Rushabh Jain **LAB 8** **D15B - 23**

Service Worker Lifecycle

A service worker progresses through three key phases in its lifecycle:

1. Registration
2. Installation
3. Activation

Registration

Registration is the initial step where you inform the browser about your service worker. This process tells the browser where to find your service worker file and initiates its background installation. Here's an implementation example:

```
// app.js
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('./service-worker.js')
    .then(function(registration) {
      console.log('Service Worker registered successfully with scope:',
registration.scope);
    })
    .catch(function(error) {
      console.log('Service Worker registration failed:', error);
    });
}
```

This code first checks if the browser supports service workers by testing for `navigator.serviceWorker`. It then registers the service worker using `navigator.serviceWorker.register()`, which returns a promise. On successful registration, it logs the scope, which defines which files the service worker can control. If registration fails, the error is logged.

By default, the service worker's scope is its location and all subdirectories. For example, if your service worker is in the root directory, it controls requests for all files on that domain.

You can also specify a custom scope:

```
// app.js
navigator.serviceWorker.register('./service-worker.js', {
  scope: '/pages/'
});
```

Installation

When a service worker is registered, it triggers an installation event. You can listen for this event to perform tasks during installation, such as precaching resources for offline use:

```
// service-worker.js
self.addEventListener('install', function(event) {
  // Perform installation tasks
});
```

During installation, service workers often cache essential files to enable offline functionality and improve loading performance on subsequent visits.

Activation

After successful installation, the service worker enters the activation phase. If any pages are still controlled by a previous service worker, the new one enters a waiting state. It only activates when all pages using the old service worker are closed. This ensures only one service worker version runs at a time.

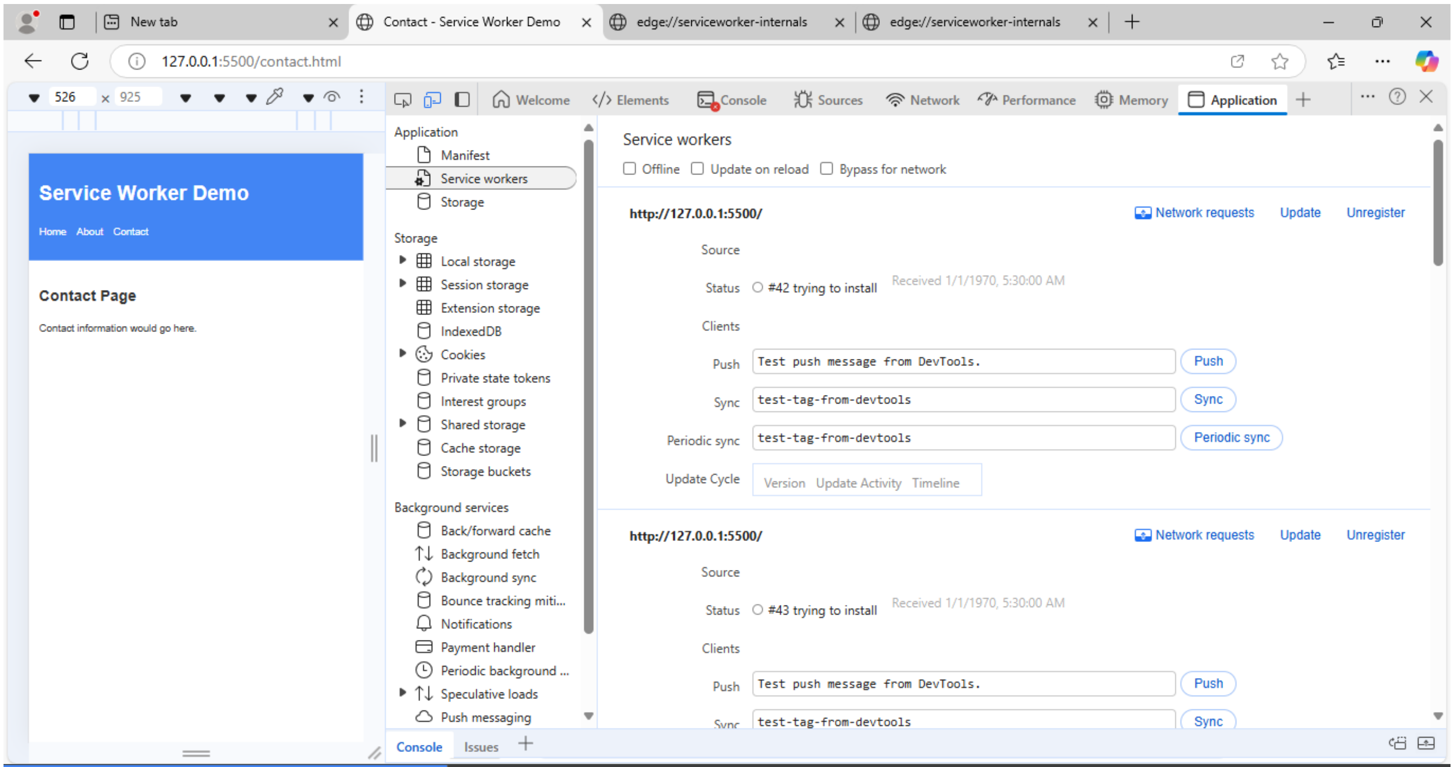
```
// service-worker.js
self.addEventListener('activate', function(event) {
  // Perform activation tasks like clearing old caches
});
```

Code

```
// service-worker.js
const CACHE_NAME = 'pwa-cache-v1';

// Resources to cache during installation
const urlsToCache = [
  '/',
  '/index.html',
  '/about.html',
  '/contact.html',
  '/offline.html',
  '/styles/main.css',
  '/scripts/app.js'
];

// Install event handler
self.addEventListener('install', function(event) {
  console.log('[Service Worker] Installing...');

  // Perform install steps
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        console.log('[Service Worker] Caching app shell');
        return cache.addAll(urlsToCache);
      })
      .then(function() {
        console.log('[Service Worker] Installation complete');
        return self.skipWaiting();
      })
  );
});

// Activate event handler
self.addEventListener('activate', function(event) {
  console.log('[Service Worker] Activating...');

  // Clean up old caches
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          if (cacheName !== CACHE_NAME) {
            console.log('[Service Worker] Deleting old cache:', cacheName);
            return caches.delete(cacheName);
          }
        })
      );
    }).then(function() {
      console.log('[Service Worker] Activation complete');
      return self.clients.claim();
    })
  );
});

// Fetch event handler
self.addEventListener('fetch', function(event) {
  console.log('[Service Worker] Fetch event for:', event.request.url);

  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return the response from the cached version
        if (response) {
          console.log('[Service Worker] Returning from cache:', event.request.url);
          return response;
        }

        // Not in cache - return the response from the network
        console.log('[Service Worker] Not in cache, fetching:', event.request.url);
        return fetch(event.request.clone())
          .then(function(response) {
            // Check if we received a valid response
            if (!response || response.status !== 200 || response.type !== 'basic') {
              return response;
            }

            // Clone the response
            var responseToCache = response.clone();

            // Add response to cache
            caches.open(CACHE_NAME)
              .then(function(cache) {
                console.log('[Service Worker] Caching new resource:', event.request.url);
                // Create explicit key-value pair
                cache.put(event.request, responseToCache);
              });

            return response;
          })
          .catch(function(error) {
            // Network request failed, try to get it from the cache
            console.log('[Service Worker] Network request failed, returning offline page');

            // For HTML requests, return the offline page
            if (event.request.headers.get('accept').includes('text/html')) {
              return caches.match('/offline.html');
            }

            // You can add additional fallbacks for images, fonts, etc.
            return new Response('Network error happened', {
              status: 408,
              headers: { 'Content-Type': 'text/plain' }
            });
          });
      })
  );
});

// Handle messages from the main thread
self.addEventListener('message', function(event) {
  if (event.data.action === 'skipWaiting') {
    self.skipWaiting();
  }
```

```
});
```

Output



Service Worker Demo

Home About Contact

Contact Page

Contact information would go here.

Service workers

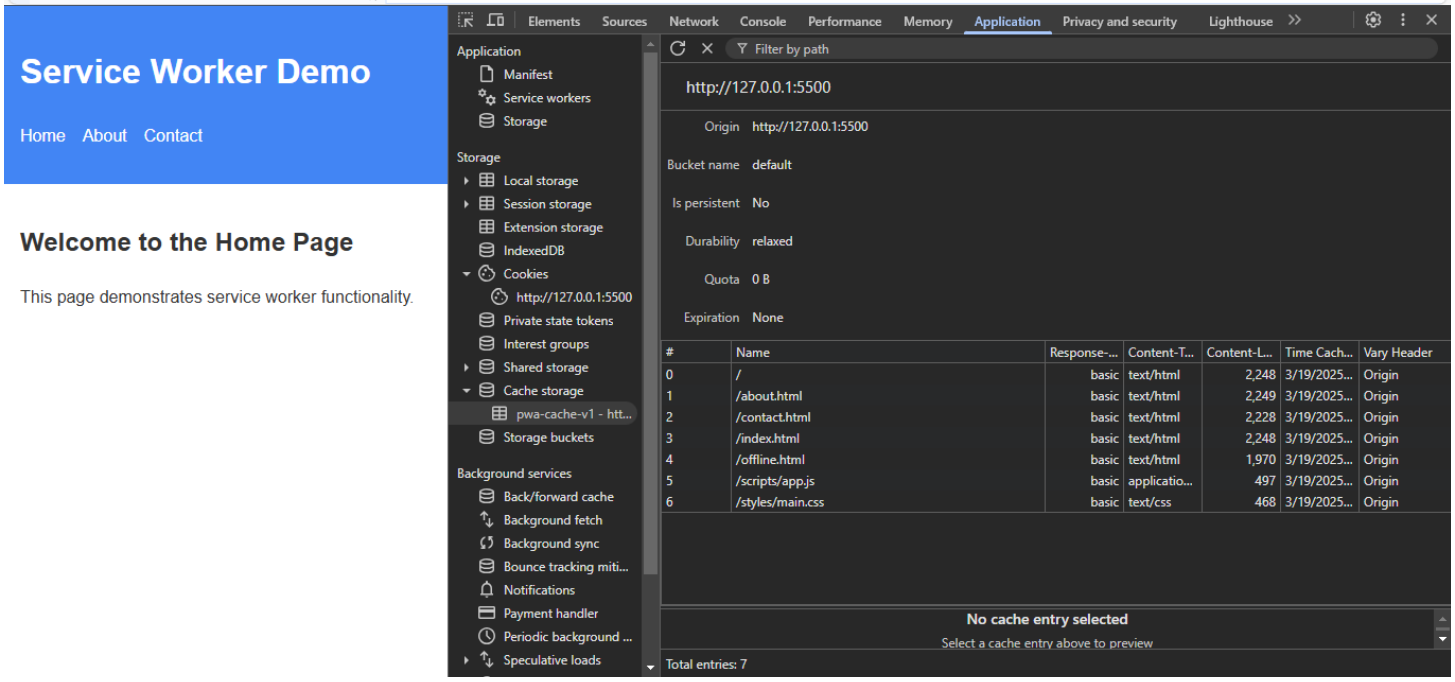
☐ Offline ☐ Update on reload ☐ Bypass for network

http://127.0.0.1:5500/

Source	
Status	#42 trying to install — Received 1/1/1970, 5:30:00 AM
Clients	
Push	Test push message from DevTools.
Sync	test-tag-from-devtools
Periodic sync	test-tag-from-devtools
Update Cycle	Version Update Activity Timeline

http://127.0.0.1:5500/

Source	
Status	#43 trying to install — Received 1/1/1970, 5:30:00 AM
Clients	
Push	Test push message from DevTools.
Sync	test-tag-from-devtools



Service Worker Demo

Home About Contact

Welcome to the Home Page

This page demonstrates service worker functionality.

http://127.0.0.1:5500

Origin	http://127.0.0.1:5500
Bucket name	default
Is persistent	No
Durability	relaxed
Quota	0 B
Expiration	None

#	Name	Response-...	Content-T...	Content-L...	Time Cach...	Vary Header
0	/	basic	text/html	2,248	3/19/2025...	Origin
1	/about.html	basic	text/html	2,249	3/19/2025...	Origin
2	/contact.html	basic	text/html	2,228	3/19/2025...	Origin
3	/index.html	basic	text/html	2,248	3/19/2025...	Origin
4	/offline.html	basic	text/html	1,970	3/19/2025...	Origin
5	/scripts/app.js	basic	applicatio...	497	3/19/2025...	Origin
6	/styles/main.css	basic	text/css	468	3/19/2025...	Origin

No cache entry selected

Select a cache entry above to preview

Total entries: 7