

Resilient Apps with Angular 2

@giorgionatili

@giorgionatili

- Engineering Lead (McGraw-Hill Education)
- Community fellow since 2004
- Open source fanatic
- The mobile dude
- Founder of Mobile Tea



#MOBILETEA

www.meetup.com/pro/mobiletea

Supporters



<http://careers.mheducation.com>



<https://dev.twitter.com>

Today's Agenda

- An Overview of Angular 2
- Handling the Offline Status
- ServiceWorker API
- ServiceWorker and Angular 2
- Redux and Angular 2

Angular 2

Tech Stack

- TypeScript
- RxJS
- JSPM or WebPack
- NPM or Grunt and Gulp (seriously?!?)
- Karma + (Jasmine or Mocha) + Chai

TypeScript

- A super set of JavaScript developed by Microsoft
- Implements classes, interfaces, inheritance, strict data typing, private properties and methods, etc.
- Angular Team choice for the new version of Angular (Microsoft and Google are now buddies!)

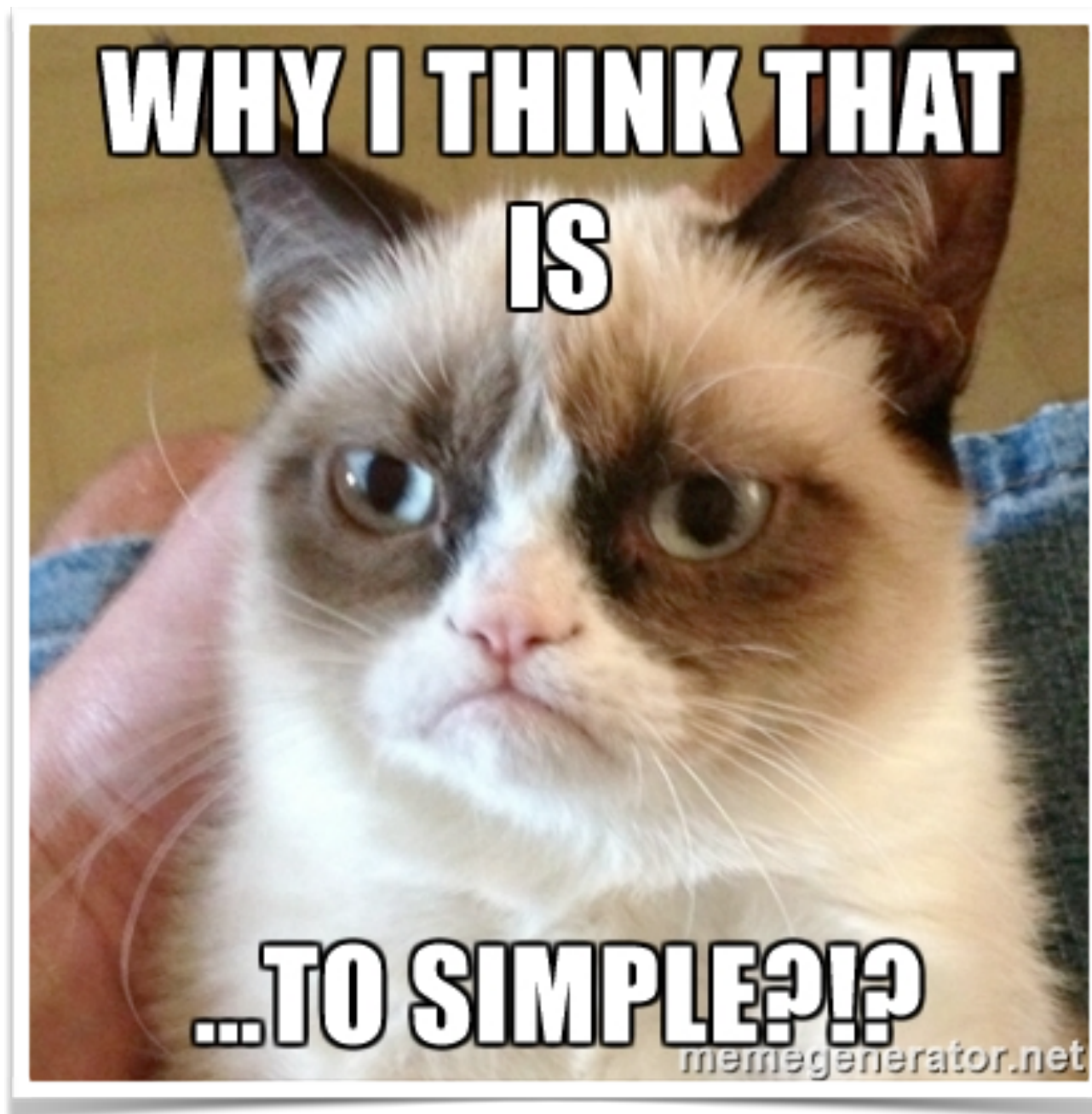
RxJS

<http://reactivex.io/rxjs/>

- A reactive streams library that allows you to work with asynchronous data streams
- Combines *Observables* and *Operators* so we can subscribe to streams and react to changes using composable operations

RxJS, hello world!

```
let counter = 0;  
rx.Observable  
  .interval(1000)  
  .take(3)  
  .safeApply($scope, function(x) {  
    counter = x;  
  })  
  .subscribe(); // shows 0, 1, 2
```



Components

```
import {Component} from '@angular/core';

@Component({
  selector: 'hello',
  template: '<p>Hello, {{name}}</p>'
})
export class Hello {
  name: string;
  constructor() {
    this.name = 'World';
  }
}
```

Components

- selector is the element property that we use to tell Angular to create and insert an instance of this component
- template is a form of HTML that tells Angular how to render this component

@Input and @Output

- The @input attribute defines a set of parameters that can be passed down from the component's parent
- The @output attribute is used to send data out of components

Constructor vs `ngOnInit()`

- The Constructor ensures proper initialization of fields in the class and its subclasses
- Angular DI analyzes the constructor parameters and when it creates a new instance it tries to find `providers` that match the types of the constructor parameters
- `ngOnInit` is a life cycle hook to indicate that that Angular is done creating the component

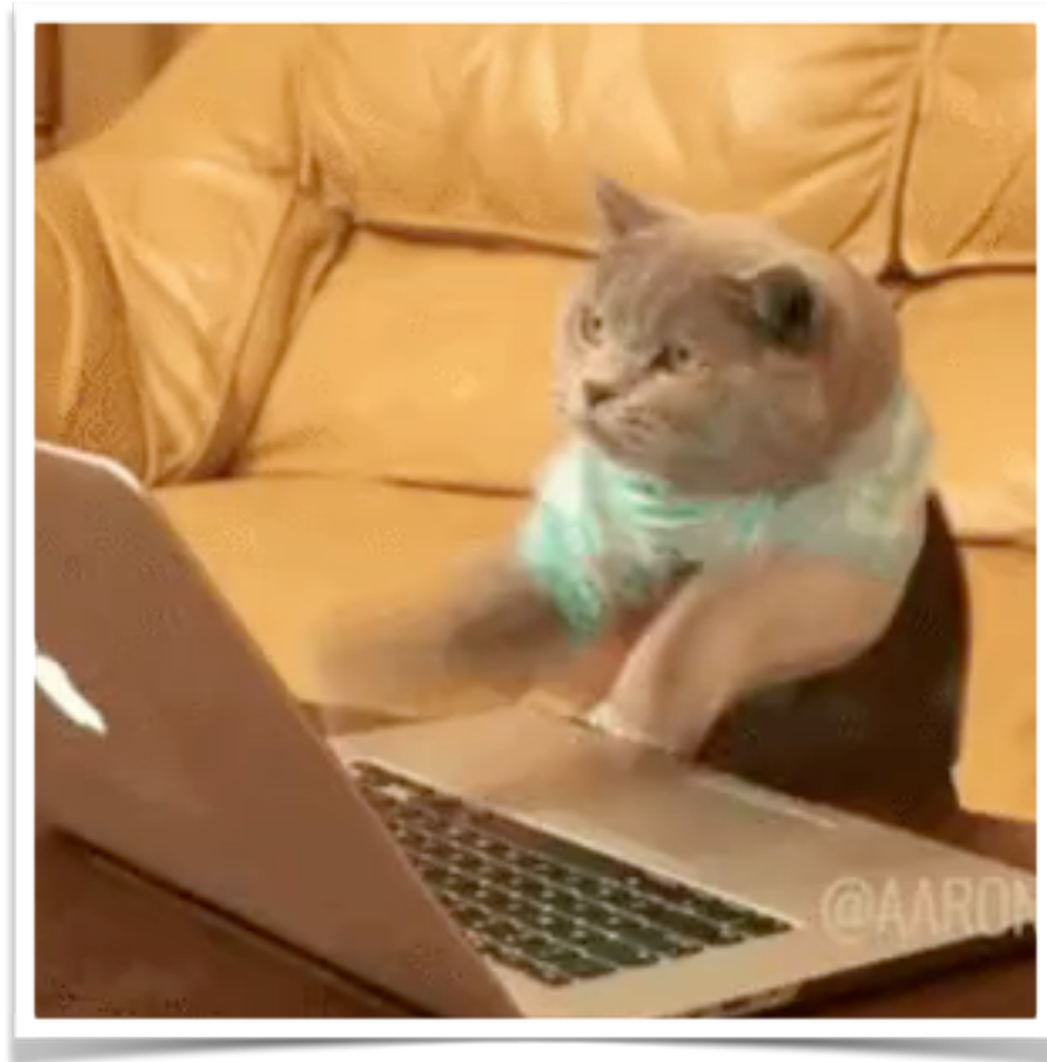
Components LifeCycle

- ngOnChanges - called when an input binding value changes
- ngOnInit - after the first ngOnChanges
- ngDoCheck - after every run of the change detection mechanism
- ngAfterContentInit - after component content initialized
- ngAfterContentChecked - after every check of component content change
- ngAfterViewInit - after component's view(s) are initialized
- ngAfterViewChecked - after every check of a component's view(s)
- ngOnDestroy - just before the component is destroyed

Template and Bindings

- Interpolation (`{{ }}`)
`<p>My current hero is
{{currentHero.firstName}}</p>`
- The pipe operator (`|`)
`<div>Title through uppercase pipe: {{title |
uppercase}}</div>`
- Event binding (`(event)`)
`<button (click)="onSave () ">Save</button>`
- Two-way binding with `NgModel` (`[()]`)
`<input [(ngModel)]="currentHero.firstName">`

Online Tutorials

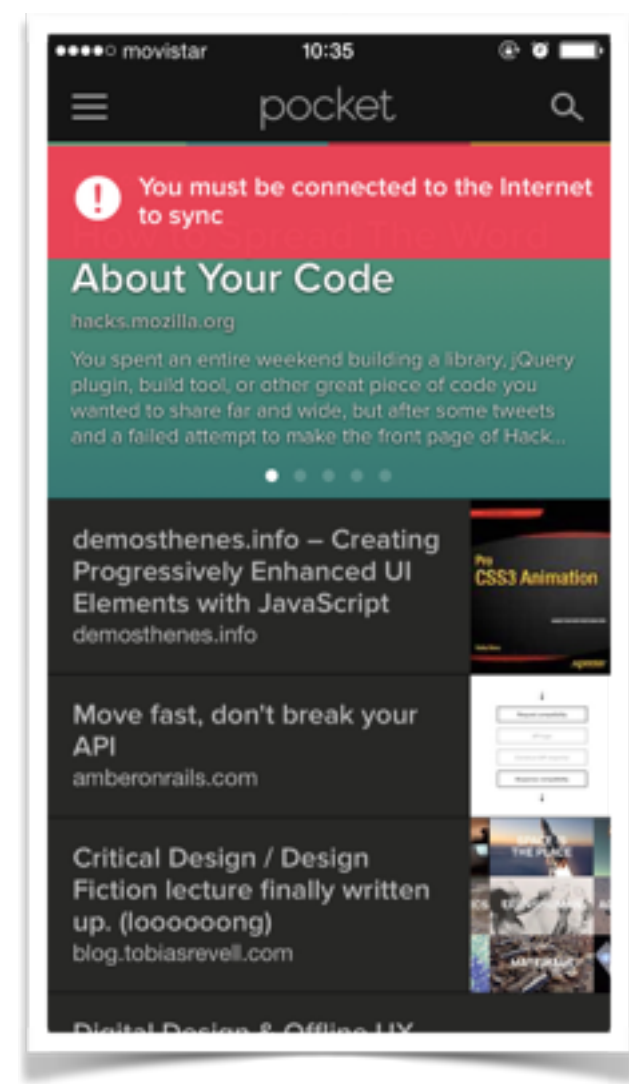
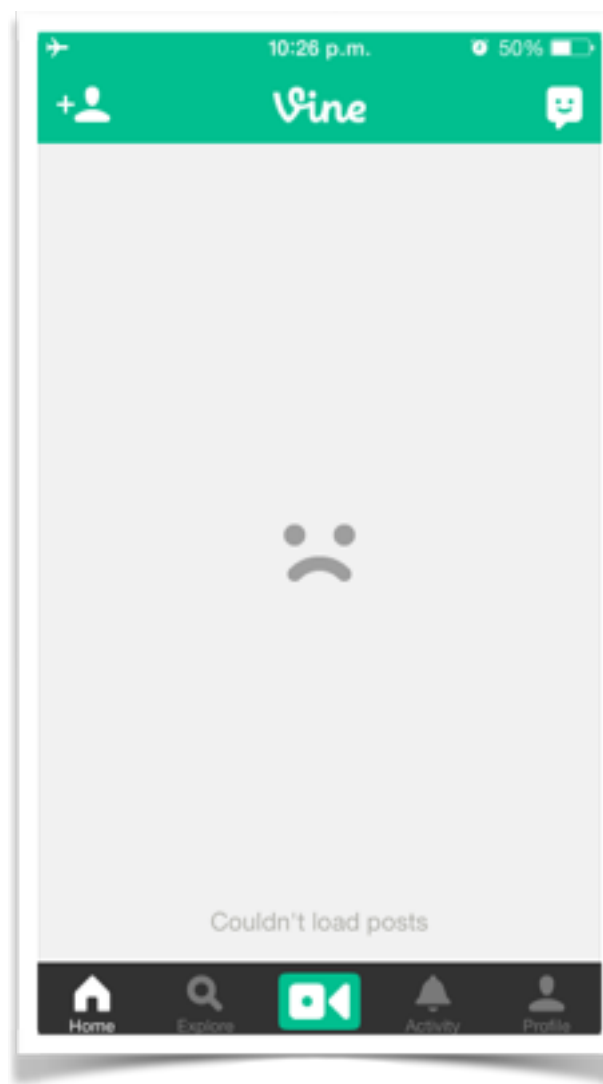


Handling Offline Status

Resilient Apps

- Treat offline as the norm
- All request must have a fallback
- Use available API's to detect device & network capabilities
- Adapt application logic to match the device & network capabilities

Offline Statuses

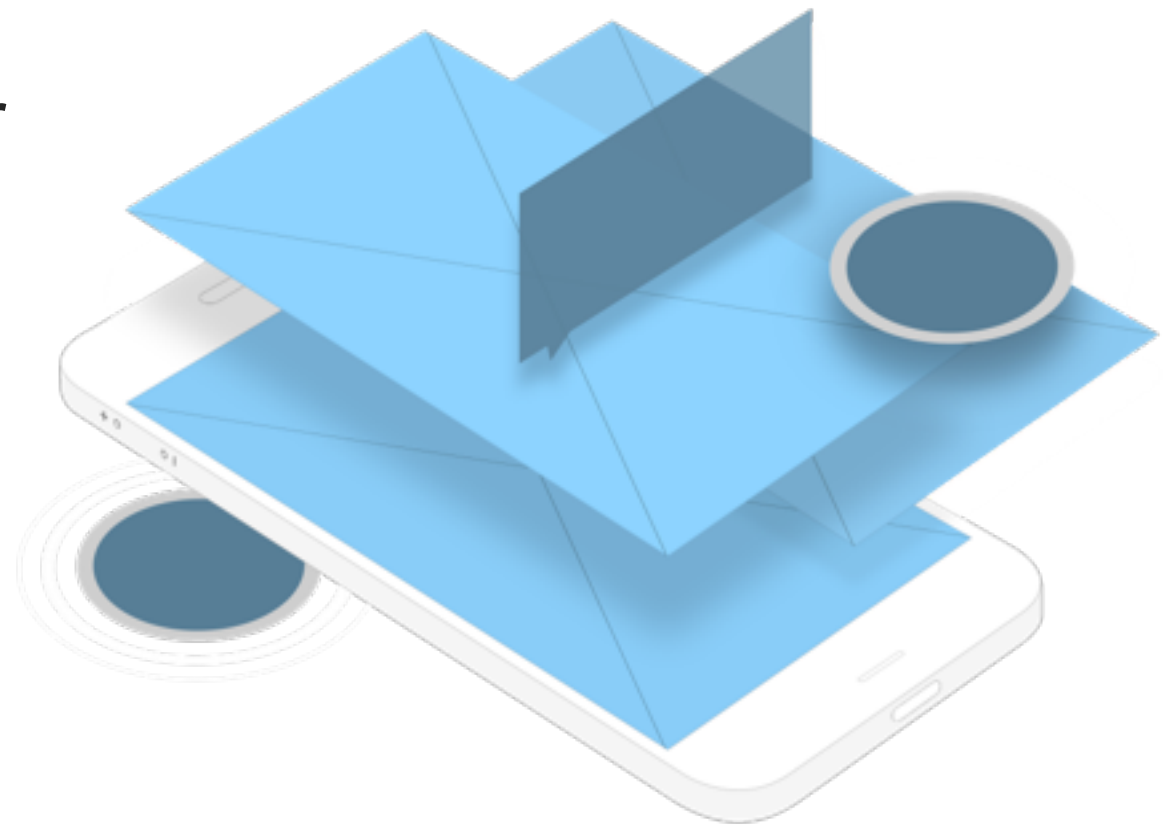


Offline is a New Opportunity

- Designing apps and their interfaces for intermittent connectivity leads to an abundance of new scenarios and problems
- Solving all of them means that your app should preempting users' needs

Identify App Layers and Assets

- Communication Layer
- Network Layer
- Bundles
- User Interface



Communication Layer

Be sure to decouple components as much as possible; otherwise caching the dependencies will prove difficult when implementing any strategy.

Network Layer

Use the fetch API and avoid implementing a call-back pyramid of doom; take advantage of the Promises and of their sequential syntax.

Bundles

Avoid a single bundle; breaking down the app in more than a single monolithic bundle will be the key for a layered caching strategy.

User Interface

Optimizing the UI for different screens means also feeding the app with appropriately compressed images, this consideration is the key to avoid issues with caches size.

Do You Want More Details?



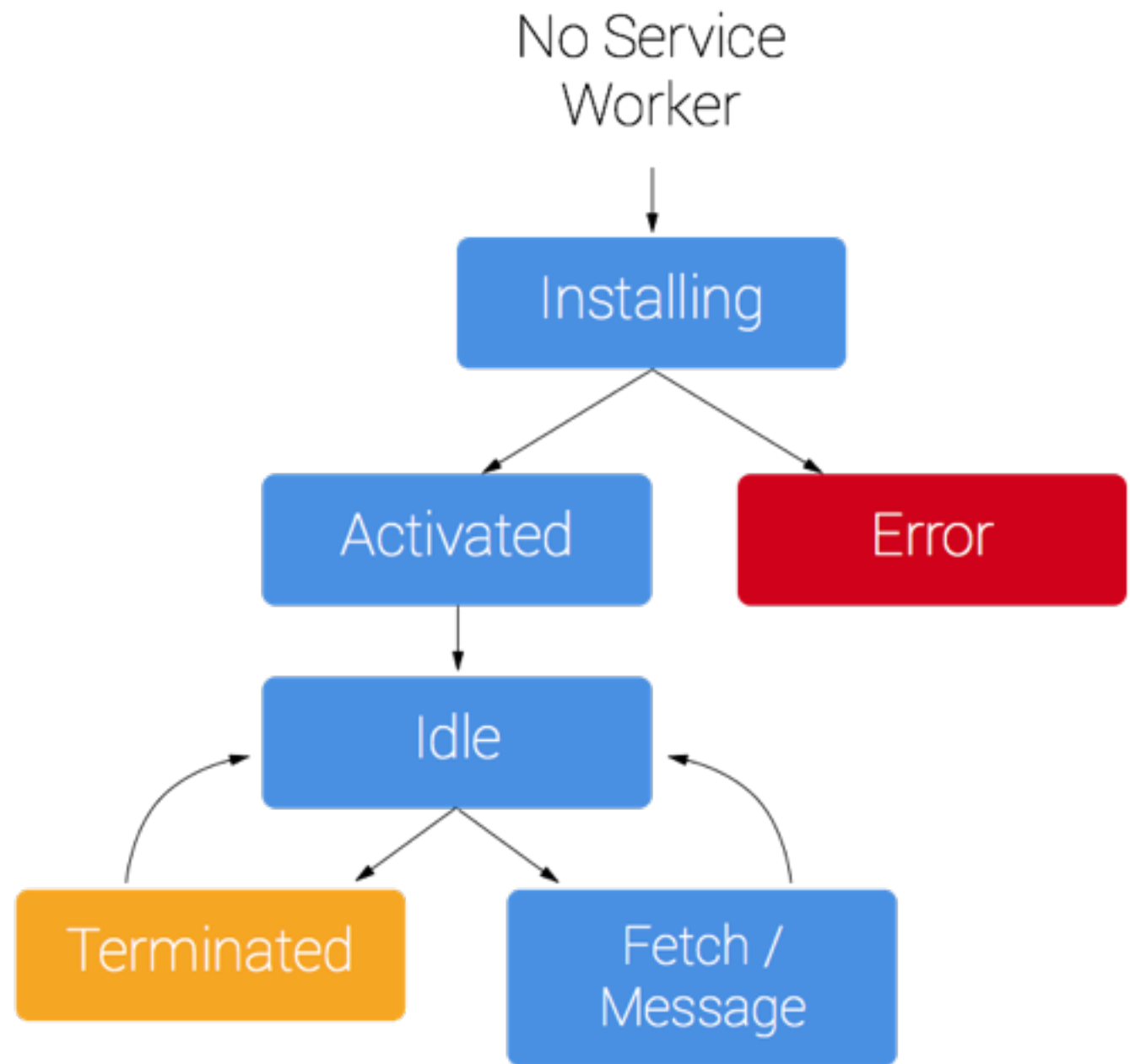
Service Worker API

In a Nutshell

- Service workers are event-driven workers that act as proxy servers that sit between web applications, the browser, and the network (when available)
- The ServiceWorker API allows us to make sites work offline through intercepting network requests and *programmatically* telling the browser what to do with these requests

Lifecycle

- On install
- On activate
- On push message
- On background-sync
- On terminate



Registering a ServiceWorker

```
let nav = <any> navigator;
if ('serviceWorker' in nav) {

  nav.serviceWorker.register(' ./service-worker.js')
    .then(function(reg) {

      console.log('yey!', <any> reg);
    }).catch(function(err) {

      console.log('boo!', <any> err);
    });
}
```


ServiceWorker Installation

```
self.addEventListener('install', function(event) {  
  event.waitUntil(  
    caches.open('v1').then(function(cache) {  
      return cache.addAll([  
        '/sw-test/',  
        '/sw-test/index.html',  
        '/sw-test/style.css',  
        '/sw-test/app.js'  
      ]);  
    })  
  );  
});
```

Not Blocking Installation

```
self.addEventListener('install', function(event) {  
    event.waitUntil(self.skipWaiting());  
});
```

Configuring Caches

```
// Import version number
importScripts('service-worker-cache-version.js');

// Caches name definition
let staticCacheName = `cache-static-v${cacheVersion}`;
let imageCacheName   = `cache-img-v${cacheVersion}`;
let apiCacheName      = `cache-api-v${cacheVersion}`;
```

Handling Caches Size

```
event.waitUntil(  
  caches.keys().then(function(keyList) {  
  
    return Promise.all(keyList.map(function(key) {  
  
      if (cacheWhitelist.indexOf(key) === -1) {  
        return caches.delete(key);  
      }  
    }));  
  }));  
);
```

Fetch Requests

```
this.addEventListener('fetch', function(event) {  
  
    // Magic happens here  
    event.respondWith(  
  
        // Optional cache fallback implementation  
        caches.match(event.request)  
    );  
});
```

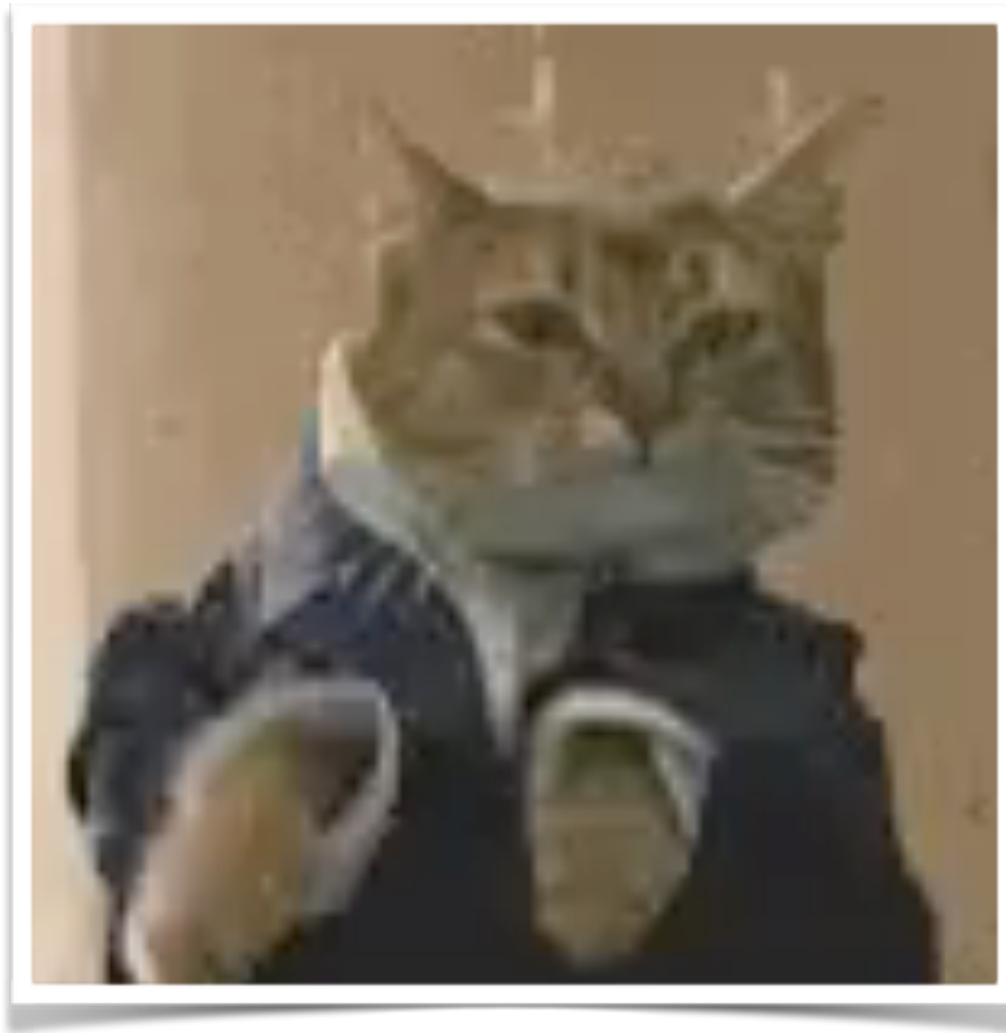
No Extra Testing Effort

- A ServiceWorker returns eventually cached results
- Emulating offline the tests previously written should fulfill without any extra effort

Security

- During development you'll be able to use ServiceWorker through localhost
- To deploy it on a site, you'll need to set up HTTPS on your server

There is More?



oh yeah!

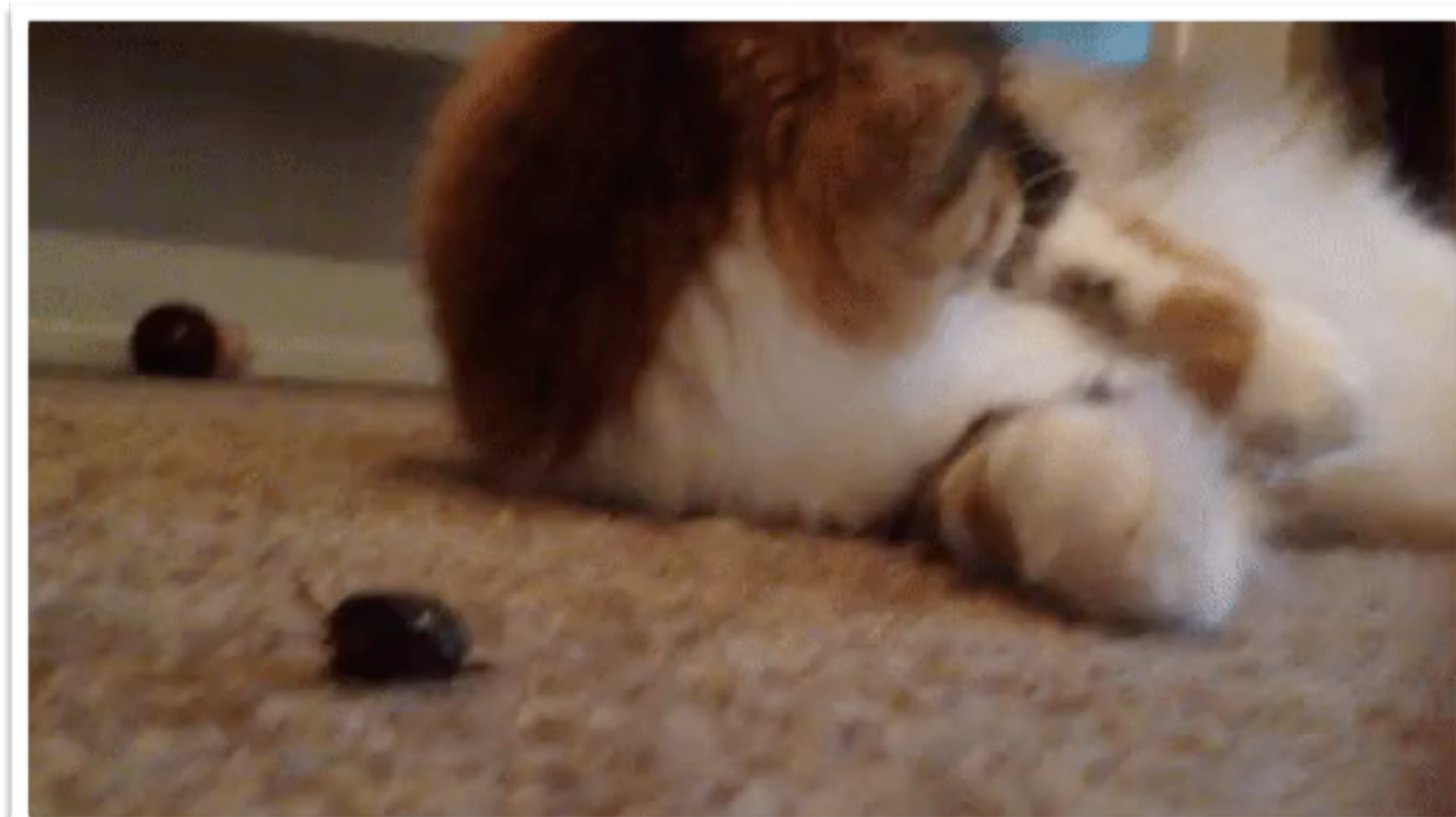
Push Notifications

```
self.addEventListener('push', function(event) {  
    if (event.data.text() == 'new-tweet') {  
        // Handle caches and prepare the content for the view  
    });  
  
self.addEventListener('notificationclick', function(event) {  
    if (event.notification.tag == 'new-tweet') {  
        // Assume that all of the resources needed to render  
        // the 'inbox' have previously been cached  
        new TweetView('inbox');  
    }  
});
```

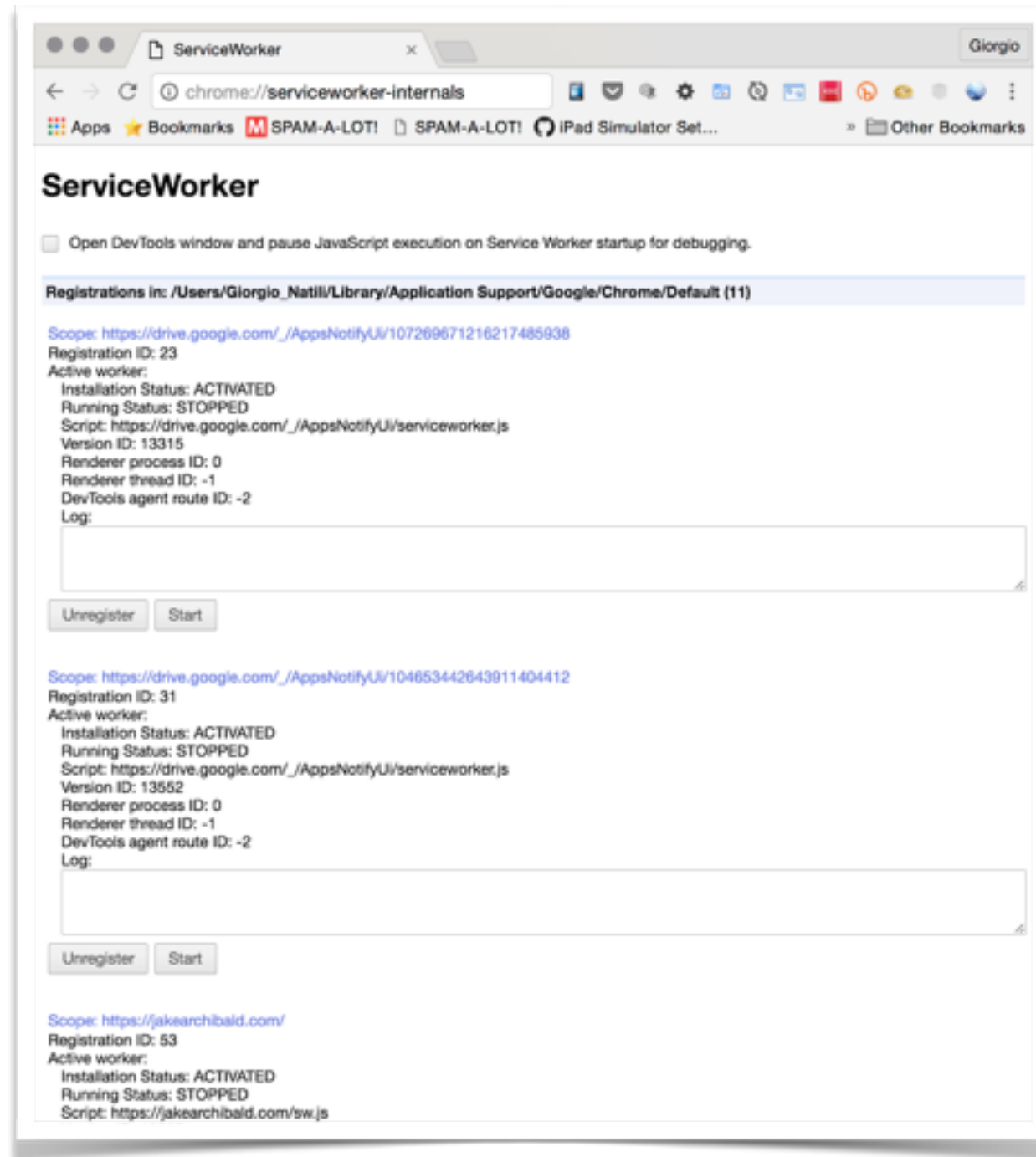
Background Sync

```
self.addEventListener('sync', function(event) {  
    if (event.id == 'update-leaderboard') {  
        // Or whatever promise that sync the data  
        return openBgSyncDB();  
    }  
});
```

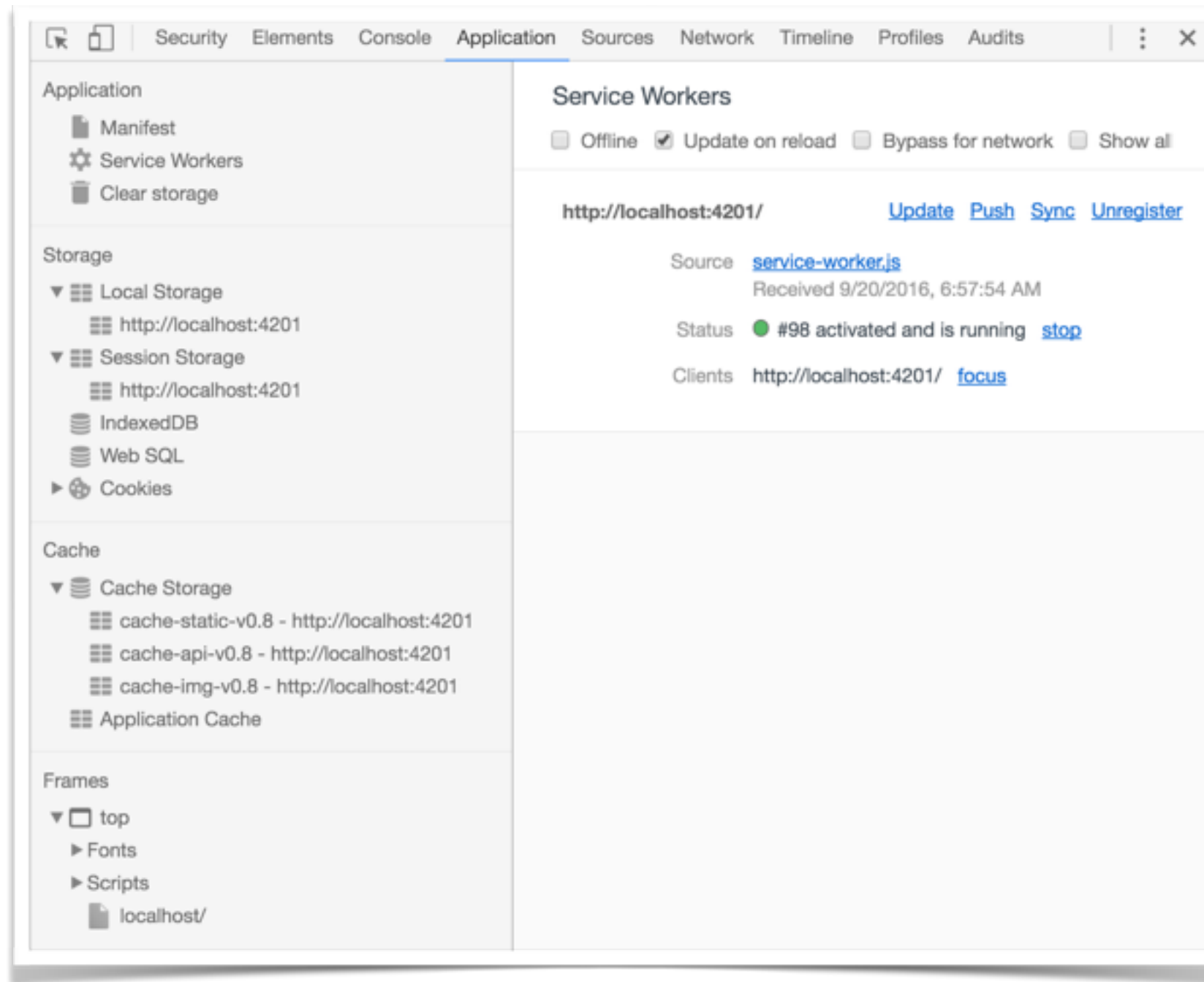
How to Debug?



Chrome Internals



Developer Tools



Demo

ServiceWorker with Angular 2

Installing the DataType

```
$ npm install --save @types/service_worker_api
```


Angular 2 Services

- Angular comes with its own HTTP library which we can use to call out to external APIs
- In Javascript, there are generally three approaches to dealing with async code:
 - Callbacks
 - Promises
 - Observables

Promises vs Observables

- Observables are cancelable
- Observables can be retried (`retry` and `retryWhen`)

Dos and Don'ts with SW API

- Don't debug your service in a standard browser window
- Never create a monolithic service
- Keep your service in your app root
- Handle scopes properly

That Sounds...



... powerful!

Redux & Angular 2

Redux in a Nutshell

- Redux is a predictable state container for JavaScript applications
- Instead of mutating the state directly, you specify the mutations you want to happen with plain objects called actions
- Then you write a special function called a reducer to decide how every action transforms the entire application's state
- You should always return the state of the app

App State

- Hold the entire app state in a single store, which is also immutable and normally serializable.
- Don't design the state to map the server side API
- Never hold duplicate data in the app state (items in stock or out of stock)

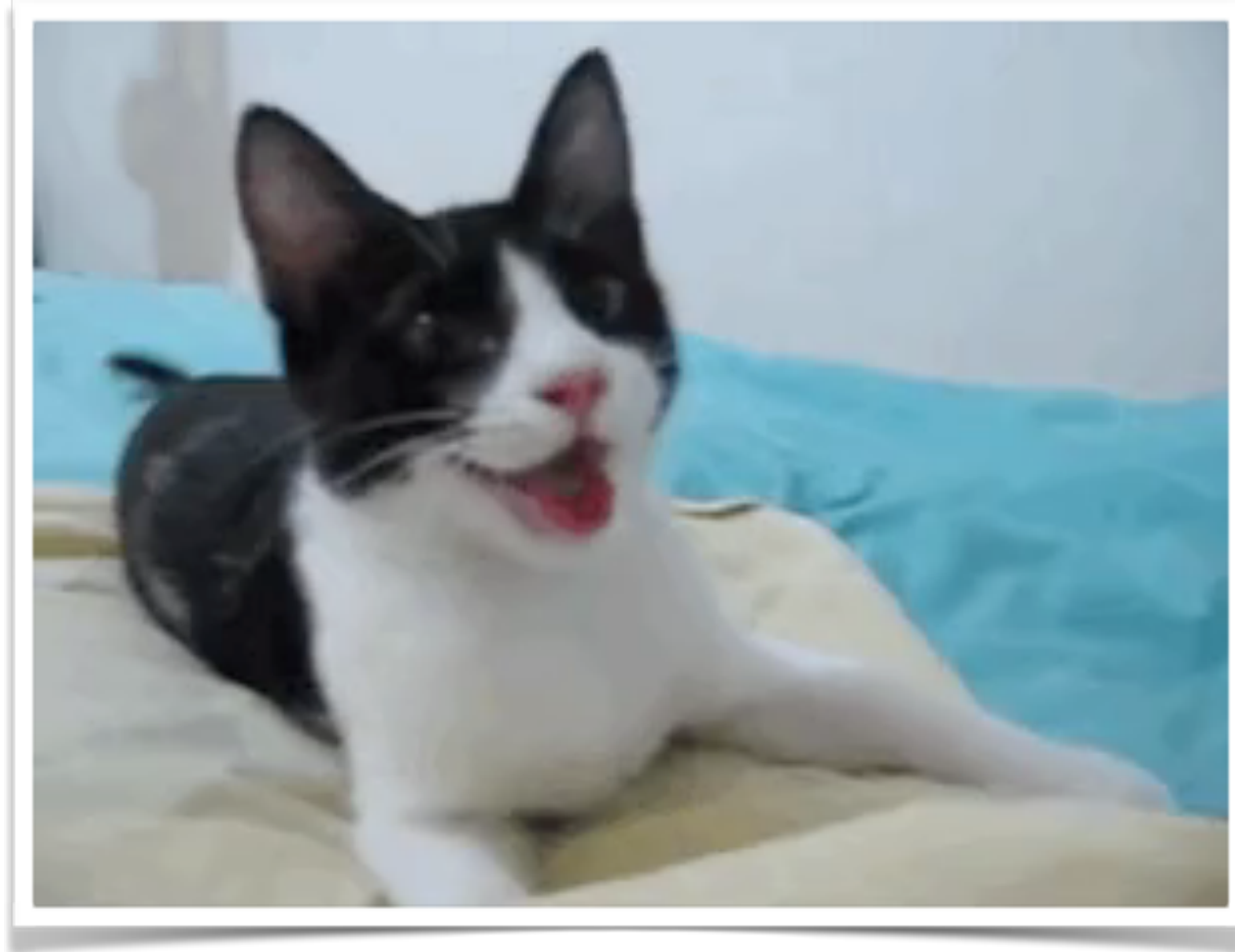
memoize



Local Storage

```
const persistState = require('redux-localstorage');  
  
export const enhancers = [  
  persistState('lastSearch',  
    { key: 'ng2-redux/examples/lastSearch' })  
];
```

Do we need redux for this?



... yes!!!

Redux Actions

“Actions are payloads of information that send data from your application to your store”

```
{ type: 'ADD_TODO', text: 'Use Redux' }  
{ type: 'REMOVE_TODO', id: 42 }  
{ type: 'LOAD_ARTICLE', response: { ... } }
```

Redux Reducers

*“Actions describe the fact that something happened, then a **reducer** handle the event and eventually update the app state”*

Redux Reducers

```
import * as TodoActions from './todoActions';

const initialState = {
  todos: [],
  currentFilter: 'SHOW_ALL'
}

export function rootReducer(state = initialState, action) {
  switch (action.type) {
    case TodoActions.ADD_TODO:
      return {
        todos: state.todos.concat({
          id: action.id,
          text: action.text,
          completed: action.completed
        }),
        currentFilter: state.currentFilter
      };
    case TodoActions.TOGGLE_TODO:
      return {};
    // Continue...
  }
}
```

Redux Stores

“A Redux *store* is the object that brings together actions and reducers offering a centralized way to dispatch actions and access the app state”

```
import { createStore } from 'redux'
import todoApp from './reducers'
import { addTodo } from './actions'

let store = createStore(todoApp)

// Log the initial state
console.log(store.getState())


// Dispatch some actions
store.dispatch(addTodo('Learn about actions'))
```

Redux Data Flow

- You call `store.dispatch(action)`
- The store calls the reducer function you gave it
- The root reducer may combine the output of multiple reducers into a single state tree
- The store saves the complete state tree returned by the root reducer

How it Sounds?

awe·some

[aw-suh m] 

adjective

: you're looking at it.

Demo

Resources

- Differences between Jasmine and Mocha <https://www.codementor.io/javascript/tutorial/javascript-testing-framework-comparison-jasmine-vs-mocha>
- NG2 CLI <https://github.com/angular/angular-cli>
- Why NPM <https://medium.freecodecamp.com/why-i-left-gulp-and-grunt-for-npm-scripts-3d6853dd22b8#.p9a02z5lf>
- Mocha with Angular <https://github.com/angular/angular/issues/11230>
- Angular2 API <https://angular.io/docs/ts/latest/api/>
- Angular Universal <https://github.com/angular/universal>
- All about offline first <https://jakearchibald.com>

Questions and Answers

*“Keep calm and
enjoy the Offline Status!”*

Thanks!

– Giorgio Natili