



Facultad de  
**INFORMÁTICA**



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Trabajo de extensión

## Desarrollo de PWA y tests de unidad

**Materia:** Proyecto de Software

**Alumnos:** Giorgetti, Valentín (17133/2) - Scoffield, David (17282/5)

# Índice

<b>1. Resumen</b>	<b>2</b>
<b>2. Palabras claves</b>	<b>2</b>
<b>3. Tecnologías utilizadas</b>	<b>2</b>
<b>4. Explicación del desarrollo</b>	<b>2</b>
4.1. PWA	2
4.2. Testing	12
<b>5. Aspectos a destacar en el desarrollo</b>	<b>16</b>
<b>6. Conclusiones</b>	<b>16</b>
<b>7. Anexo sobre el análisis del impacto ético del trabajo realizado</b>	<b>17</b>
<b>8. Fuentes de información consultadas</b>	<b>18</b>

# 1. Resumen

Con el pasar del tiempo y el incremento del uso de dispositivos móviles los últimos años, se ha visto la necesidad en los desarrolladores webs de anticipar las necesidades de los usuarios sobre estos dispositivos. Una de las grandes demandas ha sido proveer una experiencia de usuario alta, independientemente de la conectividad con la red, como sucede por lo general con las aplicaciones nativas. Aquí aparece el primer tema de desarrollo del trabajo, las PWA. En pocas palabras, una PWA es una aplicación web normal que aprovecha distintos tipos de tecnologías modernas del navegador para mejorar la experiencia de uso. El núcleo de las PWA es el service worker, el cual nos permitirá interceptar las peticiones al servidor, para trabajar con ellas previamente. Además, gracias a un “manifest” de la aplicación, la misma podrá ser instalada en el navegador, como si de una aplicación nativa se tratara.

El otro tema que se desarrollará en el informe está relacionado al testing de unidad en Python, concretamente utilizando el framework “unittest”. Las pruebas unitarias o unit testing son aquellas que permiten examinar cada módulo de manera individual para asegurar que estos funcionen de manera correcta.

## 2. Palabras claves

1. PWA
2. Vue.js
3. Python
4. Test
5. Thread

## 3. Tecnologías utilizadas

Para el desarrollo de la PWA, se han empleado distintas herramientas. Entre ellas podemos mencionar a **Vue.Js**, como principal framework para la creación de la aplicación web, y sobre la cual se aplicará la PWA. Por otra parte, utilizaremos un plugin de Vue.Js, en particular “**@vue/cli-plugin-pwa**”, que nos proveerá de las herramientas necesarias y una configuración inicial de un service worker.

Otra característica utilizada en desarrollo ha sido “**workbox-strategies**”, que nos provee las estrategias de almacenamiento en caché más comunes, para trabajar con el service worker.

Por su parte, para la implementación de los tests se utilizó el framework “**unittest**”, el cuál forma parte de la librería estándar de Python. El mismo permite la ejecución automatizada de los tests de unidad y está basado en el diseño de XUnit de Kent Beck y Erich Gamma, específicamente en el de JUnit.

## 4. Explicación del desarrollo

### 4.1. PWA

Vamos a partir desde la definición de PWA:

*“Las aplicaciones web progresivas (mejor conocidas como PWA por «Progressive Web Apps») son aplicaciones web que utilizan APIs y funciones emergentes del navegador web junto a una estrategia tradicional de mejora progresiva para ofrecer una aplicación nativa —como la experiencia del usuario para aplicaciones web multiplataforma. Las aplicaciones web progresivas son un patrón de diseño útil, aunque no son un estándar formalizado. Se puede pensar que PWA es similar a AJAX u otros patrones similares que abarcan un conjunto de atributos de aplicación, incluido el uso de tecnologías y técnicas web específicas.”<sup>1</sup>*

Como vemos, las PWA nos sirven para proporcionar grandes mejoras a la experiencia de usuario final.

Entrando un poco más en profundidad y técnicamente en esta tecnología, las PWA, necesitarán de ciertas características básicas para su funcionamiento:

“

- **Contexto Seguro (HTTPS)**

*La aplicación web se debe servir a través de una red segura. Ser un sitio seguro no solo es una buena práctica, sino que también establece tu aplicación web como un sitio confiable, especialmente si los usuarios necesitan realizar transacciones seguras. La mayoría de las funciones relacionadas con una PWA, como la geolocalización e incluso los servicios workers, solamente están disponibles cuando la aplicación se ha cargado mediante HTTPS.*

- **Servicio workers**

*Un servicio worker es un script que permite interceptar y controlar cómo un navegador web maneja tus solicitudes de red y el almacenamiento en caché de activos. Con los servicios worker, los desarrolladores web pueden crear páginas web rápidas y fiables junto con experiencias fuera de línea.*

- **El archivo manifest**

*Un archivo JSON que controla cómo se muestra tu aplicación al usuario y garantiza que las aplicaciones web progresivas sean detectables. Describe el nombre de la aplicación, la URL de inicio, los iconos y todos los demás detalles necesarios para transformar el sitio web en un formato similar al de una aplicación.*

“ 2

En particular para el framework en el que veníamos desarrollando la aplicación, Vue.js, existe un plugin que este mismo nos provee, para la creación de una aplicación base PWA.

Estamos hablando de @vue/cli-plugin-pwa, que puede ser integrado de dos maneras al desarrollo. En una primera instancia, si la aplicación recién se está creando, utilizando vue-cli, nos aparecerán opciones entre las que podemos seleccionar que sea incluido el soporte con las PWA, como podemos observar en la Imagen [4.1.1].

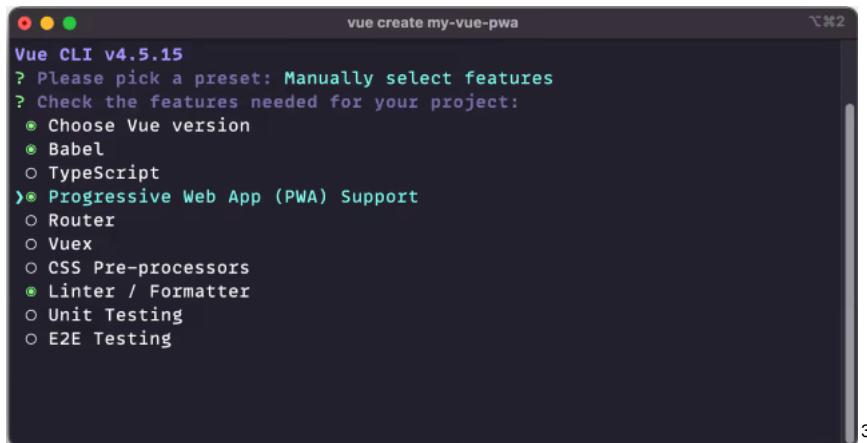
---

<sup>1</sup> Definición proporcionada por [https://developer.mozilla.org/es/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/es/docs/Web/Progressive_web_apps)

<sup>2</sup> Características básicas para funcionamiento de PWA en base a [https://developer.mozilla.org/es/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/es/docs/Web/Progressive_web_apps)

### Imagen [4.1.1]

(Representa las opciones al momento de crear la aplicación con vue-cli)



Por otra parte, en caso que el proyecto ya haya sido creado, para poder agregar la funcionalidad de las PWA al mismo, haremos uso del comando de consola **'vue add pwa'**, el cual generará algunos cambios en el proyecto.

Primero, agrega al listado de dependencias del package.json del proyecto de vue, a "register-service-worker", el cual es un script para simplificar el registro de service workers con enlaces para eventos comunes. Además, agrega como dependencia de desarrollo el plugin "@vue/cli-plugin-pwa". Podemos observar en la Imagen [4.1.2] lo mencionado anteriormente.

### Imagen [4.1.2]

(Representa las dependencias agregadas en el package.json del proyecto al agregar PWA)

```
10  "dependencies": {
11    "@popperjs/core": "2.10.2",
12    "@vue-leaflet/vue-leaflet": "0.6.1",
13    "bootstrap": "5.1.3",
14    "core-js": "^3.6.5",
15    "leaflet": "1.7.1",
16    "popper.js": "1.16.1",
17+   "register-service-worker": "^1.7.1",
18    "vue": "3.0.0",
19    "vue-router": "4.0.12"
20  },
21  "devDependencies": {
22    "@vue/cli-plugin-babel": "~4.5.0",
23    "@vue/cli-plugin-eslint": "~4.5.0",
24+   "@vue/cli-plugin-pwa": "~4.5.0",
25    "@vue/cli-service": "~4.5.0",
26    "@vue/compiler-sfc": "3.0.0",
27    "babel-eslint": "^10.1.0",
28    "eslint": "^6.7.2",
29    "eslint-plugin-vue": "^7.0.0"
30  },
```

También, el plugin se encargará de la creación del archivo registerServiceWorker.js (Imagen [4.1.3]) junto con su importación en el archivo main (Imagen [4.1.4]). Este contendrá una configuración básica del registro de un service worker.

<sup>3</sup> La imagen [4.1.1] <https://blog.logrocket.com/building-pwa-vue/>

### Imagen [4.1.3]

(Representa el archivo registerServiceWorker.js generado por el plugin de pwa)

```
1+ /* eslint-disable no-console */      You, hace 4 meses * initial pwa vue with images
2+
3+ import { register } from 'register-service-worker' 2k (gzipped: 830)
4+
5+ if (process.env.NODE_ENV === 'production') {
6+   register(`${process.env.BASE_URL}service-worker.js`, {
7+     ready() {
8+       console.log(
9+         'App is being served from cache by a service worker.\n' +
10+         'For more details, visit https://goo.gl/AFskqB'
11+       )
12+     },
13+     registered() {
14+       console.log('Service worker has been registered.')
15+     },
16+     cached() {
17+       console.log('Content has been cached for offline use.')
18+     },
19+     updatefound() {
20+       console.log('New content is downloading.')
21+     },
22+     updated() {
23+       console.log('New content is available; please refresh.')
24+     },
25+     offline() {
26+       console.log('No internet connection found. App is running in offline mode.')
27+     },
28+     error(error) {
29+       console.error('Error during service worker registration:', error)
30+     },
31+   })
32+ }
```

### Imagen [4.1.4]

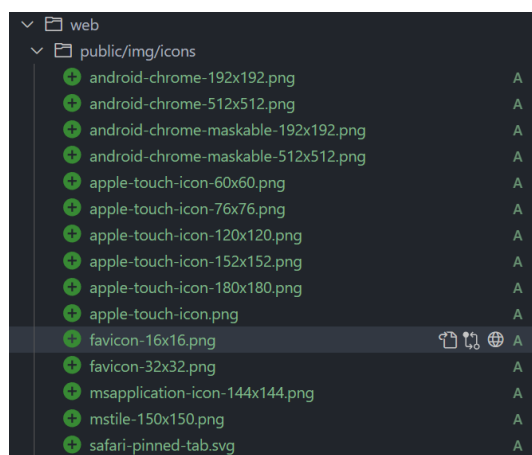
(Representa el código agregado por el plugin pwa al archivo main.js)

```
1 import { createApp } from 'vue' 48.5k (gzipped: 19k)
2 import App from './App.vue'
3 import router from './router'
4 import 'leaflet/dist/leaflet.css'
5 import './assets/global.css'
6 // import 'jquery'
7 import '@popperjs/core' 19.6k (gzipped: 7.2k)
8 import 'bootstrap/dist/js/bootstrap.bundle' 79.4k (gzipped: 24.1k)
9 import 'bootstrap/dist/css/bootstrap.min.css'
10 import 'bootstrap' 59k (gzipped: 16.2k)
11+ import './registerServiceWorker'
12
13 createApp(App).use(router).mount('#app')
14 |
```

Por último, generará en la carpeta public/img/icons las imágenes por defecto para los distintos usos de la PWA (Imagen [4.1.5]).

### Imagen [4.1.5]

(Representa los archivos de imagen generados por el plugin pwa)



A partir de este momento, tenemos una PWA totalmente funcional y con características básicas sobre nuestro proyecto.

En base a la documentación oficial del plugin, para poder configurar los parámetros básicos de la PWA, debíamos crear en el directorio root de la aplicación, el archivo “**vue.config.js**”. A través del mismo, tendríamos la posibilidad de cambiar parámetros tales como, el color del tema de la aplicación, el nombre, el título, los iconos que usaría, y uno de los más importantes, el modo con el que trabajaría el módulo **workbox-webpack-plugin**. El plugin nos ofrece dos clases, “**GenerateSW**” y “**InjectManifest**”, cada una orientada para un uso y requisitos esperados. Por una parte el “**GenerateSW**” es el utilizado por defecto, si no se especifica lo contrario, y es utilizado principalmente cuando se quiere solamente precachear archivos, y se tienen necesidades simples de cacheo en ejecución. Cuando se quiere, algo más complejo, y más configurable, podemos optar por el “**InjectManifest**”, el cual, no ofrece más control sobre el service worker, customización de rutas y estrategias de cacheo, y por ejemplo, cuando queremos usar el service worker con otras características de la plataforma como Web Push.

A partir de esta información, decidimos ir por el camino que nos ofrecía el mecanismo de “**InjectManifest**”, a pesar de que no lo hayamos explorado en su totalidad de posibilidades.

Nuestro **vue.config.js** comenzaba a verse como la imagen [4.1.6], donde establecemos el nombre de la aplicación, junto con los colores, algunas características para los dispositivos Apple, y como describíamos anteriormente, la utilización de la opción del “**InjectManifest**”, junto con la especificación de la ruta donde se encontraba el archivo **service-worker.js** (imagen [4.1.7]), el cual era obligatorio.

#### Imagen [4.1.6]

(Muestra la configuración del archivo **vue.config.js**)

```
1  module.exports = {  
2    // ... other vue-cli plugin options ...  
3    pwa: {  
4      name: 'InundaSafe App',  
5      themeColor: '#fff',  
6      msTileColor: '#000000',  
7      appleMobileWebAppCapable: 'yes',  
8      appleMobileWebAppStatusBarStyle: 'black',  
9    },  
10   // configure the workbox plugin  
11   workboxPluginMode: 'InjectManifest',  
12   workboxOptions: {  
13     // swSrc is required in InjectManifest mode.  
14     swSrc: 'src/service-worker.js',  
15     // ... other Workbox options ...  
16   },  
17 },  
18 }  
19
```

#### Imagen [4.1.7]

(Ilustra el contenido del archivo service-worker.js)

```
1 self.__precacheManifest = [].concat(self.__precacheManifest || [])
2 // workbox.precaching.suppressWarnings()
3 workbox.precaching.precacheAndRoute(self.__precacheManifest, {})
4
5 // install new service worker when ok, then reload page.
6 self.addEventListener('message', (msg) => {
7   if (msg.data.action == 'skipWaiting') {
8     self.skipWaiting()
9   }
10 })
11 |
```

Una de las primeras cosas que hicimos con el service worker, es **detectar cuando hay contenido nuevo disponible**(cuando se encuentra un nuevo service worker), **notificar al usuario y dar la opción de actualizar la página**. Para esto escuchamos un evento que se activa cuando se encuentra el nuevo service worker y está listo para instalarse. El service worker se encontrará en el estado 'waiting'. Esta funcionalidad se podría hacer manualmente donde las herramientas de desarrollo de Chrome, en el apartado de Aplicación, dando en el botón 'skipWaiting' de la pestaña "service-worker".

Ahora que tenemos un archivo service-worker.js personalizado, escucharemos en el registerServiceWorker (que se ejecuta en la ventana del navegador) el evento 'updated' y enviaremos un mensaje al service-worker.js (que se ejecuta en otro thread). Por último, escucharemos los cambios en el registro del service worker y activaremos la recarga de la página, de esta manera se cargará el nuevo service-worker.

Tal como se ve en la imagen [4.1.8] controlaremos y trabajaremos sobre el evento updated. Allí crearemos una ventana de confirmación para que el usuario opte por actualizar el contenido o no, y una vez que el usuario confirme, se envía un mensaje al archivo service-worker.js con la acción 'skipWaiting'.

En el service-worker.js de la imagen [4.1.9] se agregaron las líneas para la escucha del evento "message", y una vez que se compruebe que la acción que la activa es 'skipWaiting', se activará esta función.

Una última cosa que debe hacerse es escuchar el evento "controllerchange" en el archivo registerServiceWorker.js y activar una recarga de página (como se ve al final de la imagen [4.1.8])



#### Imagen [4.1.8]

(Refleja los cambios del archivo registerServiceWorker.js)

```
1  /* eslint-disable no-console */
2
3  import { register } from 'register-service-worker' 2k (gzipped: 830)
4
5  if (process.env.NODE_ENV === 'production') {
6    register(`${process.env.BASE_URL}service-worker.js`, {
7      ready() {
8        console.log(
9          'App is being served from cache by a service worker.\n' +
10           'For more details, visit https://goo.gl/AFskqB'
11        )
12      },
13      registered() {
14        console.log('Service worker has been registered.')
15      },
16      cached() {
17        console.log('Content has been cached for offline use.')
18      },
19      updatefound() {
20        console.log('New content is downloading.')
21      },
22+    updated(registration) {
23+      console.log('New content is available; please refresh.')
24+      let confirmationResult = confirm('New content found! Do you want to reload the app?')
25+      if (confirmationResult) registration.waiting.postMessage({ action: 'skipWaiting' })
26+    },
27      offline() {
28        console.log('No internet connection found. App is running in offline mode.')
29      },
30      error(error) {
31        console.error('Error during service worker registration:', error)
32      },
33    })
34+
35+    let refreshing
36+    navigator.serviceWorker.addEventListener('controllerchange', () => {
37+      if (refreshing) return
38+      window.location.reload()
39+      refreshing = true
40+    })
41  }
42
```

#### Imagen [4.1.9]

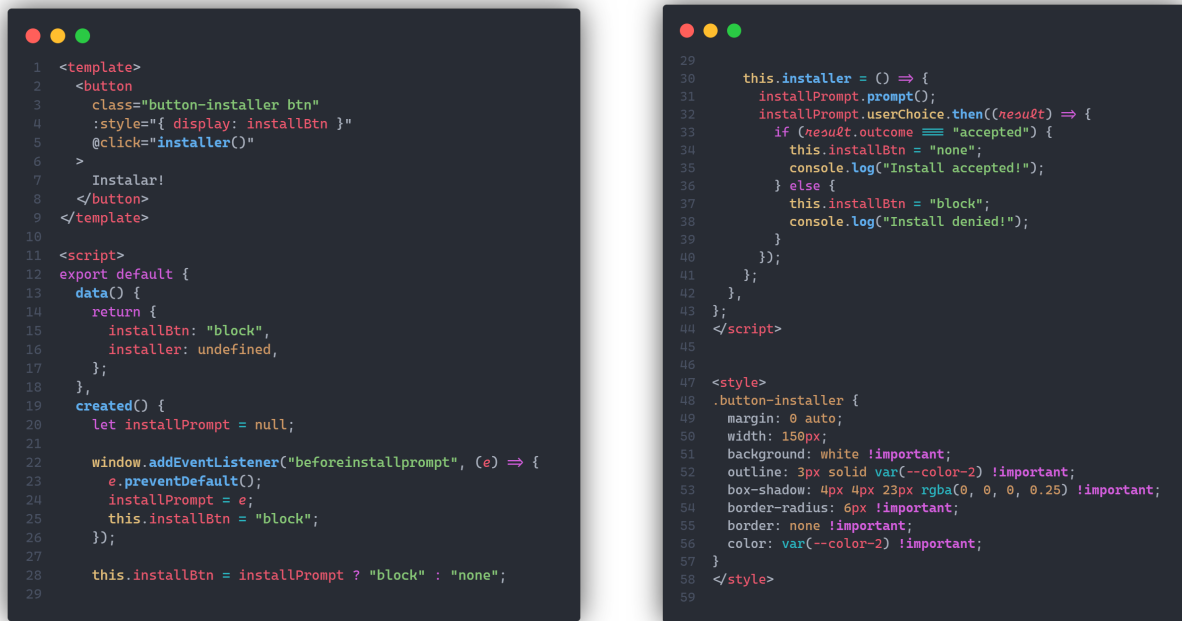
(Refleja el código agregado al service-worker.js para la funcionalidad de actualizar el contenido)

```
4+
5+ // install new service worker when ok, then reload page.
6+ self.addEventListener('message', (msg) => {
7+   if (msg.data.action === 'skipWaiting') {
8+     self.skipWaiting()
9+   }
10+ })
11+
```

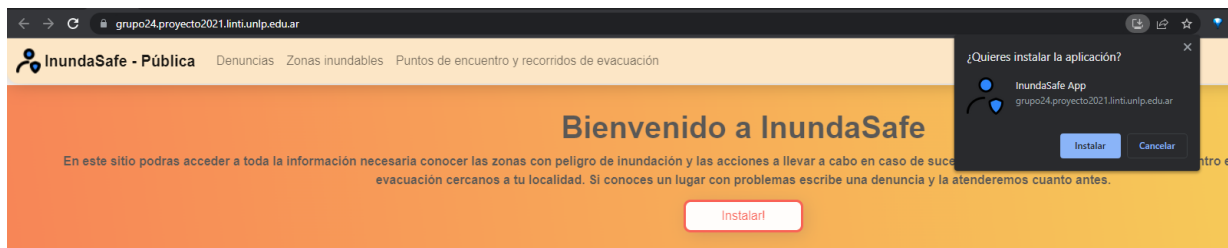
El siguiente paso/funcionalidad, fue incorporar visualmente a la página un acceso (botón) que permita instalar la aplicación de escritorio/móvil. Para llevarla a cabo, no fue necesario modificar el service-worker, ya que se podía hacer con la configuración predeterminada de Vue/PWA.

Creamos un nuevo componente llamado InstallButton (Imagen [4.1.10]), que la principal característica es mostrar un botón en el caso que se detecte que la aplicación cumple con los requisitos de una PWA (a través del evento “beforeinstallprompt” del navegador), y si se clickea sobre él, se recibirá una notificación de instalación predeterminada del navegador (Imagen [4.1.11]); y en caso de aceptar, la misma se instalará en el dispositivo.

**Imagen [4.1.10]**  
(Representa el código del archivo InstallButton.vue)



**Imagen [4.1.11]**  
(Muestra el botón de instalación en la página web)



Finalmente, como última característica para el desarrollo de la PWA, agregamos el cacheo de las consultas de datos a la API, para que una vez que se tengan localmente, en caso de no poseer disponibilidad de conexión a la red, la aplicación, mostrará a pesar de ello, los últimos datos recuperados hasta el momento en peticiones pasadas.

Para poder realizar esta funcionalidad, investigamos las distintas estrategias de caching en service workers. Workbox a través de la API de “workbox-strategies” nos provee diferentes opciones.

“

- Cache First: si hay una respuesta en la memoria caché, la solicitud se cumplirá con la respuesta almacenada en la memoria caché y la red no se utilizará en absoluto. Si no hay una respuesta en caché, la solicitud se cumplirá mediante una solicitud de red y la respuesta se almacenará en caché para que la siguiente solicitud se atienda directamente desde la memoria caché.
- Network First: de forma predeterminada, intentará obtener la última respuesta de la red. Si la solicitud es exitosa, colocará la respuesta en el caché. Si la red no puede devolver una respuesta, se utilizará la respuesta almacenada en caché.
- Network only: como su nombre lo indica, solo utilizará la red.

- Cache only: la estrategia de sólo caché garantiza que las respuestas se obtengan de un caché. Esto es menos común en Workbox, pero puede ser útil si tiene su propio paso de almacenamiento en caché.

“ 4

Para este desarrollo, hemos optado por la estrategia de Network First, la cual cumplía con nuestros objetivos iniciales.

Utilizamos como se muestra en la imagen [4.1.12], el método `registerRoute()` que provee `workbox.routing`, y le especificamos como primer parametro la ruta que queremos que atienda y cachee; en este caso todas las que se hagan a “<https://admin-grupo24.proyecto2021.linti.unlp.edu.ar/api/>”, y como segundo parametro la estrategia especifica a utilizar, Network First.

Cada vez que se haga una consulta a ese dominio, el service worker guardará cada registro en el Cache Storage (dentro de `inundSAFE-api-cache`, que fue el nombre que definimos en `cacheName` de la Imagen [4.1.12]). Esto lo podemos observar desde la ventana Application de las herramientas de desarrollador del navegador, como vemos en la Imagen [4.1.13]

---

<sup>4</sup> Definición de estrategias. <https://developer.chrome.com/docs/workbox/modules/workbox-strategies/>

### Imagen [4.1.12]

(Muestra la modificación del archivo service-worker.js para la implementación de cacheo a la API)

```
1 self.__precacheManifest = [].concat(self.__precacheManifest || [])
2 // workbox.precaching.suppressWarnings()
3 workbox.precaching.precacheAndRoute(self.__precacheManifest, {})
4
5+ workbox.routing.registerRoute(
6+   /https:\/\/admin-grupo24.proyecto2021.linti.unlp.edu.ar\/api\/.*/,
7+   workbox.strategies.networkFirst({
8+     cacheName: 'inundasafe-api-cache',
9+     plugins: [
10+       new workbox.expiration.Plugin({
11+         maxEntries: 600,
12+         maxAgeSeconds: 1200
13+       }),
14+       new workbox.cacheableResponse.Plugin({
15+         statuses: [0, 200]
16+       }),
17+     ],
18+   })
19+ )
20+
21 // install new service worker when ok, then reload page.
22 self.addEventListener('message', (msg) => {
23   if (msg.data.action == 'skipWaiting') {
24     self.skipWaiting()
25   }
26 })
27
```

### Imagen [4.1.13]

(Muestra donde se almacenan los registros del cacheo que realiza el service worker)


InundaSafe - Pública

Denuncias Zonas inundables Puntos de encuentro y recorridos de evacuación

Iniciar Sesión

← Denuncias

Nueva denuncia



Titulo	Categoria	Estado
Alcantarilla tapada desde API	Alcantarilla tapada	Resuelta
Alcantarilla tapada desde API	Alcantarilla tapada	En curso
Alcantarilla tapada desde API	Alcantarilla tapada	En curso
Alcantarilla tapada desde API	Alcantarilla tapada	En curso
Alcantarilla tapada desde API dos	Alcantarilla tapada	En curso

Anterior 1 2 Siguiente

Application

Manifest

Service Workers

Storage

#	Name	Resp...	Cont...	Cont...	Time...	Vary...
0	/api/colors	cors	text/...	0	28/5...	Orig...
1	/api/denuncias?pagina=1	cors	appli...	2,878	28/5...	Orig...
2	/api/zonas-inundables?pagina=1	cors	appli...	20,664	28/5...	Orig...

Storage

Cache

Background Services

Frames

Select a cache entry above to preview

Console

## 4.2. Testing

### Tests de unidad

Una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código, específicamente en diseño orientado a objetos, una clase. Cada una de estas pruebas debe ser independiente (su ejecución no afecta a la ejecución de otra), reutilizable (puede ejecutarse múltiples veces) y automatizable (no requiere ningún tipo de intervención manual).

Para llevar a cabo buenas pruebas unitarias, es recomendable que estén estructuradas siguiendo las tres A's del Unit Testing, que describe un proceso compuesto de tres pasos:

“

- Arrange (organizar): es el primer paso de las pruebas unitarias, el cuál consiste en definir los requisitos que debe cumplir el código.
- Act (actuar): es el paso intermedio donde se ejecuta el test que dará lugar a los resultados que posteriormente deberán ser analizados.
- Assert (afirmar): es el último paso de las pruebas unitarias, donde se comprueba si los resultados obtenidos son los esperados; en caso contrario se corrige el error hasta solucionarlo.

“ (YeePLY, 2021)

### ¿Cuál es la motivación para realizar tests de unidad?

Realizar tests de unidad nos provee varios beneficios, por ejemplo:

- Fomentan el cambio: los programadores pueden refactorizar el código para mejorar su estructura y al disponer de las pruebas de unidad se aseguran que dichos cambios no han introducido errores.
- Simplifica la integración: permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente.
- Documenta el código: los test unitarios sirven como documentación del proyecto.
- Mejora la legibilidad del código: ayuda a los desarrolladores a entender el código base, lo cuál facilita hacer cambios más rápidamente.
- Los errores están más acotados y son más fáciles de localizar: además, como las pruebas unitarias dividen el código en pequeños fragmentos, es posible probar distintas partes del proyecto sin tener que esperar a que otras estén completas.

### Framework para automatización de tests unitarios: unittest

Tal como se menciona en la guía oficial “*unittest - Unit testing framework*”, la infraestructura de los tests unitarios de este framework se inspiró en la de *JUnit* y ofrece aspectos similares a las principales estructuras de tests unitarios más importantes de otros lenguajes.

Provee automatización de tests, permite compartir código de inicialización (setup) y finalización (shutdown) entre tests, incluir tests en colecciones para crear de conjuntos de pruebas e independencia de los tests de la infraestructura que los reporta.

Para conseguir esto, unittest da soporte a los siguientes conceptos de una forma orientada a objetos:

- *Test fixture* (configuración de prueba): representa los preparativos para realizar una o más pruebas y las acciones de “limpieza” asociadas.
- *Test case* (caso de prueba): es la unidad mínima de prueba. Verifica una respuesta específica ante un determinado conjunto de inputs. Se provee una clase base, *TestCase*, la cuál se puede utilizar para crear nuevos test cases.
- *Test suite* (conjunto de pruebas): es una colección de test cases, test suites o ambos. Se usa para agrupar pruebas que se han de ejecutar juntas.
- *Test runner* (ejecutor de pruebas): es un componente que dirige la ejecución de las pruebas y proporciona el resultado de ejecución de las mismas.

### ¿Cómo se escriben las pruebas de unidad en unittest?

Para crear un nuevo *test case* se debe crear una subclase de “*TestCase*”, definida por *unittest*. Luego, dentro de esta clase, los tests individuales son definidos a partir de métodos que comienzan con la palabra “*test*”, de esta manera, el *test runner* puede identificar cuales son los tests que debe ejecutar.

Dentro de estos métodos, para poder testear una determinada condición se utiliza algún método *assert\**() proporcionado por la clase base *TestCase*, tales como *assertEqual()*, *assertNotEqual()*, *assertTrue()*, *assertFalse()*, entre otros. Si la prueba falla, se generará una excepción con un mensaje explicativo y *unittest* identificará el *test case* como una *falla*. Cualquier otra excepción será tratada como un *error*.

El método *setUp()* se utiliza para preparar el *test fixture*, se ejecuta por cada método de test, inmediatamente antes de comenzar a ejecutarlo. Aquí puede definirse un código de inicialización que sea común a todos los tests. Si el método *setUp()* genera una excepción mientras se ejecuta, el framework considerará que la prueba ha sufrido un *error* y el método de test no se ejecutará. El método *tearDown()* será ejecutado una vez que finalice cada método de test.

Entonces, la estructura sería la siguiente:

```
import unittest

class TestEjemplo(unittest.TestCase):

    def setUp(self):
        # inicialización del test fixture

    def test_prueba1(self):
        ...
        self.assertEqual(...)
        ...
        self.assertFalse(...)
```

```

...

...

def test_pruebaN(self):
    ...
    self.assertTrue(...)
    ...
    self.assertNotEqual(...)

def tearDown(self):
    # limpieza luego de ejecutar el test

if __name__ == '__main__':
    unittest.main()

```

Luego, para ejecutar los tests, se puede correr el comando `python -m unittest -v`. De esta forma se podrá observar el resultado de la ejecución de los tests como salida del comando anterior, el cuál puede ser “ok” (el test pasó correctamente), “FAIL” (el test no pasó y generó una excepción *AssertionError*) o “ERROR” (el test generó una excepción que no es *AssertionError*).

### Utilización de unittest para diferentes modelos del proyecto

Empleamos el framework unittest para realizar los tests de unidad sobre los modelos de usuario, recorridos de evacuación y zonas de inundación, de la siguiente forma:

- `__init__.py`: archivo donde se implementa la clase “*BaseTestClass*”, la cual extiende de “*unittest.TestCase*”. En esta clase se definen los métodos `setUp()` y `tearDown()`, los cuales son comunes a los test cases que serán implementados, para que luego éstos hereden de *BaseTestClass* y puedan acceder a ellos.

El método `setUp()` inicializa el *test fixture*. Para esto, en primer lugar liga a la instancia del test case una instancia de una aplicación flask en entorno de testing, la cual tiene asociada una base de datos únicamente para este fin, llamada “*grupo24tests*”. Luego, inicializa esta base de datos creando tablas con algunos datos necesarios para los tests, como por ejemplo la cantidad de páginas de los listados, criterio de ordenamiento, entre otros. El método `tearDown()` elimina los datos de las pruebas.

- `test_user.py`: archivo donde se implementa la clase “*TestUserMethods*”, la cual extiende de *BaseTestClass*. En esta clase se extiende el método `setUp()` para crear algunos usuarios que serán parte del *test fixture*. Luego se define una serie de tests:
  - `test_verify_password()`: verifica que la contraseña ingresada se corresponda con la contraseña del usuario, la cuál está hasheada y almacenada en la base de datos.
  - `test_delete_user()`: verifica que el usuario se elimine lógicamente de la base de datos.
  - `test_find_by_email_and_pass()`: verifica si en la base de datos existe un usuario con un email y contraseña dados.

- *test\_find\_user\_by\_id()*: verifica que se retorne el usuario con el id buscado.
  - *test\_find\_user\_by\_id\_not\_deleted()*: verifica que el id retornado se corresponda con el del usuario buscado y que no esté borrado lógicamente.
  - *test\_check\_existing\_email\_or\_username()*: verifica que en la base de datos no exista otro usuario con el mismo email o nombre de usuario.
- *test\_evacuation\_route.py*: archivo donde se implementa la clase “*TestEvacuationRouteMethods*”, la cual extiende de *BaseTestClass*. En esta clase se extiende el método *setUp()* para crear algunos recorridos de evacuación que serán parte del *test fixture*. Luego se define una serie de tests:
    - *test\_all()*: verifica que se retornen todos los recorridos de evacuación publicados y paginados, teniendo en cuenta la cantidad de elementos por página.
    - *test\_exists\_name()*: verifica si en la base de datos existe otro recorrido de evacuación con un nombre dado.
    - *test\_search()*: verifica que se retorne una lista con los recorridos de evacuación paginados teniendo en cuenta los filtros (nombre y estado *publicado* o *despublicado*).
    - *test\_new()*: verifica que se cree un nuevo recorrido y se almacene en la base de datos con los datos ingresados.
    - *test\_update()*: verifica que se actualicen los datos del recorrido de evacuación y se almacenen correctamente en la base de datos.
  - *test\_flood\_zones.py*: archivo donde se implementa la clase “*TestFloodZoneMethods*”, la cual extiende de *BaseTestClass*. En esta clase se extiende el método *setUp()* para crear algunas zonas de inundación que serán parte del *test fixture*. Luego se define una serie de tests:
    - *test\_find\_by\_id()*: verifica que se retorne la zona inundable correspondiente a un id dado.
    - *test\_find\_by\_name()*: verifica que se retorne una zona inundable con un nombre dado.
    - *test\_delete\_coordinates()*: verifica que se eliminen todas las coordenadas de una zona inundable.
    - *test\_add\_coordinates()*: verifica que se agreguen las coordenadas recibidas a la zona inundable.
    - *test\_delete()*: verifica que una zona inundable se elimine correctamente de la base de datos.

Finalmente, para ejecutar los tests, crear una base de datos con el nombre “*grupo24tests*” (las credenciales a utilizar se encuentran especificadas en el archivo “*config.py*”, en la definición de la clase “*TestingConfig*”) y luego ejecutar el comando *python -m unittest -v* desde la raíz del proyecto.



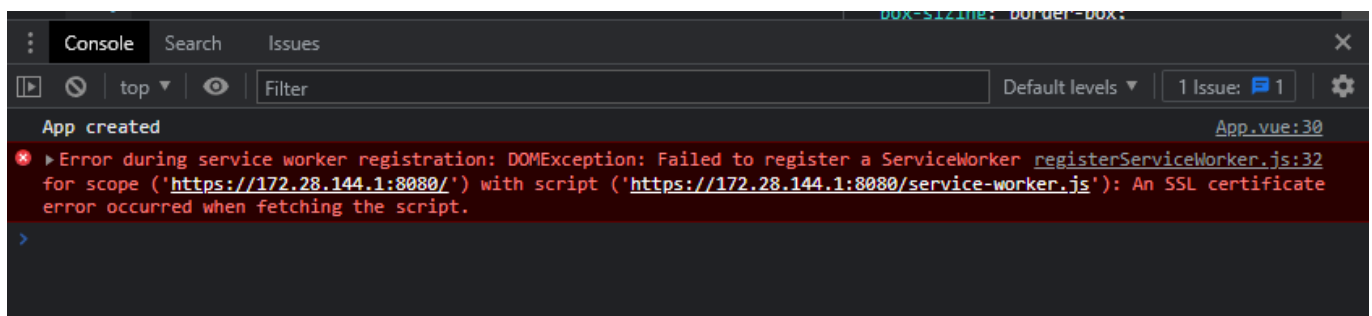
## 5. Aspectos a destacar en el desarrollo

Como ya mencionamos al principio del informe, las PWA para su correcto funcionamiento necesitan entre otras cosas, de un entorno seguro (HTTPS). Esto a la hora de desarrollar ha generado ciertos conflictos. Principalmente, porque ciertas funcionalidades si no se poseía un certificado SSL válido, no era posible registrar correctamente el service worker, como podemos ver en la Imagen [5.1] al levantar la aplicación sobre un servidor local. Además, otra de las restricciones que nos encontramos a la hora de iniciar el desarrollo, fue que las PWA sólo corren en ambientes productivos, ¿que quiere decir esto?, que para poder testear cualquier cambio que hacíamos, debíamos construir nuestra aplicación, por ejemplo, con el comando **“npm run build”**, lo cual ya nos generaba una demora extra, y luego levantar los archivos estáticos creados por el comando sobre algún servidor. Nosotros por ejemplo, hemos utilizado el paquete de npm **“servor”** que nos permitió servir contenido estático sobre un servidor local rápidamente.

Como se puede ver, y comparando con la facilidad que nos provee Vue.js para desarrollar aplicaciones, en donde solo ejecutábamos el comando **“npm serve”**, y podíamos desarrollar con la actualización automática de los cambios, la implementación de PWA nos obligó a llevar a cabo algunos pasos extra.

**Imagen [5.1]**

(Muestra el error arrojado en la consola del navegador cuando no se tenía un certificado SSL válido registrado)



Con respecto a la implementación de los tests, la parte más complicada es efectivamente plantear casos de prueba que realmente verifiquen que el comportamiento del sistema es el esperado y a su vez especificar un test fixture que sea útil para la mayoría de los test cases. Por otro lado, aprender el framework unittest no representó una dificultad demasiado grande, ya que la documentación oficial es de muy buena calidad y además hay una gran cantidad de recursos que presentan diversos ejemplos de cómo utilizarlo, desde videos explicativos hasta artículos.

## 6. Conclusiones

Como hemos visto a lo largo de este informe, las PWA agregaron un conjunto de características interesantes hacia el usuario en nuestra aplicación. Gracias al plugin **“@vue/cli-plugin-pwa”** que nos provee Vue.js, tuvimos de manera rápida un primer acercamiento a la PWA. Luego a partir de allí, investigamos sobre diferentes herramientas para potenciar el funcionamiento de las misma. A lo largo, de este proceso, nos fuimos encontrando con diferentes complicaciones, como la obligatoriedad de correr la aplicación en modo producción, teniendo que construirla y deployarla en un servidor local frente a cada cambio, y los problemas con los certificados SSL que para algunas funcionalidades, requeríamos de subir el desarrollo al servidor de producción de la cátedra para poder probarlos.

A pesar de lo mencionado, realizando las correspondientes investigaciones en el mundo de la Internet, concluimos con un correcto funcionamiento de estas características sobre la aplicación.

A su vez, al implementar tests de unidad, tenemos mayor seguridad para afirmar que el sistema actúa en base al comportamiento esperado, además es una forma de documentar y mejorar la legibilidad del código. También permite que los errores estén más acotados y sean más fáciles de identificar, lo cual minimiza la complejidad a la hora de integrar los diferentes componentes. Esto logra que la calidad del proyecto en general se vea incrementada notablemente.

## 7. Anexo sobre el análisis del impacto ético del trabajo realizado

### David Scoffield

En este punto del trabajo, voy a hacer un análisis sobre el proceso y los recursos que hemos utilizado para llevar a cabo el desarrollo de todo el proyecto.

Como es evidente, para poder concretar el desarrollo del mismo, hemos necesitado de diferentes recursos, librerías, datos del usuario, etc. Observando ahora en específico sobre las librerías que se utilizaron, veo que todas son de libre acceso y uso, pero tengo que comentar que durante el proceso de desarrollo, no era algo que teníamos en cuenta, y que es un factor decisivo a la hora de elegir qué usar y qué no. Recursos como imágenes, o íconos, no hemos tenido que preocuparnos, ya que todos eran o creaciones propias, o capturas de pantalla sobre nuestra propia aplicación.

Otro factor a mencionar, es el tema de la accesibilidad, que por temas de tiempos de entrega y demás, no ha sido un punto fuerte de nuestra aplicación, porque a pesar de tener en consideración, por ejemplo, que las paletas de colores, cumplan en su mayoría con los criterios de accesibilidad visual, otros aspectos como utilización de metatags y herramientas para facilitar el acceso a personas con quizás discapacidades visuales o motrices no fueron totalmente abordadas.

Finalmente, con respecto a la privacidad del usuario, en el desarrollo del login con Google de la aplicación privada, le solicitamos al usuario que nos de permisos sobre la menor cantidad de datos personales que necesitábamos, como nombre de usuario, email, nombre y apellido, los cuales son públicos inicialmente.

### Valentín Giorgetti

Haciendo un análisis del impacto ético del trabajo, con respecto a las licencias de las herramientas utilizadas, todas ellas son de acceso libre. Todas las releases de Python son Open Source. También puede mencionarse que Flask, emplea la licencia “BSD-3-Clause Source License”, la cual permite uso libre del software siempre y cuando se incluya el aviso de licencia y derechos de autor de BSD. A su vez, el framework Vue.js usa la licencia “MIT License”, la cuál permite uso privado y comercial, distribución y modificación del mismo. Por otro lado, recursos tales como imágenes o íconos utilizados en el sistema, fueron creados por integrantes del grupo de desarrollo.

Otro aspecto importante a mencionar está relacionado a la accesibilidad. Durante el desarrollo la idea era cumplir con la WCAG (Web Content Accessibility Guidelines, directrices de accesibilidad para el contenido web) publicada por la WAI (Web Accessibility Initiative, iniciativa de accesibilidad web), para hacer que el contenido web sea más accesible a personas con discapacidades o con dispositivos con diferentes limitaciones (como teléfonos celulares con pequeñas pantallas), intentando cumplir con los

diferentes objetivos que propone (perceptible, operable, entendible y robusto). Para lograr esto utilizamos distintos validadores, como WAVE (Web Accessibility Evaluation Tool), los cuales nos ayudaron a detectar diferentes defectos, por ejemplo, errores de contraste, y de esa forma fuimos corrigiendo las paletas de colores que ofrecimos en la aplicación para que el contenido sea más legible.

Por cuestiones de tiempo, no pudimos enfocarnos demasiado en este último aspecto, pero considero que es muy relevante, ya que la OMS (Organización Mundial de la Salud) afirma que un gran porcentaje de la población mundial (alrededor del 15%) padece de alguna forma de discapacidad. Por lo tanto, para futuros desarrollos, definitivamente desde el comienzo se tendrá presente el aspecto de accesibilidad.

## 8. Fuentes de información consultadas

- *¿Qué son las pruebas unitarias y cómo llevar una a cabo?* | YeePLY. Recuperado de: <https://www.yeeply.com/blog/que-son-pruebas-unitarias/> [Último acceso: Junio 2022]
- *unittest - Unit testing framework* | Python documentation. Recuperado de: <https://docs.python.org/3/library/unittest.html> [Último acceso: Junio 2022]
- *unittest - Marco de prueba automatizado* | Ernesto Rico Schmidt. Recuperado de: <https://rico-schmidt.name/pymotw-3/unittest/index.html> [Último acceso: Junio 2022]
- *Aplicaciones Web Progresivas* | MDN Web docs. Recuperado de: [https://developer.mozilla.org/es/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/es/docs/Web/Progressive_web_apps) [Último acceso: Junio 2022]
- *workbox-webpack-plugin, Which Plugin to Use* | Chrome Developers Docs. Recuperado de: [https://developer.chrome.com/docs/workbox/modules/workbox-webpack-plugin/#which\\_plugin\\_to\\_use](https://developer.chrome.com/docs/workbox/modules/workbox-webpack-plugin/#which_plugin_to_use) [Último acceso: Junio 2022]
- *@vue/cli-plugin-pwa* | Vue.js CLI docs. Recuperado de: <https://cli.vuejs.org/core-plugins/pwa.html> [Último acceso: Junio 2022]
- *Getting started with PWAs and Vue 3* | John Lim. Recuperado de: <https://www.vuemastery.com/blog/getting-started-with-pwas-and-vue3/> [Último acceso: Junio 2022]