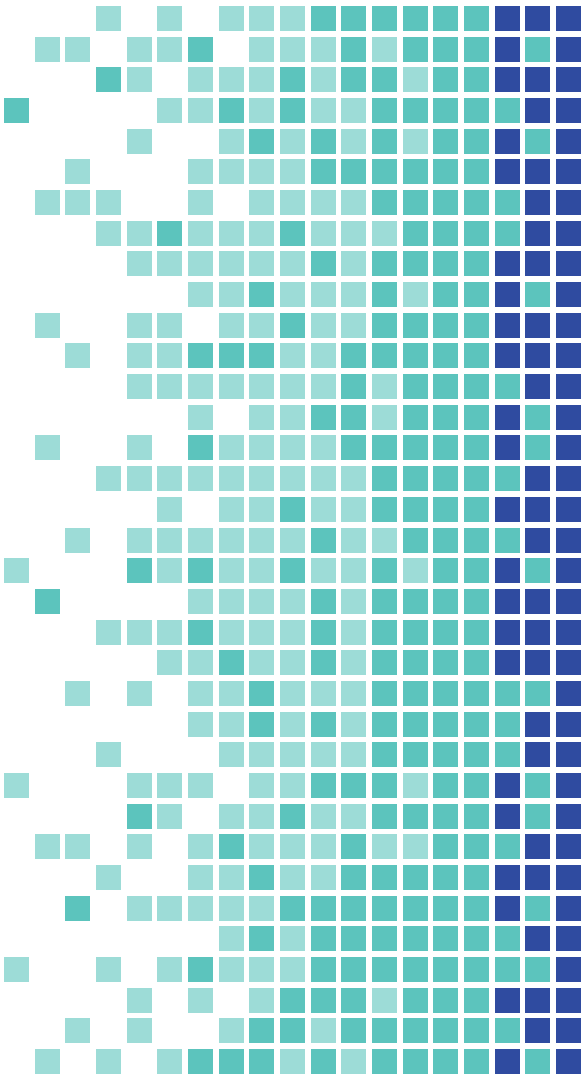
The last piece of service worker functionality we'll mention is Background Sync.

Background Sync is the key to syncing data with the server even when the app is closed.

When an application tries to send data to the server when it is offline, the service worker will take that request and put it in a queue.

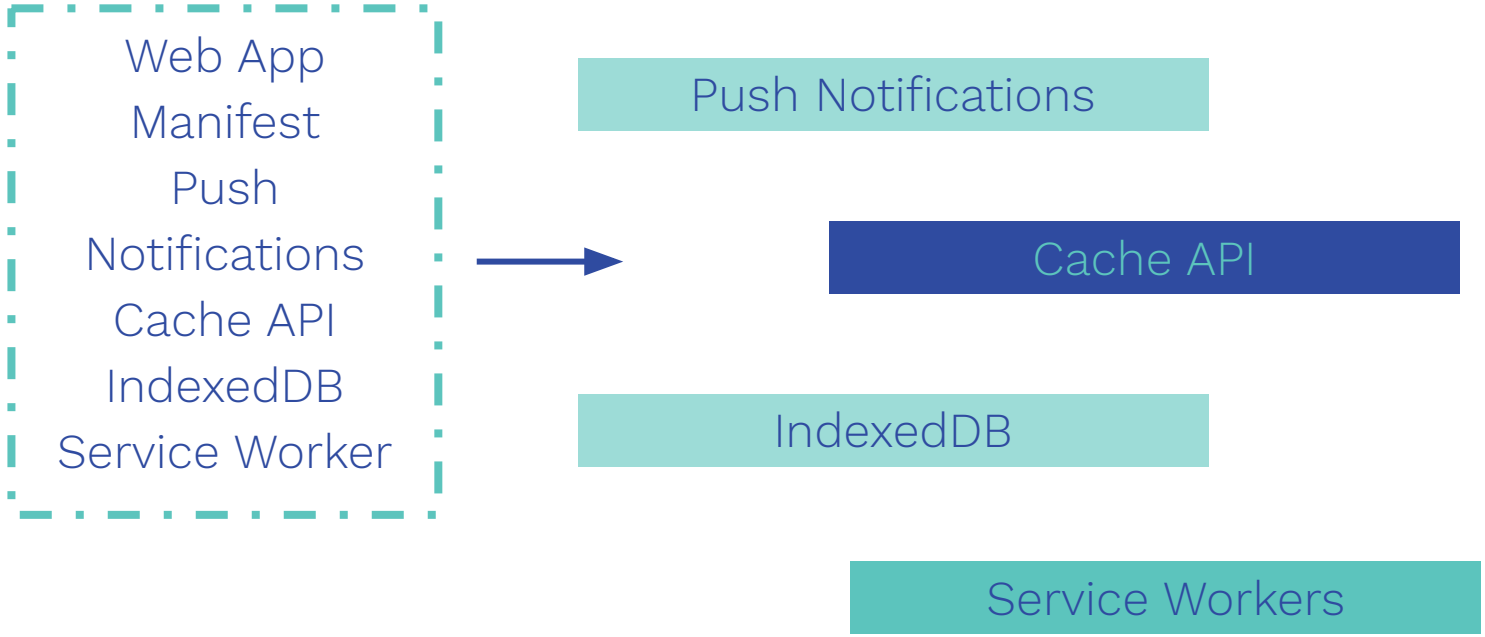
Once the device comes back online, an event will trigger and the request will be removed from the queue and sent to the server.

Toolkit Modularity



PWA is not all or Nothing

Different apps may be able to benefit from some tools, but not others.



Roadmap

Section 1:

- ❑ Introduction to Progressive Web Apps
- ❑ The Progressive Web App Toolkit

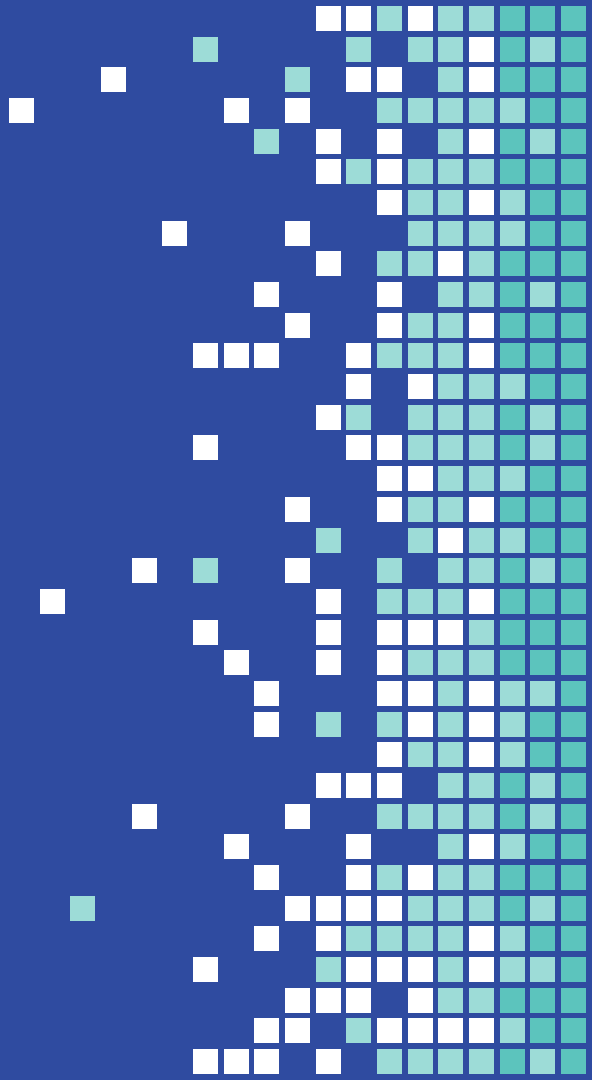
Section 2:

- ❑ Offline FIrst Data Management
- ❑ Frameworks / Solutions

Section 3:

- ❑ CushionDB
- ❑ Challenges and Considerations

Offline-First Data Management



Where is client side data storage most valuable?

Apps with public-facing data

Example apps

Value of client side data:

Relies on

Apps with group-facing data

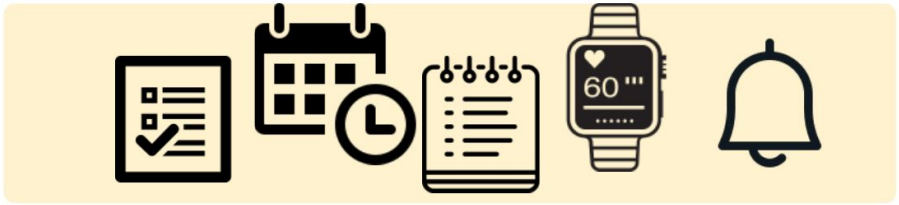
Example apps

Value of client side data:

Relies on

Apps with private data only

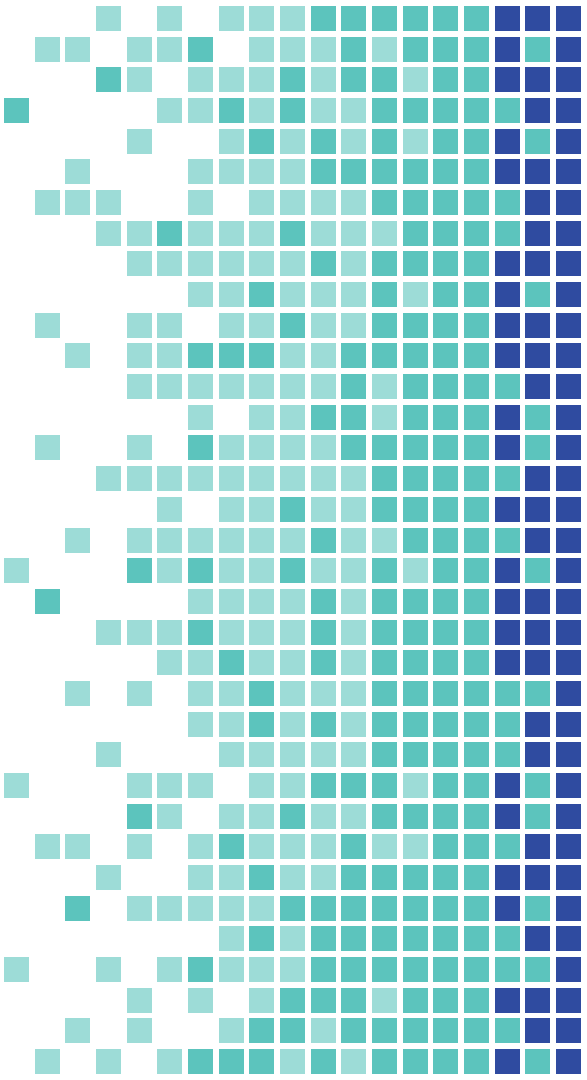
Example apps



Value of client side data:

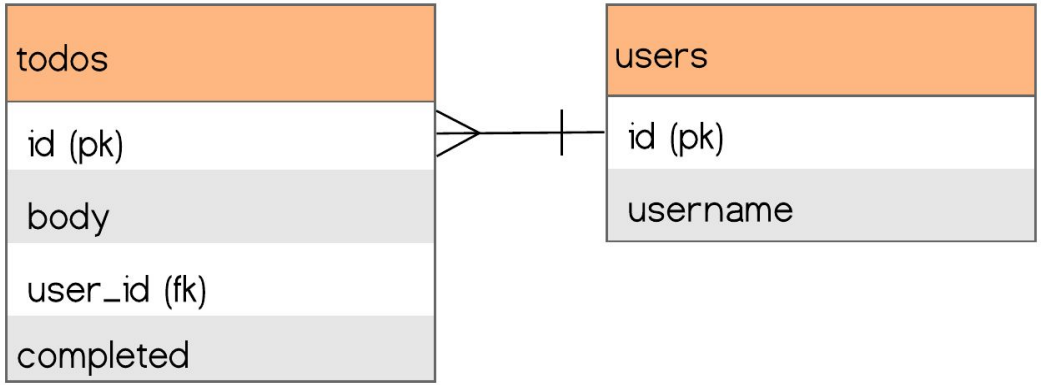
Relies on

What would a single user app data structure look like?



Traditional - Relational

The data structure for a todo app in the relational world might end up looking a lot



To get all of a user's todos:

```
SELECT *
FROM todos
WHERE user_id = ?;
```

Alternative - noSQL

Data structure using the document-database design of IndexedDB

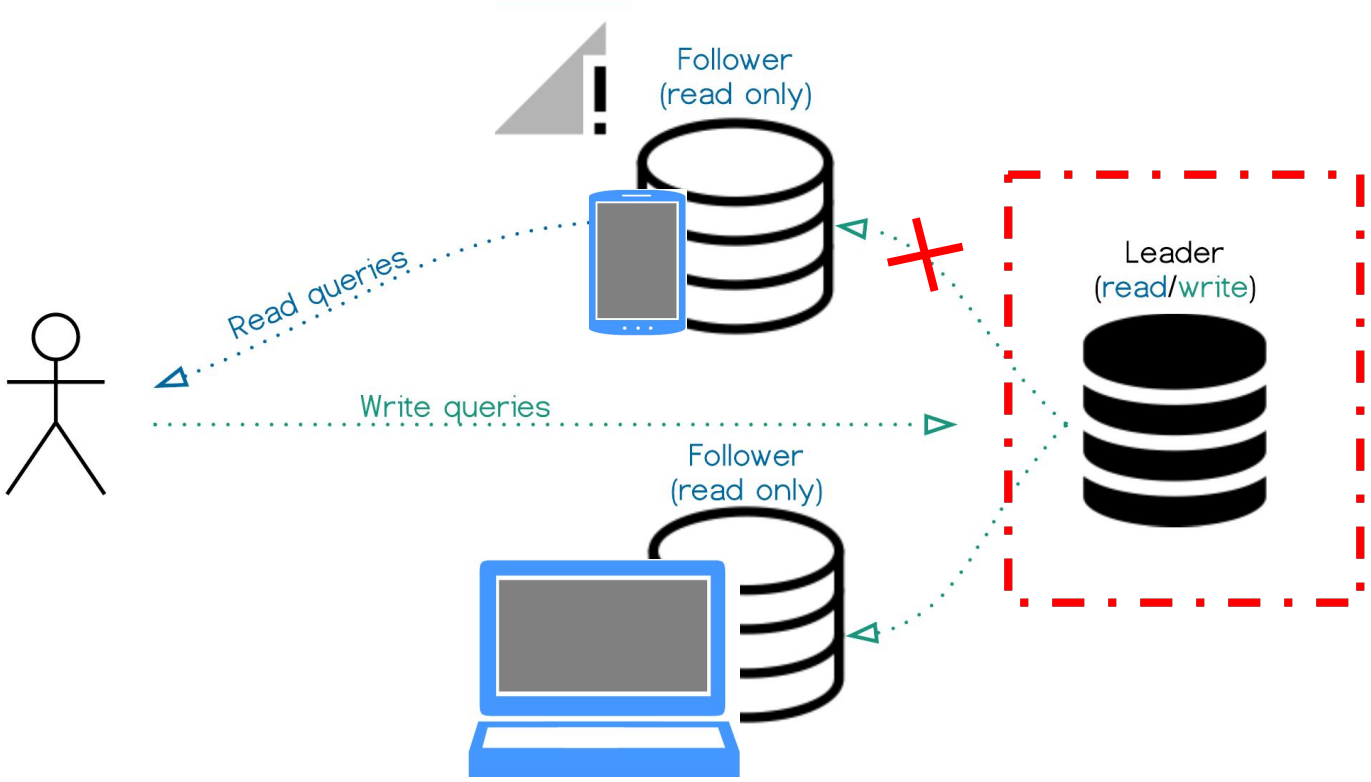
```
username: [
  id (integer): {
    body: (string),
    completed: (boolean)
  },
]
```

To get all of a user's todos:

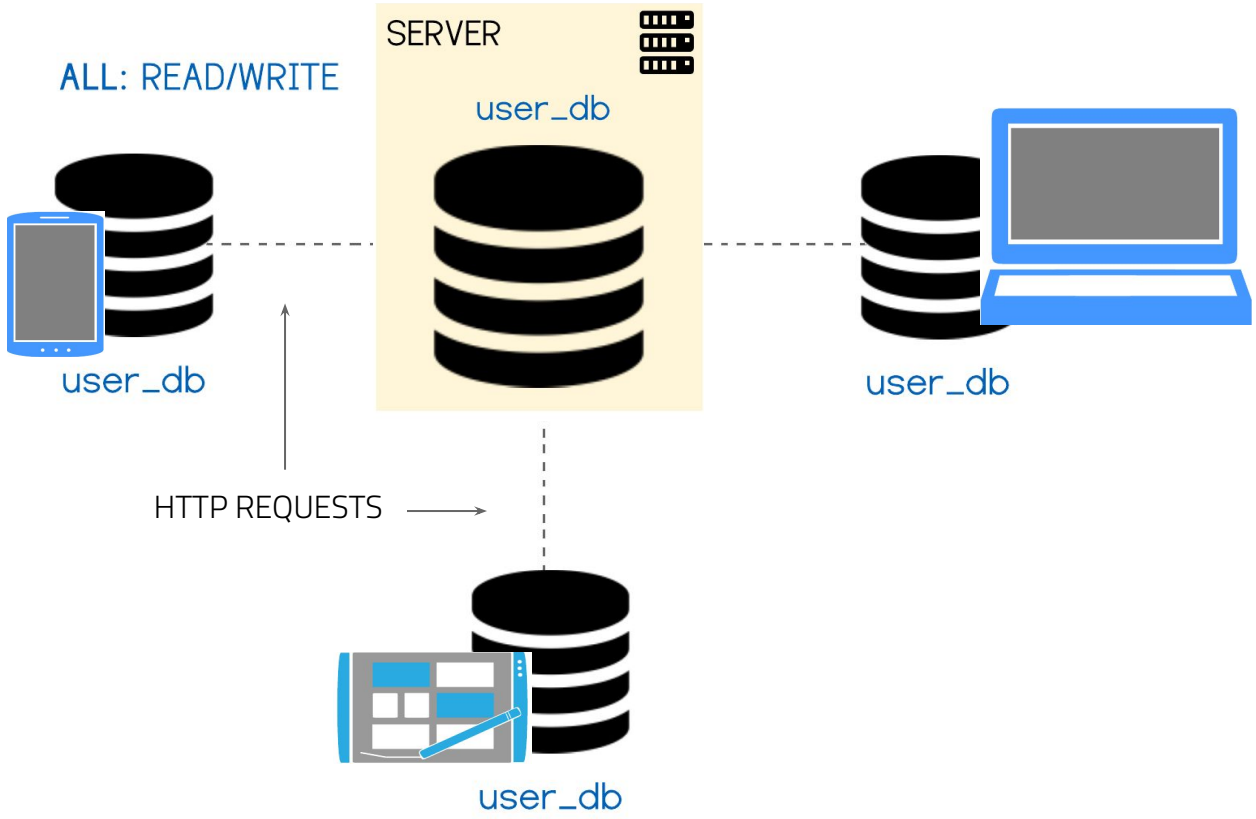
```
user_todos.getAll();
```

What Would Replication look like?

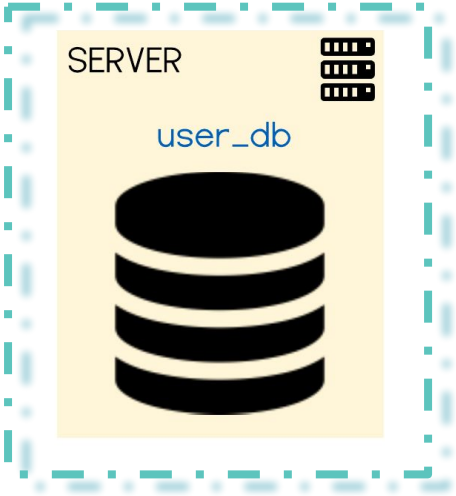
Traditional - Leader/Follower



Alternative - Multi-Leader



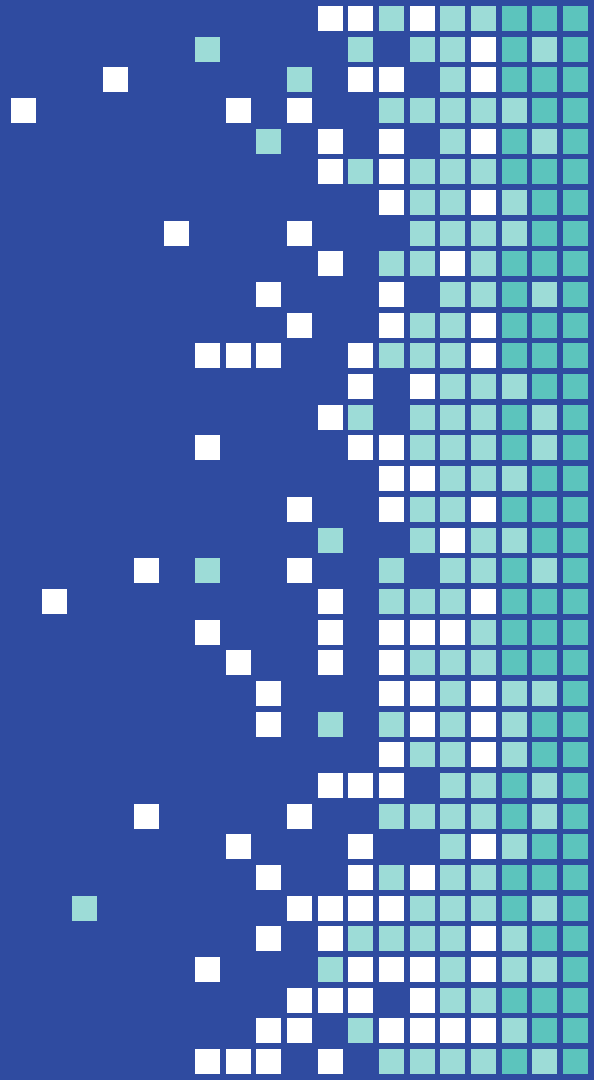
Alternative - Multi-Leader



Authentication for DB per User

Data Structure Summary

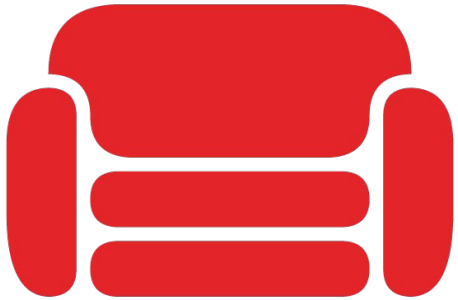
Solutions



> *Friends don’t let friends design their own sync engines*

PouchDB & CouchDB

CouchDB API



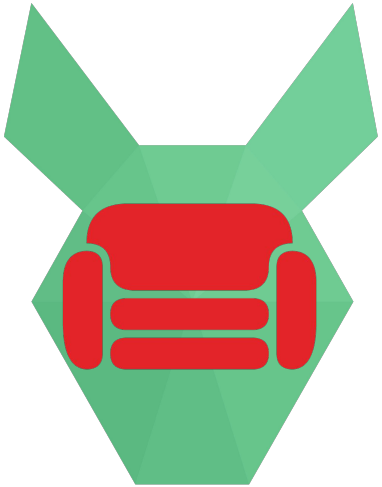
This query in MongoDB:

```
db.todos.find( {} )
```

Turns into:

```
curl -X GET http://127.0.0.1:5984/todos/_all_docs
```

PouchDB



Sample PouchDB Code

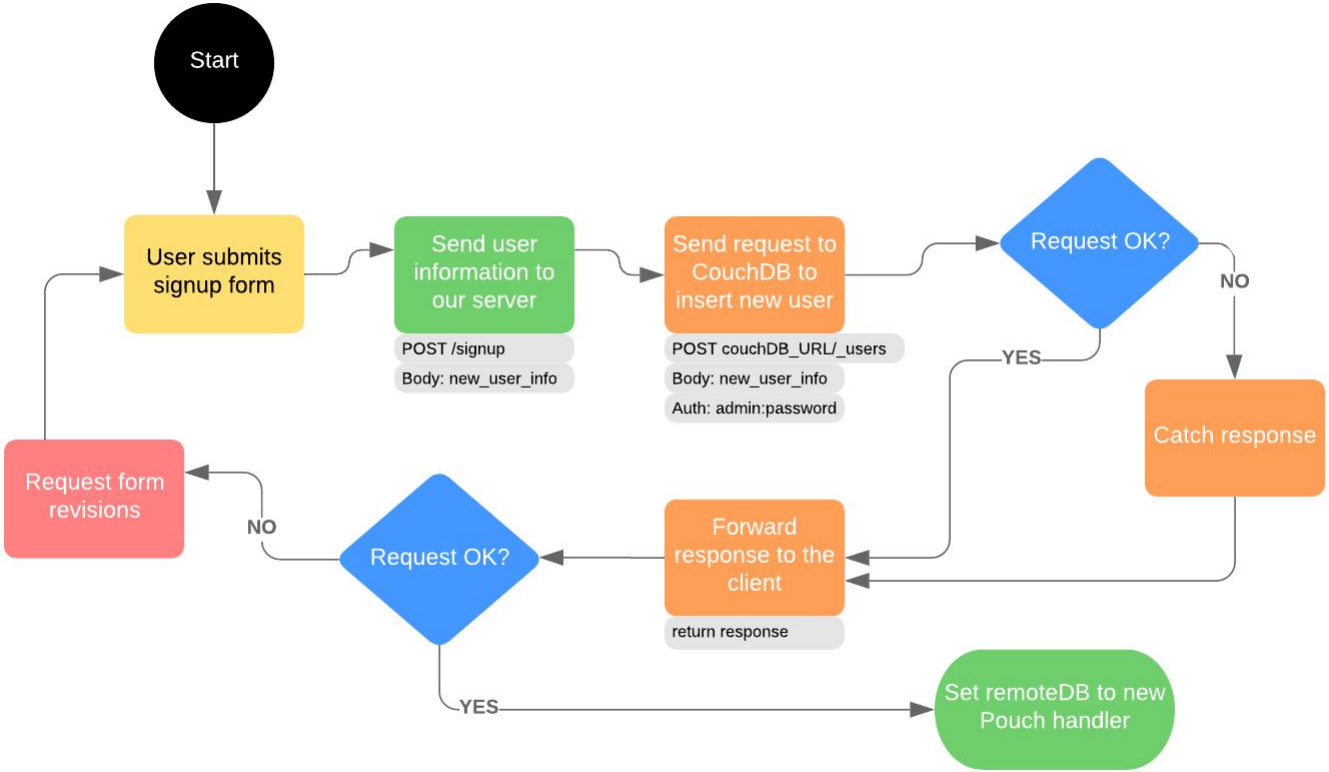
Syncing with CouchDB

```
import PouchDB from 'pouchdb';

// Create or connect to local database (IndexedDB)
const localDB = new PouchDB('todos');
// Create or connect to remote database (CouchDB)
const remoteDB = new PouchDB('http://127.0.0.1:5984/user1_todos');

// One time sync from remote to local database
localDB.replicate.from(remoteDB);

// Setup localDB to constantly replicate
// changes to remoteDB
localDB.changes({
  since: 'now',
  live: true
}).on('change', () => {
  localDB.replicate.to(remoteDB);
});
```

Pouch & Couch

Pros	Cons
Takes care of replicating the server and authenticating admins (the users in our case) remotely	Complicated manual setup
Can be configured to set up a database per user created automatically	Multiple HTTP requests required for some operations are hard to debug
Communicated with via HTTP entirely - does not require a seperate server	In most use cases would require a super-admin which necessitates setting up a server and makes some operations very convoluted.
Highly modular	

Simple User Signup with Pouch & Couch



Hoodie

A no back-end offline first data management tool

Sample Hoodie Code

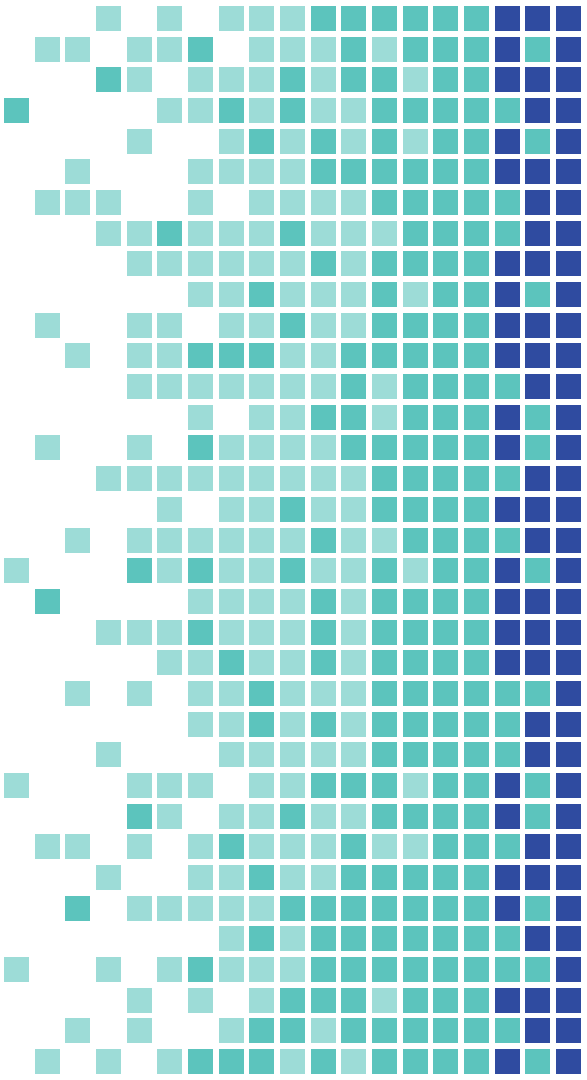
Setting up replication and signing up

```
const Hoodie = require('@hoodie/client');
const hoodie = new Hoodie({
  // Set host where Hoodie server runs
  url: 'http://127.0.0.1:8000',
  // Require the PouchDB constructor you will use
  PouchDB: require('pouchdb'),
  // Supply address of remote CouchDB
  remoteBaseUrl: 'http://127.0.0.1:5984'
});

// Create a new user
hoodie.account.signUp({
  username: 'user',
  password: 'secret'
}).then(accountAttr => {
  console.log(`Signed in as ${accountAttr.username}`);
}).catch(err => {
  console.log(err);
});
```

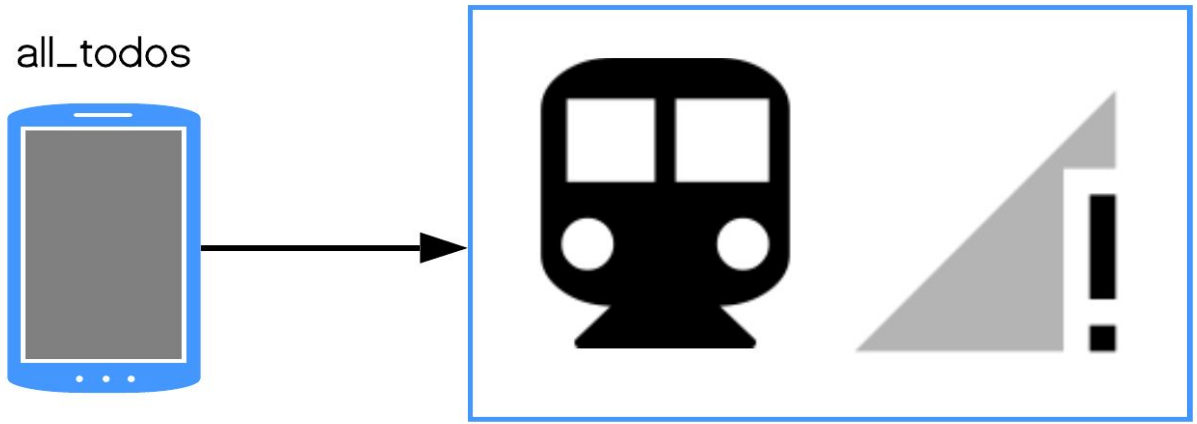
Hoodie

Pros	Cons
Simplifies configuration	Not PWA compliant
Simple API for user authentication and management	Manual installation and complicated to deploy
Simplifies setup	Not as modular

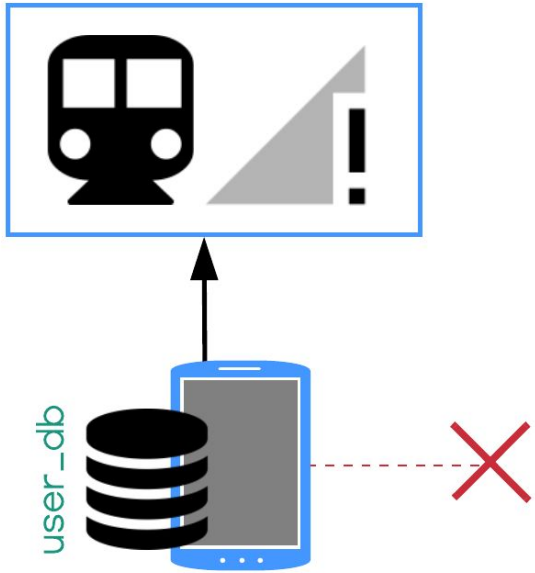
Our Todos App in Action

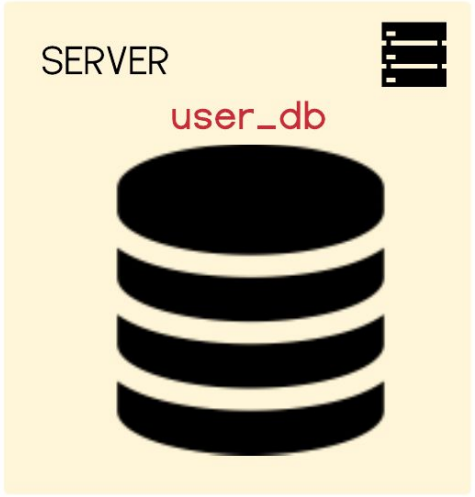


Using the app offline



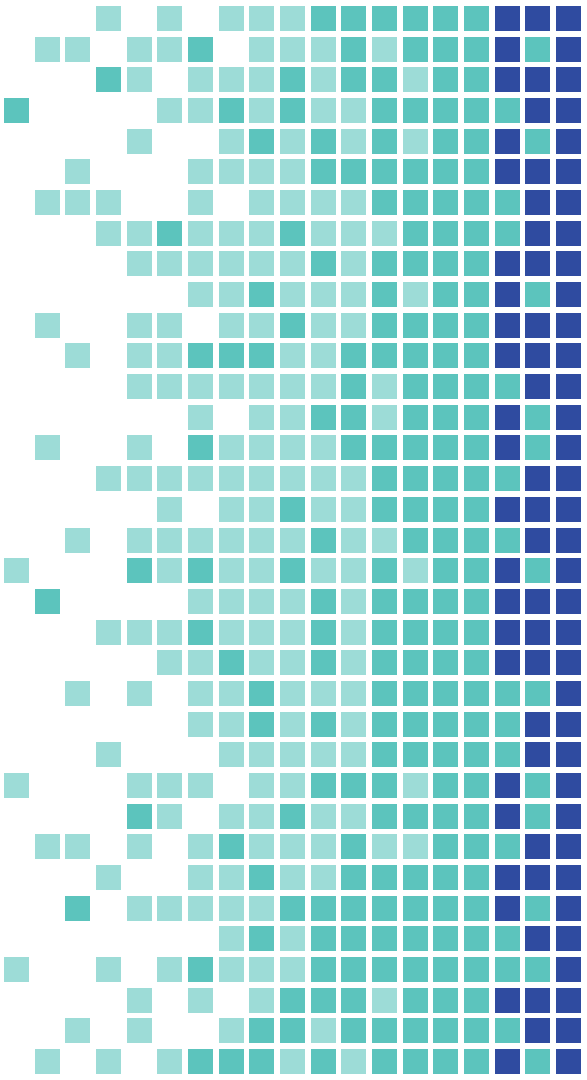
App does not sync

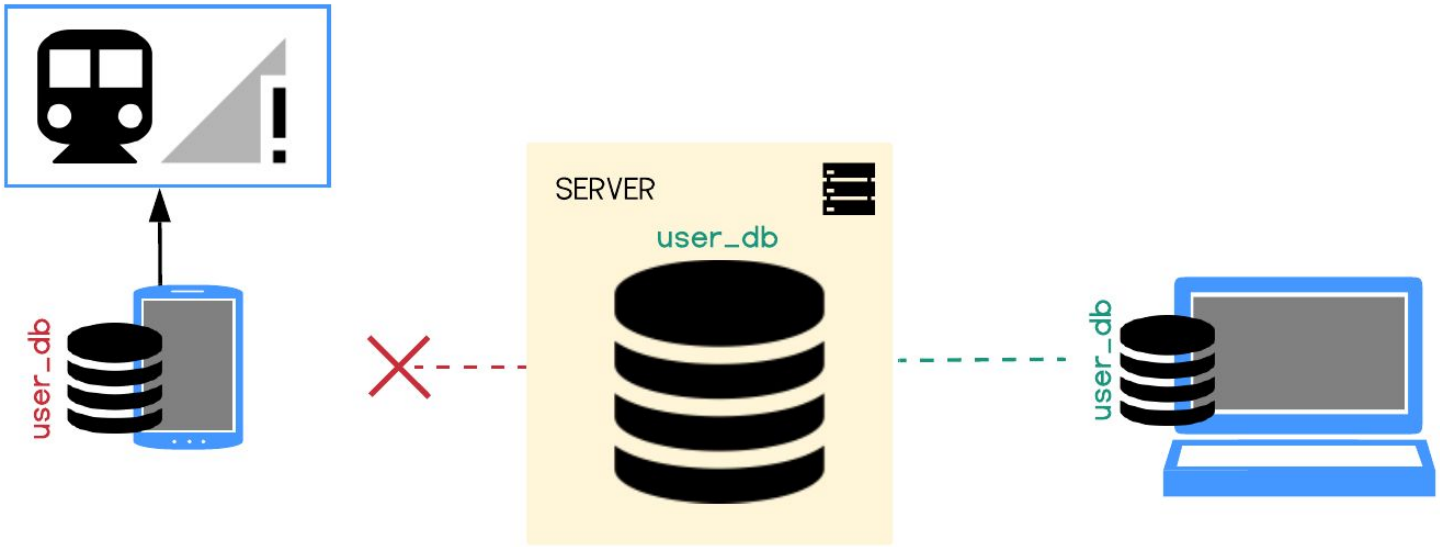




How are you going to get them in sync?

Let’s consider another scenario





Scenario cont...

The Problem

Sending and Receiving updates are not completely automated and requires manual intervention by the end user.

Roadmap

Section 1:

- Introduction to Progressive Web Apps
- The Progressive Web App Toolkit

Section 2:

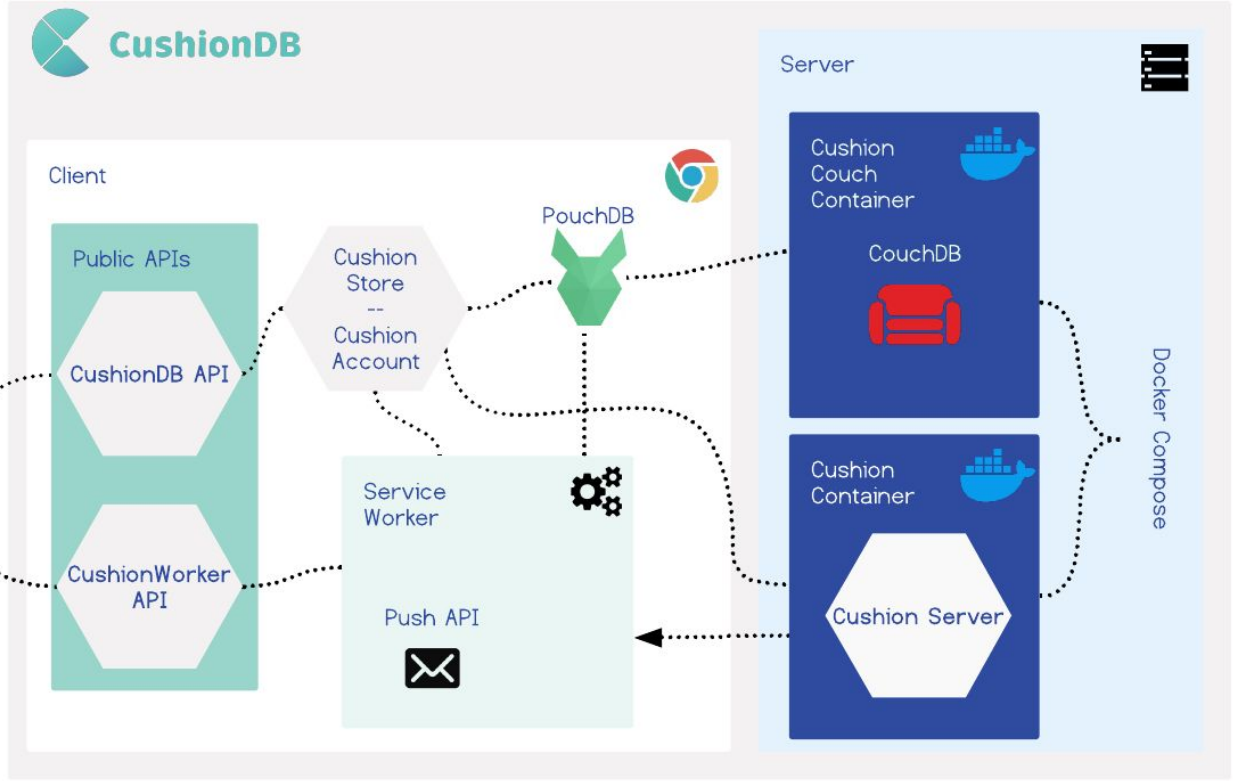
- Offline FIrst Data Management
- Frameworks / Solutions

Section 3

- CushionDB
- Challenges and Considerations

CushionDB

CushionDB simplifies building Progressive Web Apps by providing an easy to use API for authentication and data management.

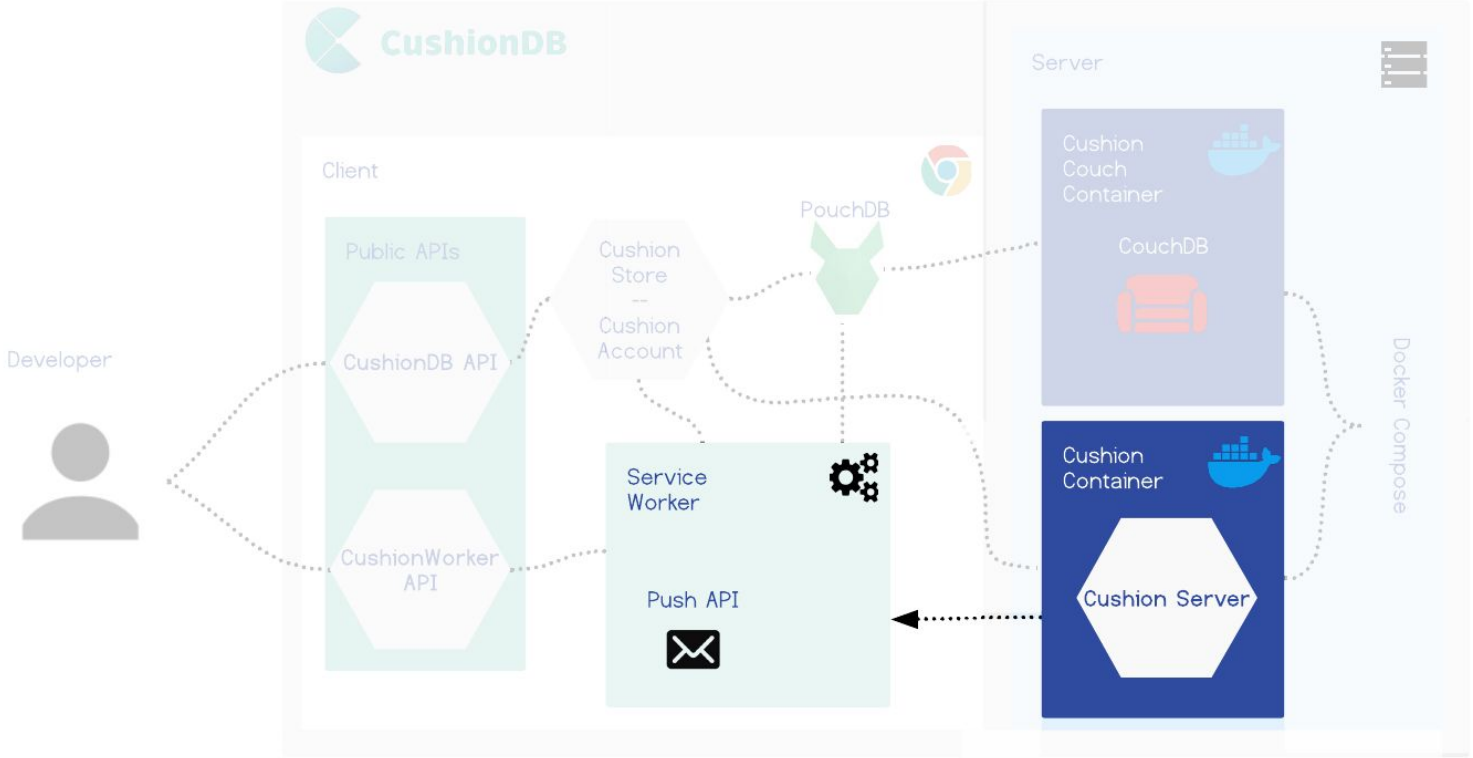


CushionDB Architecture

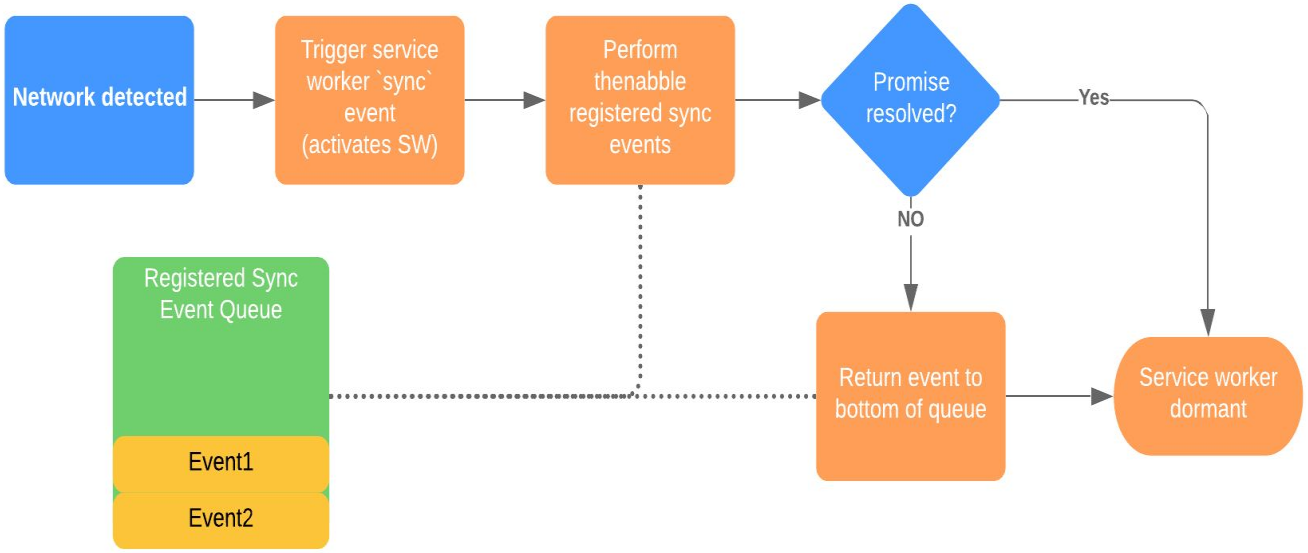
CushionDB's approach to updating server

How does CushionDB automate updating the server without any end user intervention?

Syncing through service worker



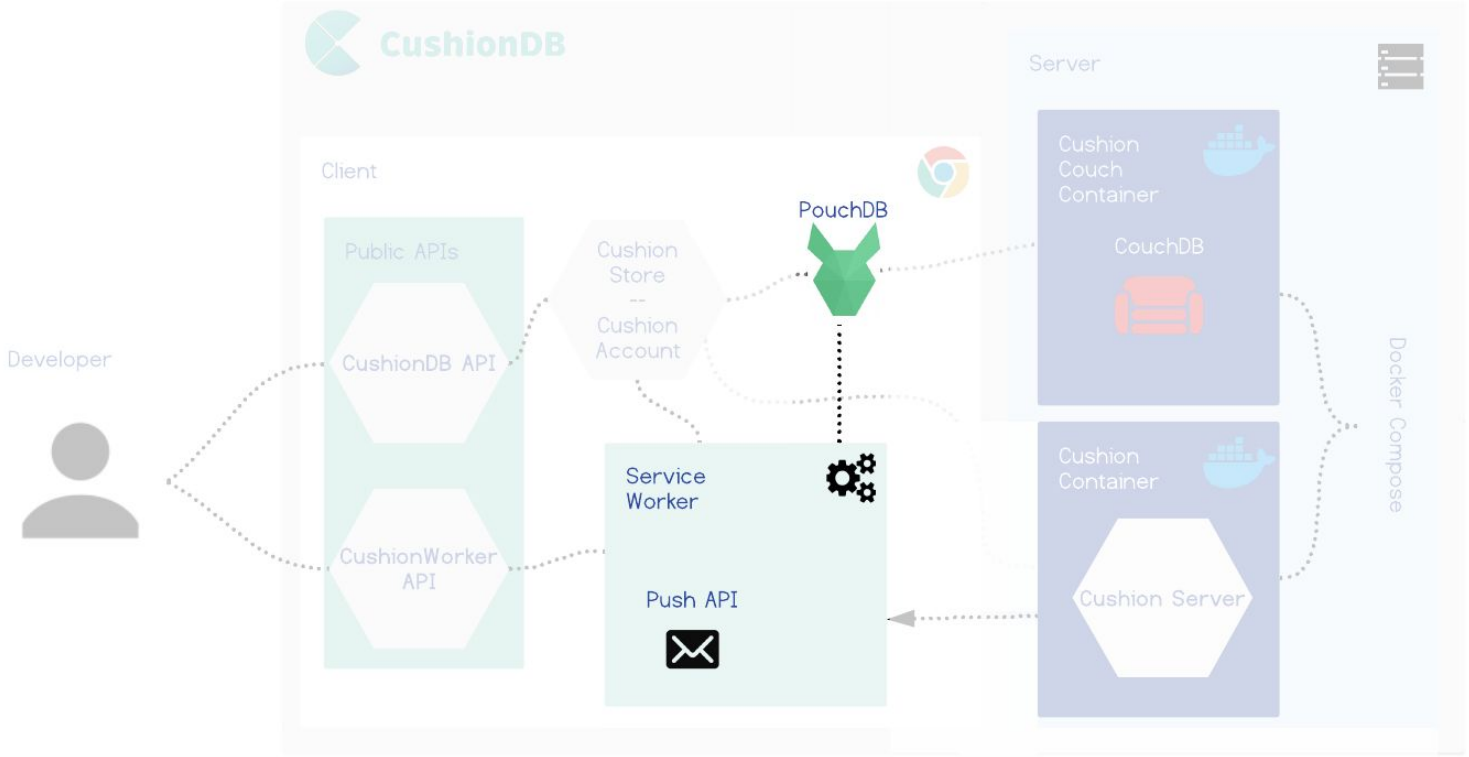
Syncing through service worker



CushionDB's approach to updating clients

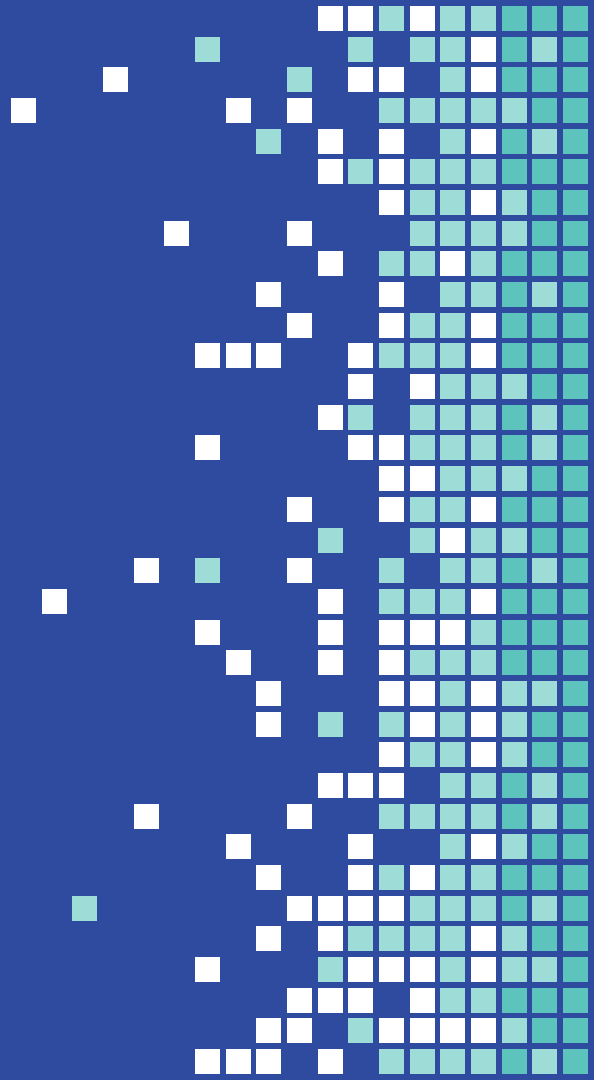
How does CushionDB automate updating other devices without any end user intervention?

Replication with Push Notifications



PWA compliant

Working with CushionDB



Requirements

- Nodejs 10.16 LTS
- Docker

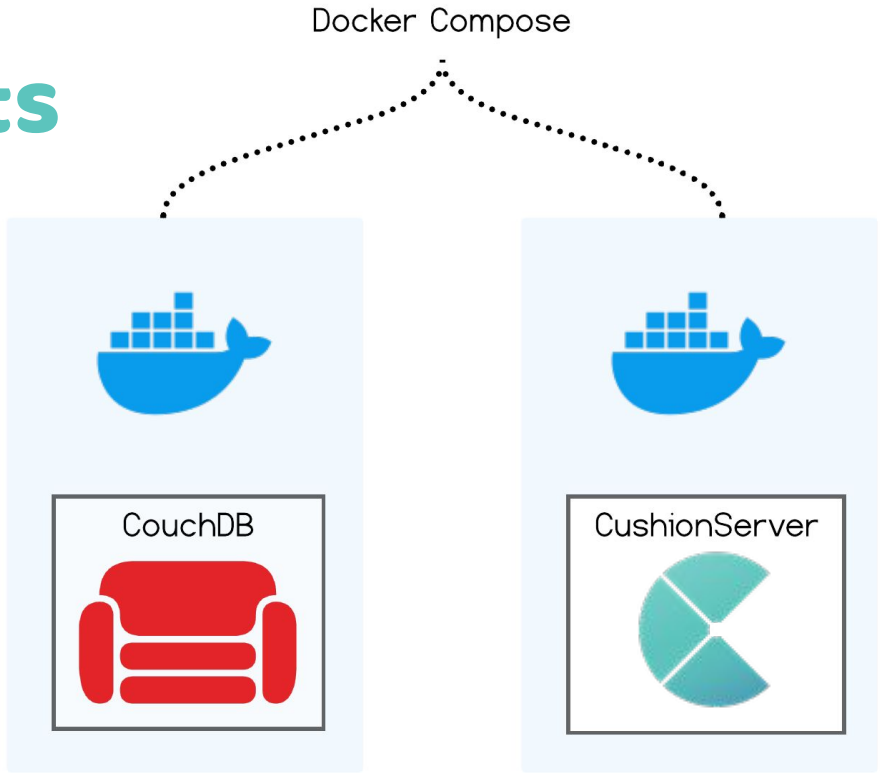
Installation

- Client side *(CushionClient)*
- Server side *(DockerCushionContainer)*

```
✔ Cleanup
✔ Installing dependencies
∴ Running tests
  Bumping version
  Publishing package
  Pushing tags
```

Client Installation

Server Components

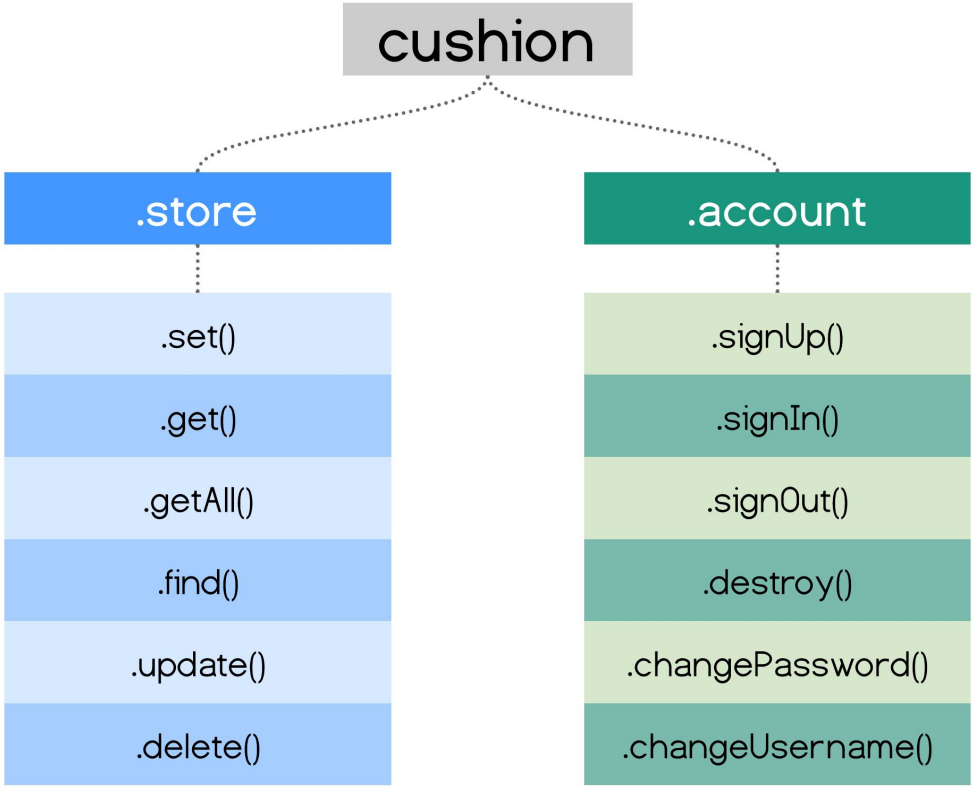


Server installation

```
version: "3"
services:
  app:
    image: cushion-server:latest
    ports:
      - "3001:3001"
    networks:
      backend:
        aliases:
          - backend
    depends_on:
      - "db"
  db:
    image: cushion-couch:latest
    volumes:
      - data:/opt/couchdb/data
    ports:
      - "5984:5984"
    env_file:
      - .env/.couch_credentials.env
volumes:
  data:
```

CushionDB docker-compose.yml sample

CushionDB API Sample



CushionDB

Developer API sample:

```
import Cushion from 'cushiondb';

const cushion = new Cushion();

// Sign up a new user
cushion.account.signUp({
  username: 'user',
  password: 'secret'
}).then(res => {
  // Successful user sign up
}).catch(err => {
  console.log(err);
})

// Add a todo to database
cushion.store.set({
  title: 'todo',
  completed: false
}).then(docId => {
  // Todo successfully added
}).catch(err => {
  console.log(err);
});
```

In the background:

- Creates a user
- Sets up user authentication
- Creates a local and remote data store for the user
- Creates and Stores a document
- Syncs updates to the server

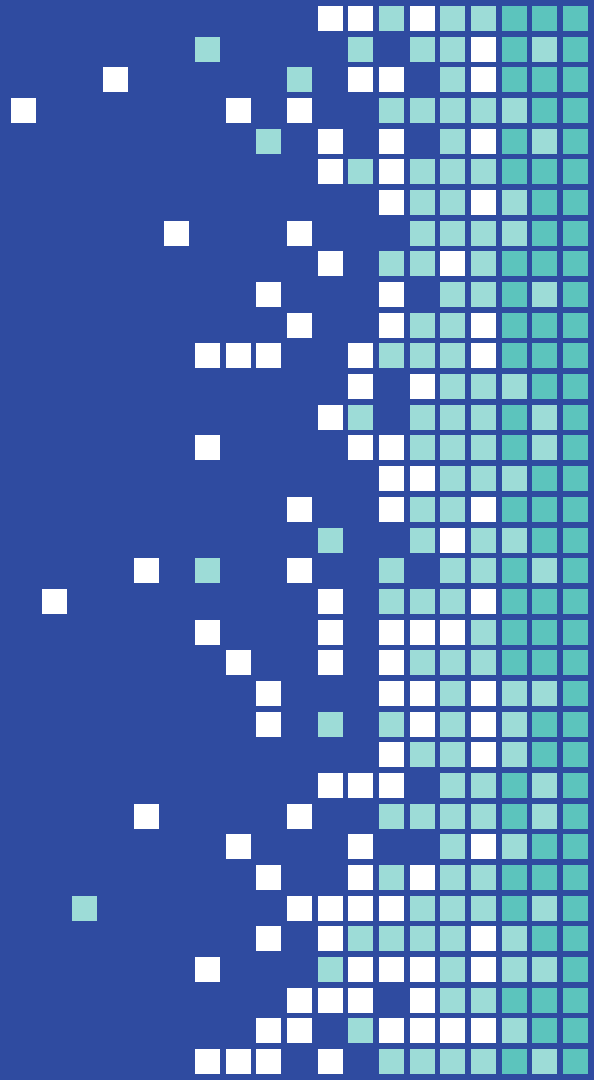
Summary

- Enables offline first functionality
- Abstracts data management and authentication
- Automates data synchronization
- Simplifies building native-like web apps

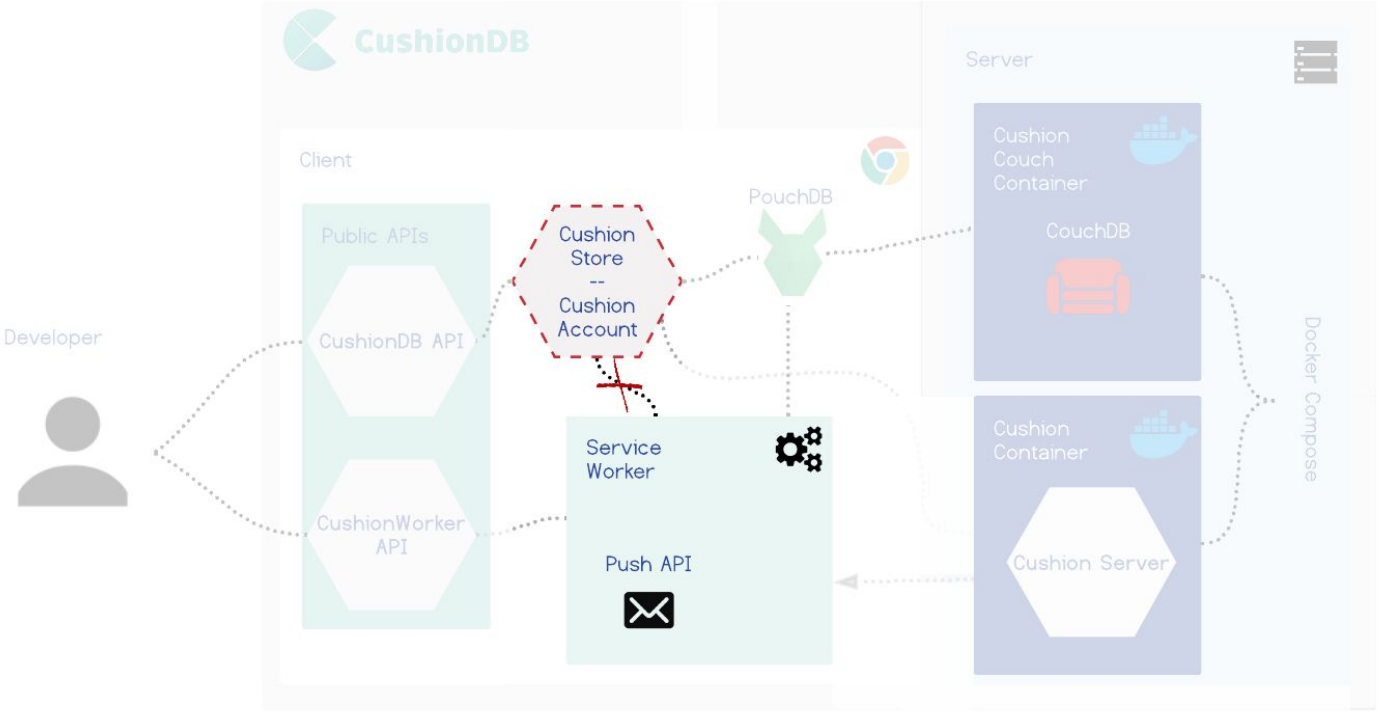
Limitations

- Chrome Browser only
- Suitable for personal/private type of apps
- It is not extendable (yet)
- In the early stages of development - not production ready

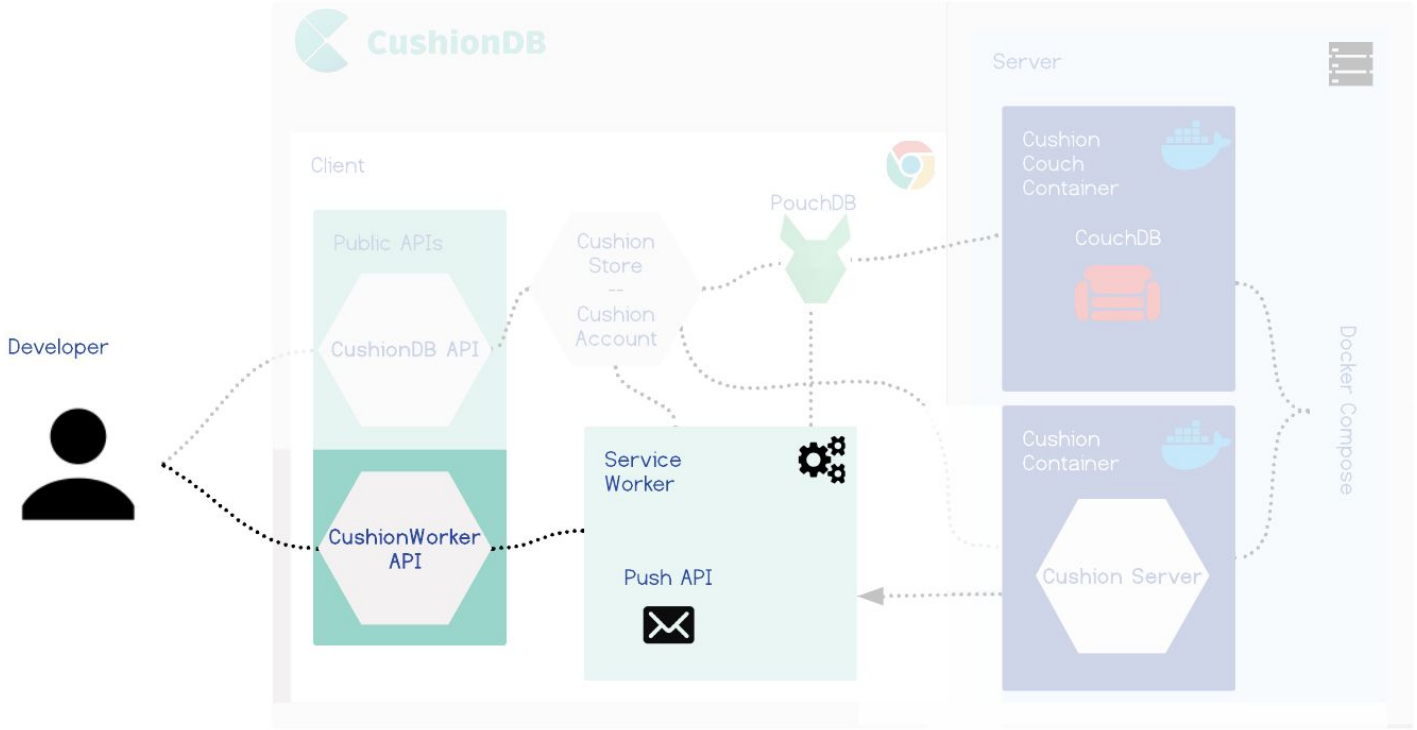
Challenges and Considerations



Service Worker Accessing Data when App is closed



Service Worker Event Hijacking



Cushion Worker API

- addPushEvent()
- addSyncEvent()
- addMessageEvent()

Thank you