CouchDB and PouchDB

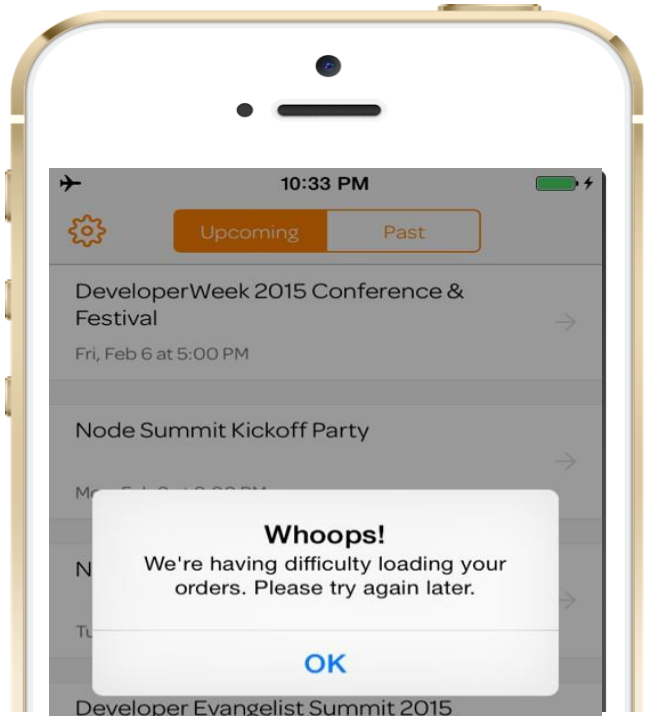
Enrique Barra

Intro

Not just mobile first…

…Offline first

“Because being offline shouldn´t be an error condition”



NoSQL document databases

- The origin was Lotus Notes
- Data model: collections of documents (JSON, XML, BSON) that contain key-value pairs
- Examples: CouchDB, MongoDB
- Good at:
  - Can work with complex data, admit nested document (objects)
  - Correspond with the way objects and properties are modeled in Object Oriented Programming Languages
  - Web oriented: CRUD interface

```
{
  "name": "Enrique",
  "occupation": "Teacher",
  "age": 35,
  "hobbies": [
    "basketball",
    "swimming",
    "programming"
  ],
  "_id": "1234",
  "_rev": "1-3e3be7e7b331eeea2d4221b5193"
}
```


CouchDB
relax

CouchDB

```
Apache CouchDB has started. Time to relax.
```

- http://guide.couchdb.org/draft/why.html

- Apache CouchDB is a distributed, fault tolerant and schema-free document-oriented database accesible via a RESTful HTTP/JSON API

- Reasons to use:
  - Learn CouchDB should be **natural** for anyone that has worked as web developer
  - Architecture is **distributed and fault tolerant**
  - It **scales**, CouchDB imposes some limitations, but are limitations that are there for something, to favor scaling

Where is CouchDB used?

- CERN, BBC, Skechers, …
- http://readwrite.com/2010/08/26/lhc-couchdb/
- https://wiki.apache.org/couchdb/CouchDB_in_the_wild

Other characteristics of CouchDB

- **Views** are how data is queried using CouchDB
- **Queries**: Javascript as a query language, using **MapReduce**

```
// Map
function(doc) {
  emit(doc._id, 1);
}
// Reduce
function(keys, values, rereduce) {
  return sum(values);
}
```

- **Replication:** unidirectional, bidirectional, continuous, filtered
- **Conflicts**: CouchDB manages conflicts automatically
- Administration of the database is done through a web interface called **Futon**

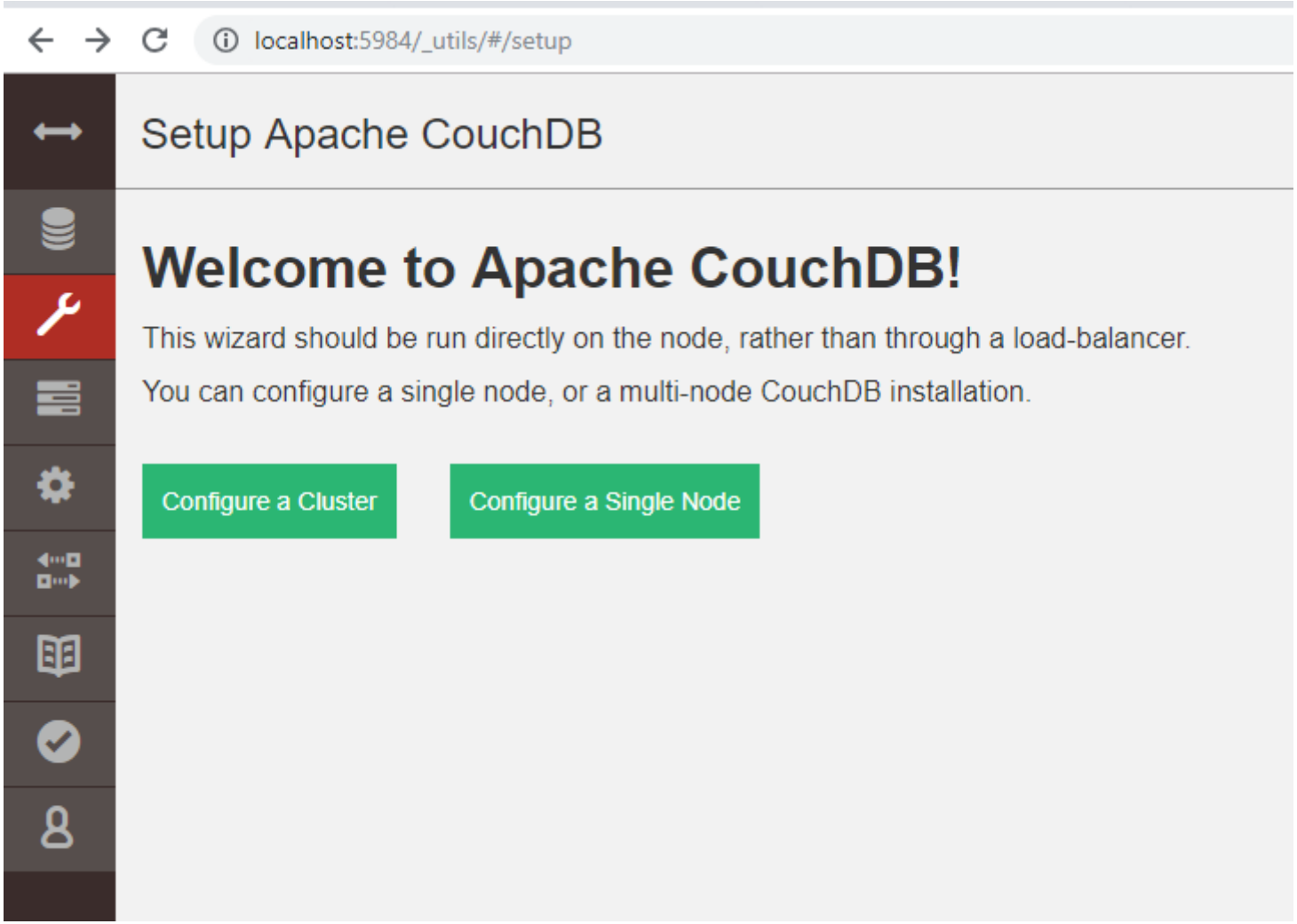
CouchDB Installation with docker

- http://docs.couchdb.org/en/2.3.1/install/docker.html
- https://hub.docker.com/_/couchdb/

- docker run -p 5984:5984 -d couchdb

- These images are built using Debian 8 (jessie), expose CouchDB on port **5984** of the container, run everything as user **couchdb**, and support use of a Docker **volume** for data at **/opt/couchdb/data**

Configure a single node



Data…

Database name

Create Database

{ } JSON

Databases

Setup

Active Tasks

Configuration

Replication

Documentation

Verify

Your Account

Name	Size	# of Docs	Actions
_global_changes	494 bytes	2	
_replicator	2.3 KB	1	
_users	2.3 KB	1	

CouchDB

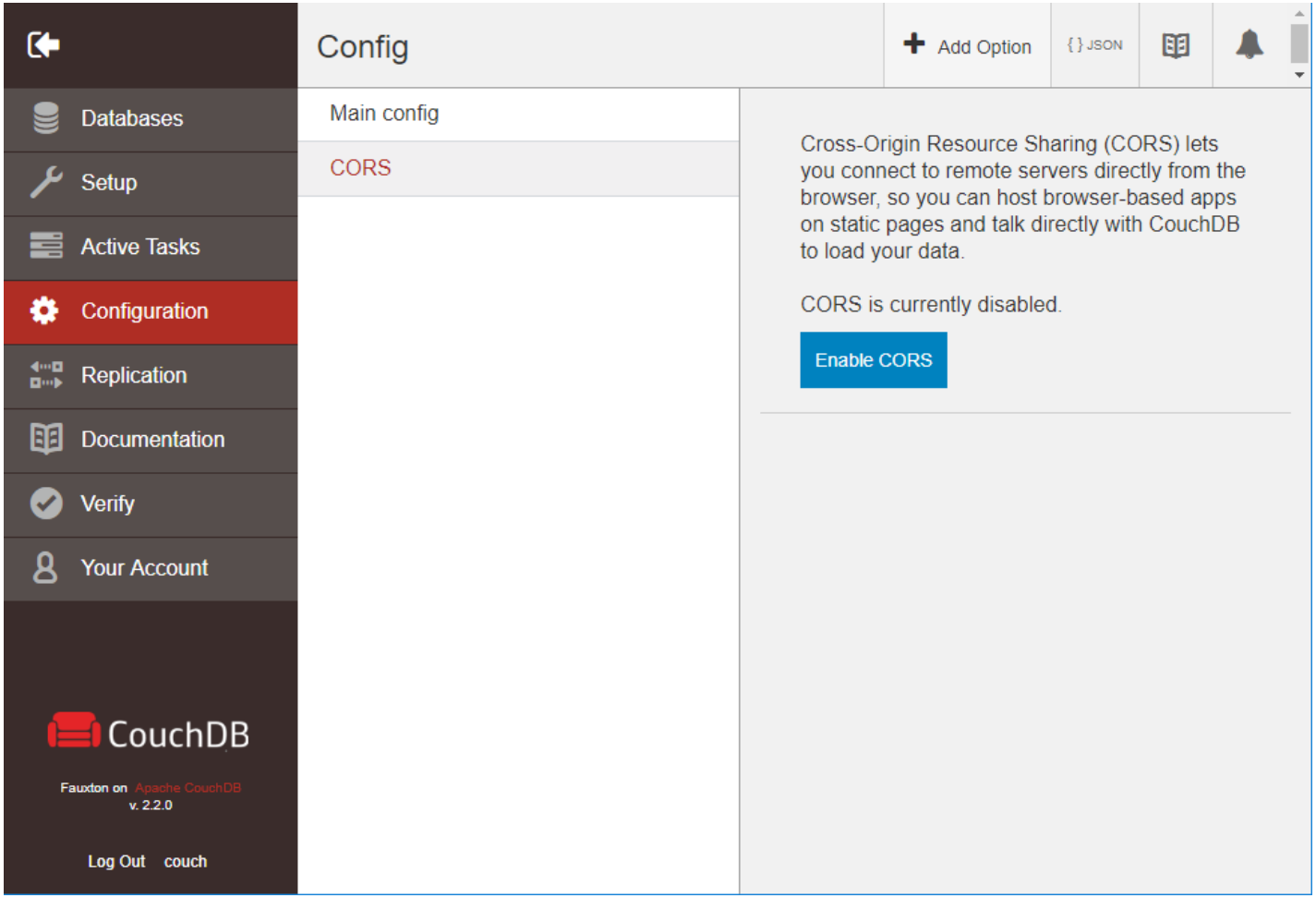
Fauxton on Apache CouchDB
v. 2.2.0

Log Out couch

Showing 1–3 of 3 databases.

« 1 »

Enable CORS


Config
Add Option
{ } JSON
Databases
Setup
Active Tasks
Configuration
Replication
Documentation
Verify
Your Account
Main config
CORS
Cross-Origin Resource Sharing (CORS) lets you connect to remote servers directly from the browser, so you can host browser-based apps on static pages and talk directly with CouchDB to load your data.
CORS is currently disabled.
Enable CORS
CouchDB
Fauxton on Apache CouchDB v. 2.2.0
Log Out
couch

Set CORS to All domains

CORS settings updated.

Databases
Setup
Active Tasks
Configuration
Replication
Documentation
Verify
Your Account

CouchDB

Fauxton on Apache CouchDB v. 2.2.0

Log Out couch

Main config

CORS

Cross-Origin Resource Sharing (CORS) lets you connect to remote servers directly from the browser, so you can host browser-based apps on static pages and talk directly with CouchDB to load your data.

CORS is currently enabled.

Disable CORS

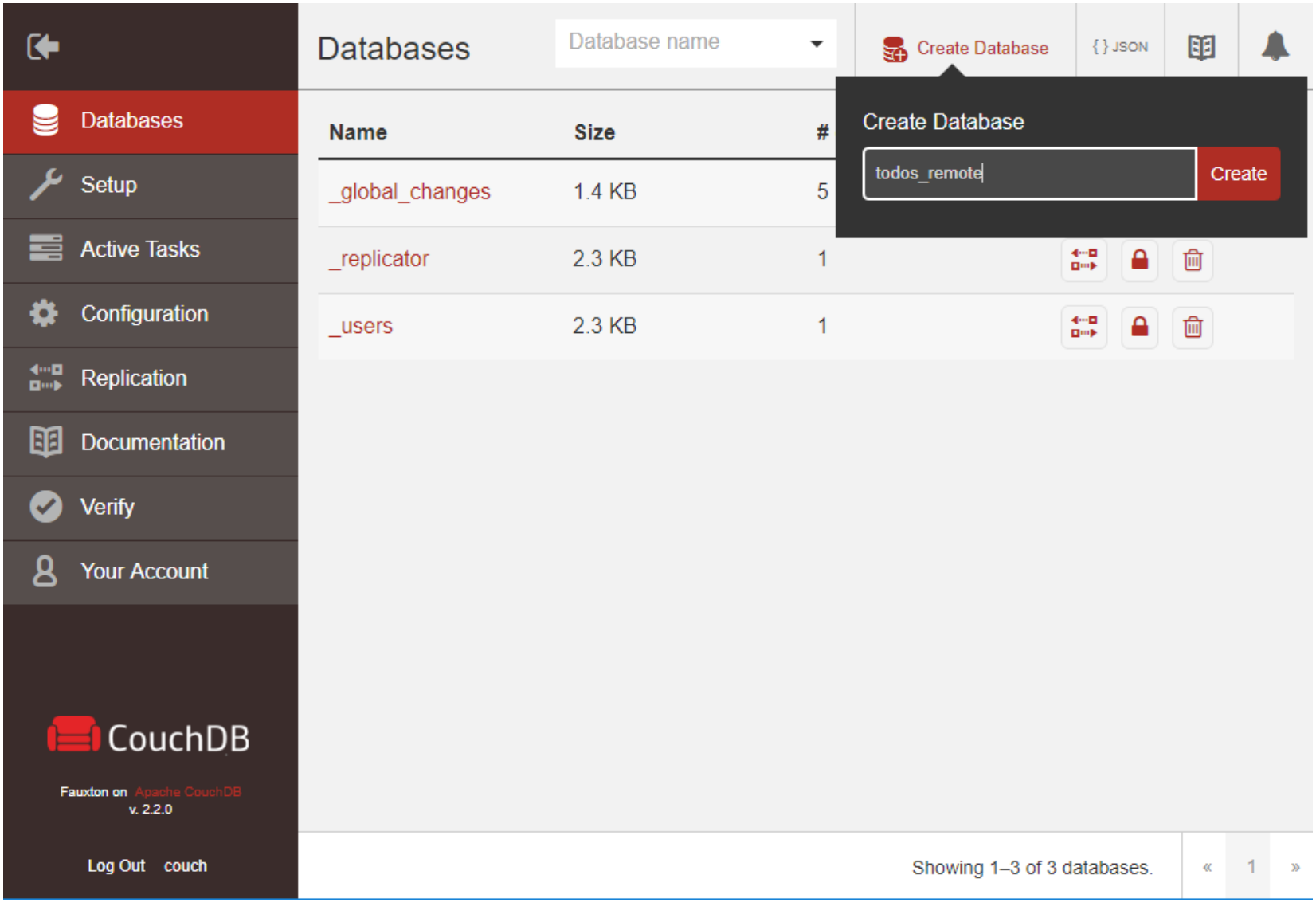
Origin Domains

Select this option

Databases will accept requests from these domains:

- (•) All domains (*)
- ( ) Restrict to specific domains

Create database



Futon

localhost:5984/_utils/#database/todos_remote/_all_docs

Databases
Setup
Active Tasks
Configuration
Replication
Documentation
Verify
Your Account

CouchDB
Fauxton on Apache CouchDB v. 2.3.1

Log Out couch

todos_remote

All Documents
Run A Query with Mango
Permissions
Changes
Design Documents

Document ID | Options | { } JSON

Table | Metadata | {} JSON | Create Document

id "2019-12-12T11:33:00.271Z"


```
{
 "id": "2019-12-12T11:33:00.271Z",
 "key": "2019-12-12T11:33:00.271Z",
 "value": {
  "rev": "1-9d35eac22800b3db9b05cda24000c8d1"
 }
}
```


id "2019-12-12T11:33:01.519Z"


```
{
 "id": "2019-12-12T11:33:01.519Z",
 "key": "2019-12-12T11:33:01.519Z",
 "value": {
  "rev": "2-e98078a03449df6001cfaa2c91fc05cb"
 }
}
```


Showing document 1 - 2. Documents per page: 20

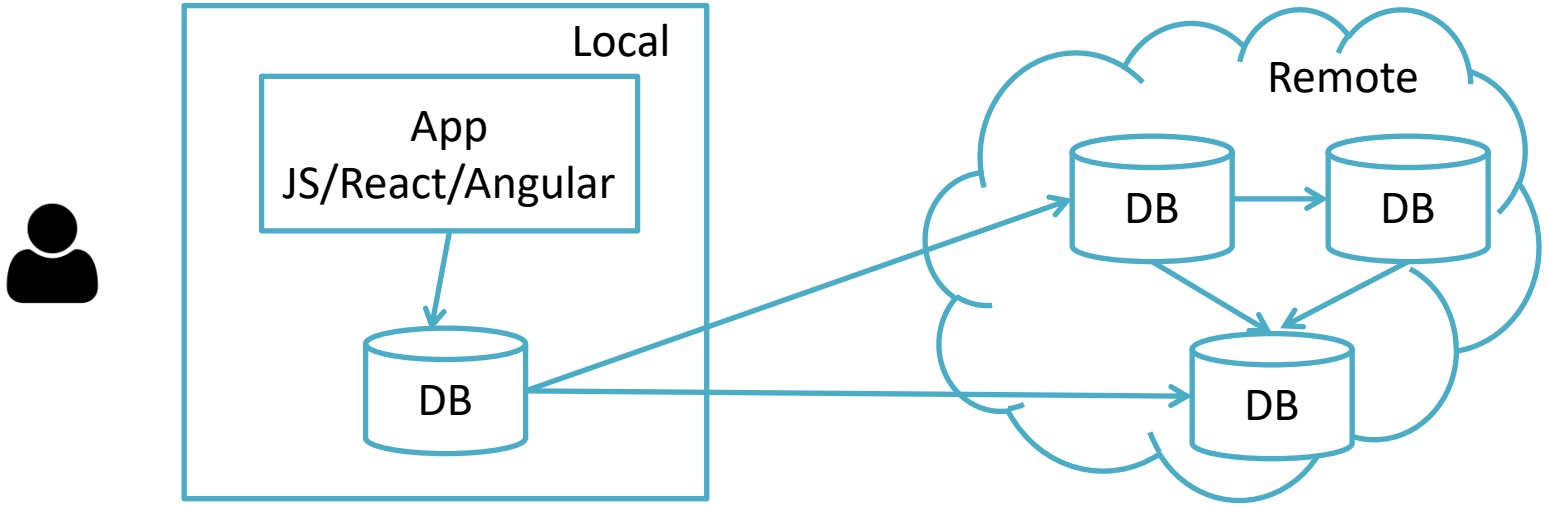
PouchDB

PouchDB – Definition and basics

- CouchDB JavaScript implementation
- **Database in the browser!!**
- Works in any browser, using **IndexedDB** where available and **WebSQL** in others
- **Offline-first**: allows storing and accessing data for offline applications
- Can be **synchronized** with CouchDB or another compatible database when coming online, it uses CouchDB replication protocol
- Also works with Node.js (using LevelDB), so can be used in the **backend** and we will have the same interface (API) in all components

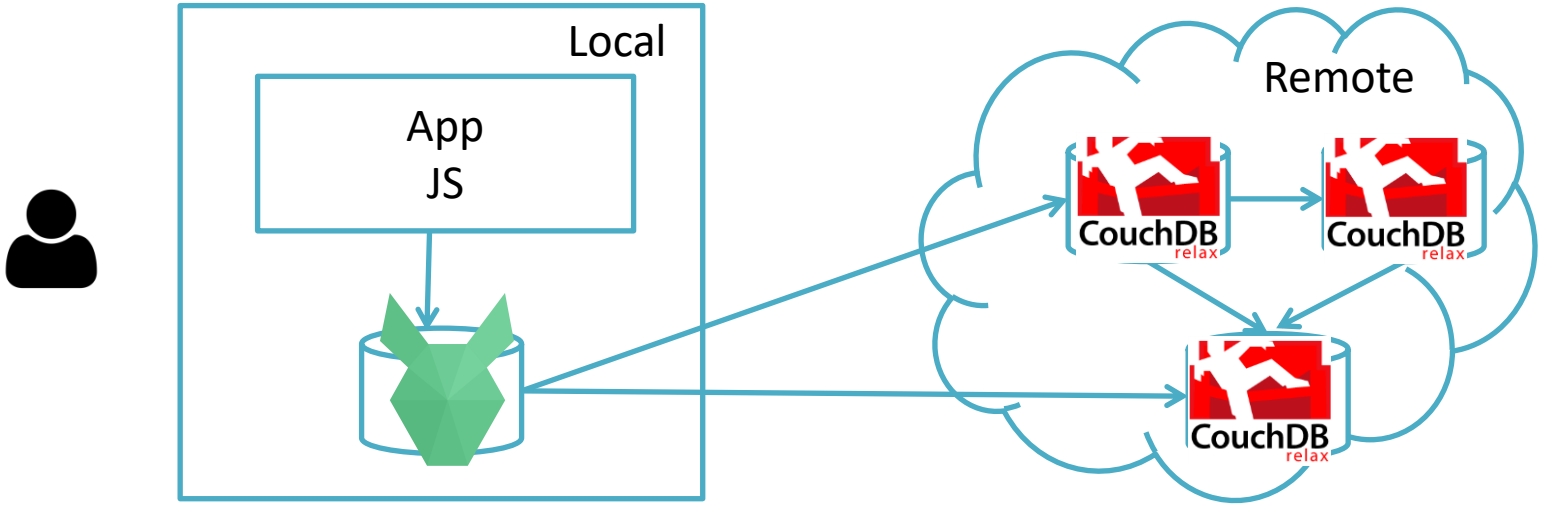
How it works

- Offline:
  - Data is stored locally using IndexedDB and WebSQL in the browser
- Online:
  - Synchronizes with CouchDB or any other compatible DB



How it works

- Offline:
    - Data is stored locally using WebSQL and IndexedDB in the browser
- Online:
    - Synchronizes with CouchDB or any other compatible DB



PouchDB Browser Support

Adapter	**IE (10+)**	**Firefox**	**Chrome < 43, Android**	**Chrome >= 43**	**Safari < 7.1, iOS < 8**	**Safari >= 7.1, iOS >= 8**
IndexedDB	Blob	Blob	Base-64	Blob		
WebSQL			Blob	Blob	UTF-16 Blob	Blob

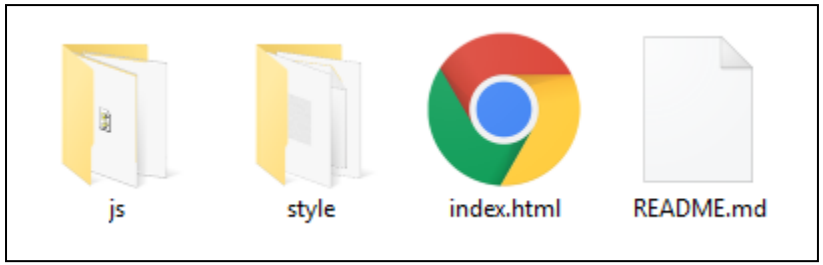
Installation

- We only have to include the PouchDB script in our html file.
- either from a CDN

```
<script src="//cdn.jsdelivr.net/pouchdb/7.1.1/pouchdb.min.js"></script>
<script src="js/app.js"></script>
```

- or locally

```
<script src="js/lib/pouchdb.min.js"></script>
<script src="js/base.js"></script>
<script src="js/app.js"></script>
```



Create database

- You don't need to create a schema for the database. After giving it a name, you can immediately start writing or reading objects
- We will use this db object for all operations on the database

```
var db = new PouchDB('todos');
```

Create database with remoteCouch

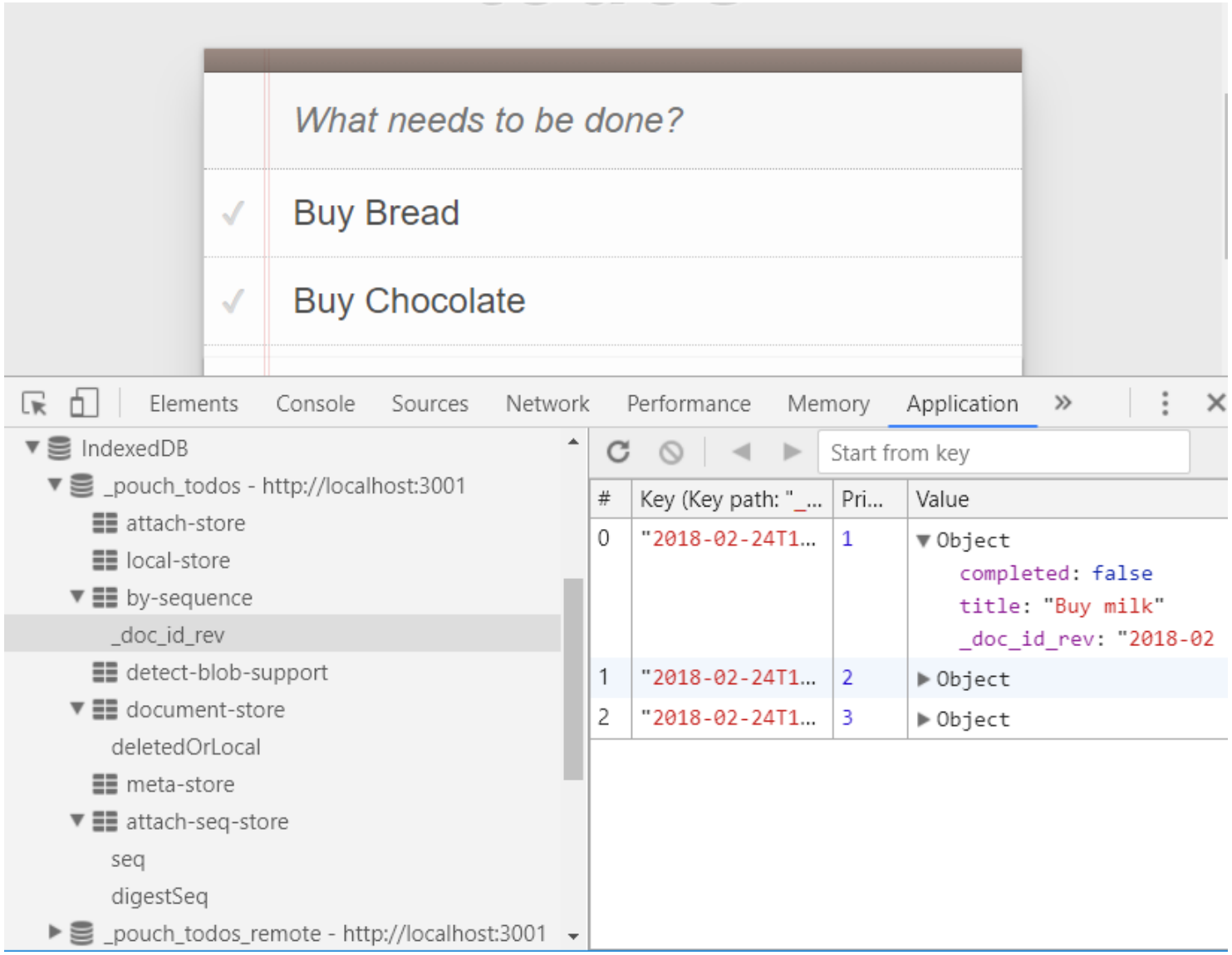
- If you set up a remote Couch Database, all the data will be automatically synchronized (using a sync function that should be added)

```
// Replace with remote instance, this just replicates to another local instance.
var remoteCouch = 'todos_remote';
//var remoteCouch = 'http://localhost:5984/todos_remote';

// Initialise a sync with the remote server
function sync() {
  syncDom.setAttribute('data-sync-state', 'syncing');
  var opts = {live: true};
  db.replicate.to(remoteCouch, opts, syncError);
  db.replicate.from(remoteCouch, opts, syncError);
}

// There was some form or error syncing
function syncError() {
  syncDom.setAttribute('data-sync-state', 'error');
}
```

Create database



Working with documents – get (one item)

```
db.get('mydocId').then(function (doc) {
  console.log(doc);
}).catch(function (err) {
  console.log(err);
});
```

```
try{
  let doc = await db.get('mydocId');
  console.log(doc);
} catch (err){
  console.log(err);
}
```

- Result:

```
{
  "_id": "mydocId",
  "_rev": "1-84e1c6bf31d49680e202039accacea01"
  "title": "React course",
  "other_field": "This is other field"
}
```

Working with documents – get (all items)

```
try {
  var doc = await db.allDocs({include_docs: true, descending: true});
  var todos = doc.rows.map(function(item, index){
    return item.doc;
  });
  console.log(todos);
} catch (err) {
  console.log(err);
}
```

- **Result**:

```
▼{total_rows: 2, offset: 0, rows: Array(2)}
  offset: 0
  ▼rows: Array(2)
    ▼0:
      ▶doc: {title: "Angular Course", completed: false, _id: "2018-01-30T00:43:08.689Z", _rev: "1-91dcfc05c84e97326fbe9b4093
       id: "2018-01-30T00:43:08.689Z"
       key: "2018-01-30T00:43:08.689Z"
      ▶value: {rev: "1-91dcfc05c84e97326fbe9b4093ba27f9"}
      ▶__proto__: Object
    ▼1:
      ▶doc: {title: "React Course", completed: false, _id: "2018-01-30T00:43:05.019Z", _rev: "1-853038a31a879c4473934bf52170
       id: "2018-01-30T00:43:05.019Z"
       key: "2018-01-30T00:43:05.019Z"
      ▶value: {rev: "1-853038a31a879c4473934bf521702629"}
      ▶__proto__: Object
     length: 2
    ▶__proto__: Array(0)
  total_rows: 2
▶__proto__: Object
```

Working with documents – create (put)

```
var todo = {
  _id: "mydocId",
  title: "Buy Bread",
  completed: false
};
try {
  let result = await db.put(todo);
  console.log('Successfully posted a todo!' + result.id + " " + result.rev);
} catch (err) {
  console.log(err);
}
```

Understanding revisions (_rev)

- The field "**_rev**" is the *revision marker*. It is a randomly-generated ID that changes whenever a document is created or updated.
- Unlike most other databases, whenever you update a document in PouchDB or CouchDB, you must present the *entire document* along with its current *revision marker*.
- If you fail to include the correct _rev, you will get the following sad error:

```
{
  "status": 409,
  "name": "conflict",
  "message": "Document update conflict"
}
```

Working with documents – update (put)

- For an update we need to specify an existing _id and the corresponding _rev
- The best option is to get the document directly first and then updating

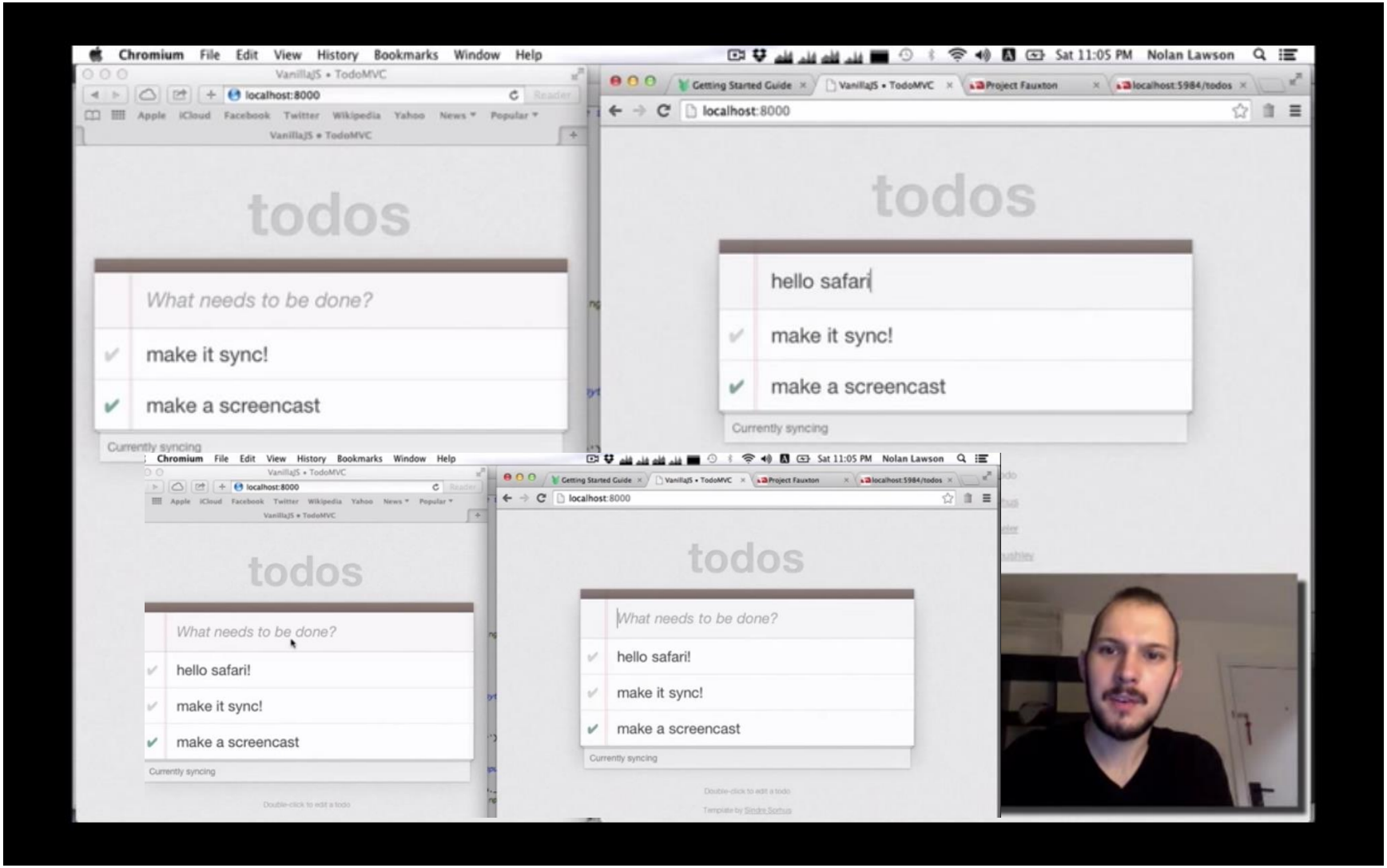
```
try{
  let doc = await db.get('mydocId');
  doc.title = "Buy apples";
  await db.put(doc);
} catch (err){
  console.log(err);
}
```

Working with documents – delete (remove)

- For a remove we need to specify an existing _id and the corresponding _rev
- The best option is to get the document directly first and then updating

```
try{
  let doc = await db.get('mydocId');
  await db.remove(doc);
  console.log("Successfully removed doc");
} catch (err){
  console.log(err);
}
```

Synched!!


Chromium File Edit View History Bookmarks Window Help
Sat 11:05 PM Nolan Lawson
localhost:8000
todos
What needs to be done?
make it sync!
make a screencast
Currently syncing
hello safari
hello safari!

Progressive Web Apps - PWA

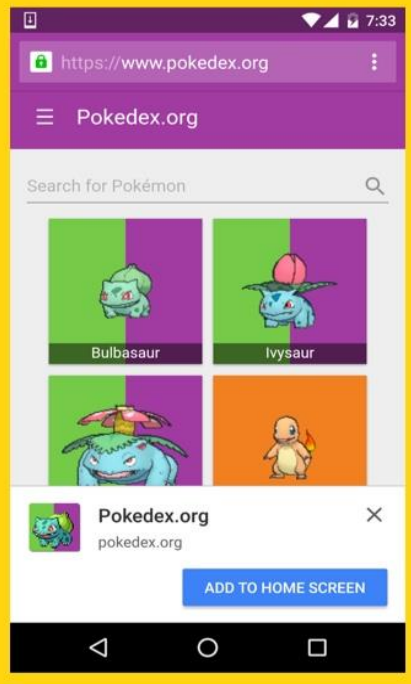
- "A Progressive Web App uses modern web capabilities to deliver an app-like user experience"

- Use web APIs to be as similar as possible to native apps
- Fast load (53% of users left the webpage if it takes more than 3 seconds to load) and are responsive

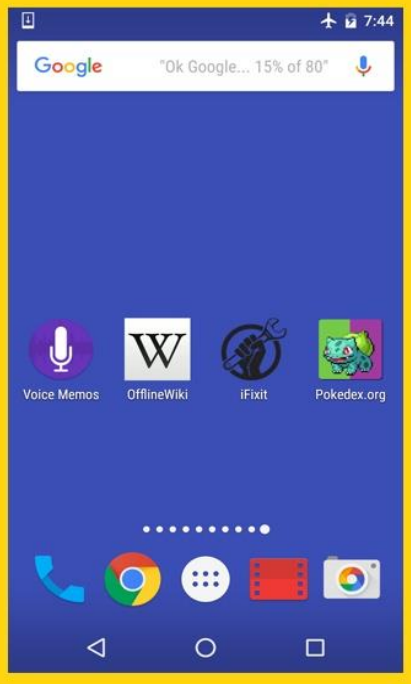
- Service workers:
  - Service Worker is a worker script that works behind the scenes, independent of your app, and runs in response to events like network requests, push notifications, connectivity changes, and more, como un "proxy"
  - They power offline functionality, push notifications, background content updating, content caching, and a whole lot more.
- App manifest to be installed in the home screen of the device
- Offline, cache, notifications, …

- Checklist for PWA
- https://developers.google.com/web/progressive-web-apps/checklist?hl=es

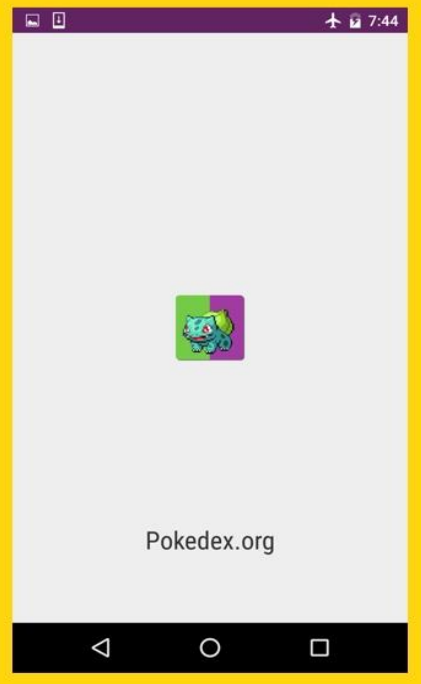
Progressive Web Apps



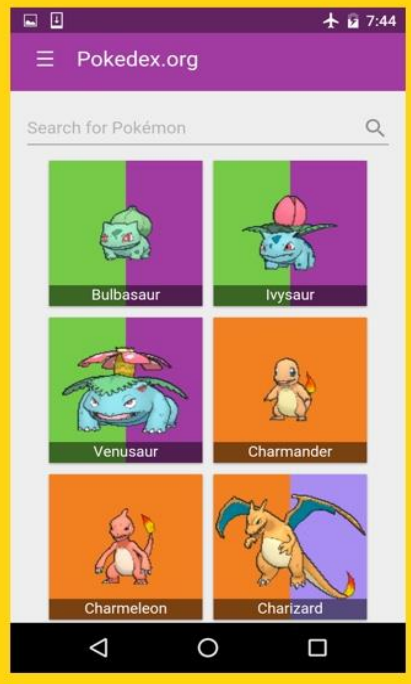
Web App install banner for engagement



Launch from user's home screen



Splash screen (Chrome for Android 47+)



Works offline with Service Worker

Source: https://addyosmani.com/blog/getting-started-with-progressive-web-apps/

PWA – another example

- https://flights.airberlin.com/en-DE/progressive-web-app

- Airberlin. In this website they explain why and how they built it and the use case results

- They used web components (polymer), a wrapper for indexedDB and websql (localForage), service workers, manifest and metadata, HistoryAPI, custom events and vanillaJS (i.e. pure JS, with no libraries except moments.js for dates and times)