readme.md for @push.rocks/webrequest

Modern, fetch-compatible web request library with intelligent HTTP caching, retry strategies, and advanced fault tolerance. Works seamlessly in browsers, Node.js, Deno, and Bun.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Features

- **Fetch-Compatible API** — Drop-in replacement for native `fetch()` with enhanced features
- **Intelligent HTTP Caching** — Respects `Cache-Control`, `ETag`, `Last-Modified`, and `Expires` headers (RFC 7234)
- **Multiple Cache Strategies** — network-first, cache-first, stale-while-revalidate, network-only, cache-only
- **Advanced Retry System** — Configurable retry with exponential, linear, or constant backoff
- **Request/Response Interceptors** — Middleware pattern for transforming requests and responses
- **Request Deduplication** — Automatically deduplicate simultaneous identical requests
- **TypeScript Generics** — Type-safe response parsing with `webrequest.getJson<T>()`
- **Multi-Endpoint Fallback** — Fault tolerance via fallback URLs with retry strategies
- **Timeout Support** — Configurable request timeouts with AbortController
- **Cross-Runtime** — Works in browsers, Node.js, Deno, and Bun

Installation

```
pnpm install @push.rocks/webrequest
# or
npm install @push.rocks/webrequest
```

This package requires a modern JavaScript environment with ESM and TypeScript support.

Quick Start

Basic Fetch-Compatible Usage

```
import { webrequest } from '@push.rocks/webrequest';

// Use exactly like fetch()
const response = await webrequest('https://api.example.com/data');
const data = await response.json();

// With options (fetch-compatible + enhanced)
const response = await webrequest('https://api.example.com/data', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ key: 'value' }),
  timeout: 30000,
  retry: true,
});
```

JSON Convenience Methods

```
import { webrequest } from '@push.rocks/webrequest';

// GET JSON with type safety
interface User {
  id: number;
  name: string;
  email: string;
}
```

```
const user = await webrequest.getJson<User>('https://api.example.com/user/1');
// user is typed as User

// POST JSON
const result = await webrequest.postJson('https://api.example.com/users', {
  name: 'John Doe',
  email: 'john@example.com',
});

// PUT and DELETE
await webrequest.putJson(url, data);
await webrequest.deleteJson(url);
```

Cache Strategies

Network-First (Default)

Always fetch from network, fall back to cache on failure. Respects HTTP caching headers.

```
const data = await webrequest.getJson('https://api.example.com/data', {
  cacheStrategy: 'network-first',
});
```

Cache-First

Check cache first, only fetch from network if not cached or stale.

```
const data = await webrequest.getJson('https://api.example.com/data', {
  cacheStrategy: 'cache-first',
  cacheMaxAge: 60000, // 60 seconds
});
```

Stale-While-Revalidate

Return cached data immediately, update in background.

```
const data = await webrequest.getJson('https://api.example.com/data', {
  cacheStrategy: 'stale-while-revalidate',
});
```

Network-Only and Cache-Only

```
// Always fetch from network, never cache
const fresh = await webrequest.getJson(url, {
  cacheStrategy: 'network-only',
});

// Only use cache, never fetch from network
const cached = await webrequest.getJson(url, {
  cacheStrategy: 'cache-only',
});
```

HTTP Header-Based Caching

The library automatically respects HTTP caching headers:

```
// Server returns: Cache-Control: max-age=3600, ETag: "abc123"
const response = await webrequest('https://api.example.com/data', {
  cacheStrategy: 'network-first',
});

// Subsequent requests automatically send:
// If-None-Match: "abc123"
// Server returns 304 Not Modified — cache is used
```

Custom Cache Keys

```
const response = await webrequest('https://api.example.com/search?q=test', {
  cacheStrategy: 'cache-first',
  cacheKey: (request) => {
    const url = new URL(request.url);
    return `search:${url.searchParams.get('q')}`;
```

```
  },
});
```

Retry Strategies

Basic Retry

```
const response = await webrequest('https://api.example.com/data', {
  retry: true, // Uses defaults: 3 attempts, exponential backoff
});
```

Advanced Retry Configuration

```
const response = await webrequest('https://api.example.com/data', {
  retry: {
    maxAttempts: 5,
    backoff: 'exponential', // or 'linear', 'constant'
    initialDelay: 1000,     // 1 second
    maxDelay: 30000,        // 30 seconds
    retryOn: [408, 429, 500, 502, 503, 504],
    onRetry: (attempt, error, nextDelay) => {
      console.log(`Retry attempt ${attempt}, waiting ${nextDelay}ms`);
    },
  },
});
```

Multi-Endpoint Fallback

When the primary endpoint fails, automatically try fallback URLs:

```
const response = await webrequest('https://api1.example.com/data', {
  fallbackUrls: [
    'https://api2.example.com/data',
    'https://api3.example.com/data',
  ],
```

```
  retry: {
    maxAttempts: 3,
    backoff: 'exponential',
  },
});
```

Each URL is tried with the configured retry strategy. If all attempts for a URL fail with server errors, the next fallback URL is tried.

Request/Response Interceptors

Global Interceptors

```
import { webrequest } from '@push.rocks/webrequest';

// Add authentication to all requests
webrequest.addRequestInterceptor((request) => {
  const headers = new Headers(request.headers);
  headers.set('Authorization', `Bearer ${getToken()}`);
  return new Request(request, { headers });
});

// Log all responses
webrequest.addResponseInterceptor((response) => {
  console.log(`${response.status} ${response.url}`);
  return response;
});

// Handle errors globally
webrequest.addErrorInterceptor((error) => {
  console.error('Request failed:', error);
  return error;
});

// Clear all interceptors when needed
webrequest.clearInterceptors();
```

Per-Request Interceptors

```
const response = await webrequest('https://api.example.com/data', {
  interceptors: {
    request: [(req) => {
      console.log('Sending:', req.url);
      return req;
    }],
    response: [(res) => {
      console.log('Received:', res.status);
      return res;
    }],
  },
});
```

Request Deduplication

Automatically prevent duplicate simultaneous requests:

```
// Only one actual network request is made
const [res1, res2, res3] = await Promise.all([
  webrequest('https://api.example.com/data', { deduplicate: true }),
  webrequest('https://api.example.com/data', { deduplicate: true }),
  webrequest('https://api.example.com/data', { deduplicate: true }),
]);

// All three get the same response (cloned)
```

Deduplication works for GET and HEAD requests by matching URL + method. Non-GET/HEAD requests are not deduplicated since they may have different bodies.

WebrequestClient

For more control, use `WebrequestClient` to set default options that apply to all requests made through that client:

```
import { WebrequestClient } from '@push.rocks/webrequest';

const apiClient = new WebrequestClient({
  logging: true,
  timeout: 30000,
  cacheStrategy: 'network-first',
  retry: {
    maxAttempts: 3,
    backoff: 'exponential',
  },
});

// Add global interceptors to this client
apiClient.addRequestInterceptor((request) => {
  const headers = new Headers(request.headers);
  headers.set('X-API-Key', 'my-key');
  return new Request(request, { headers });
});

// All requests through this client use the configured defaults
const data = await apiClient.getJson('https://api.example.com/data');

// Standard fetch-compatible API
const response = await apiClient.request('https://api.example.com/data');

// Create a client from the webrequest function
const client = webrequest.createClient({ timeout: 5000 });
```

Advanced Features

Timeout

```
const response = await webrequest('https://api.example.com/data', {
  timeout: 5000, // 5 seconds — throws Error on timeout
});
```

Cache Management

```
// Clear all cached responses
await webrequest.clearCache();
```

API Reference

Main Function

```
webrequest(input: string | Request | URL, options?: IWebrequestOptions): Promise<Response>
```

Convenience Methods

```
webrequest.getJson<T>(url: string, options?: IWebrequestOptions): Promise<T>
webrequest.postJson<T>(url: string, body: any, options?: IWebrequestOptions): Promise<T>
webrequest.putJson<T>(url: string, body: any, options?: IWebrequestOptions): Promise<T>
webrequest.deleteJson<T>(url: string, options?: IWebrequestOptions): Promise<T>
```

Global Methods

```
webrequest.addRequestInterceptor(interceptor: TRequestInterceptor): void
webrequest.addResponseInterceptor(interceptor: TResponseInterceptor): void
webrequest.addErrorInterceptor(interceptor: TErrorInterceptor): void
webrequest.clearInterceptors(): void
webrequest.clearCache(): Promise<void>
webrequest.createClient(options?: Partial<IWebrequestOptions>): WebrequestClient
webrequest.getDefaultClient(): WebrequestClient
```

Options Interface

```
interface IWebrequestOptions extends Omit<RequestInit, 'cache'> {
  // Standard fetch options
```

```
  method?: string;
  headers?: HeadersInit;
  body?: BodyInit;

  // Caching
  cache?: 'default' | 'no-store' | 'reload' | 'no-cache' | 'force-cache' | 'only-if-cached';
  cacheStrategy?: 'network-first' | 'cache-first' | 'stale-while-revalidate' | 'network-only'
| 'cache-only';
  cacheMaxAge?: number;
  cacheKey?: string | ((request: Request) => string);
  revalidate?: boolean;

  // Retry & Fault Tolerance
  retry?: boolean | IRetryOptions;
  fallbackUrls?: string[];
  timeout?: number;

  // Interceptors
  interceptors?: {
    request?: TRequestInterceptor[];
    response?: TResponseInterceptor[];
  };

  // Deduplication
  deduplicate?: boolean;

  // Logging
  logging?: boolean;
}
```

Retry Options

```
interface IRetryOptions {
  maxAttempts?: number;        // Default: 3
  backoff?: 'exponential' | 'linear' | 'constant'; // Default: 'exponential'
  initialDelay?: number;       // Default: 1000 (ms)
  maxDelay?: number;           // Default: 30000 (ms)
  retryOn?: number[] | ((response: Response, error?: Error) => boolean);
```

```
  onRetry?: (attempt: number, error: Error, nextDelay: number) => void;
}
```

Examples

Complete Example with All Features

```
import { webrequest } from '@push.rocks/webrequest';

async function fetchUserData(userId: string) {
  interface User {
    id: string;
    name: string;
    email: string;
  }

  const user = await webrequest.getJson<User>(
    `https://api.example.com/users/${userId}`,
    {
      cacheStrategy: 'stale-while-revalidate',
      cacheMaxAge: 300000, // 5 minutes
      retry: {
        maxAttempts: 3,
        backoff: 'exponential',
        retryOn: [500, 502, 503, 504],
      },
      fallbackUrls: [
        `https://api-backup.example.com/users/${userId}`,
      ],
      timeout: 10000,
      deduplicate: true,
      interceptors: {
        request: [(req) => {
          console.log(`Fetching user ${userId}`);
          return req;
        }],
      },
```

```
    }
  );

  return user;
}
```

Building a Typed API Client

```
import { WebrequestClient } from '@push.rocks/webrequest';

interface User {
  id: string;
  name: string;
  email: string;
}

interface CreateUserData {
  name: string;
  email: string;
}

class ApiClient {
  private client: WebrequestClient;

  constructor(private baseUrl: string, private apiKey: string) {
    this.client = new WebrequestClient({
      timeout: 30000,
      cacheStrategy: 'network-first',
      retry: {
        maxAttempts: 3,
        backoff: 'exponential',
      },
    });

    this.client.addRequestInterceptor((request) => {
      const headers = new Headers(request.headers);
      headers.set('Authorization', `Bearer ${this.apiKey}`);
      headers.set('Content-Type', 'application/json');
```

```
      return new Request(request, { headers });
    });
  }

  async getUser(id: string): Promise<User> {
    return this.client.getJson<User>(`${this.baseUrl}/users/${id}`);
  }

  async createUser(data: CreateUserData): Promise<User> {
    return this.client.postJson<User>(`${this.baseUrl}/users`, data);
  }

  async updateUser(id: string, data: Partial<User>): Promise<User> {
    return this.client.putJson<User>(`${this.baseUrl}/users/${id}`, data);
  }

  async deleteUser(id: string): Promise<void> {
    await this.client.deleteJson(`${this.baseUrl}/users/${id}`);
  }
}

const api = new ApiClient('https://api.example.com', 'my-api-key');
const user = await api.getUser('123');
```

Migration from v3

Version 4.0 is a **complete rewrite** of `@push.rocks/webrequest`. The v3 API has been removed entirely.

Key Changes

v3	v4
`new WebRequest()`	`webrequest()` function or `new WebrequestClient()`
`client.getJson(url, true)`	`webrequest.getJson(url, { cacheStrategy: 'cache-first' })`
`client.requestMultiEndpoint([...urls])`	`webrequest(url, { fallbackUrls: [...] })`
`request(url, { timeoutMs: 30000 })`	`webrequest(url, { timeout: 30000 })`

Migration Examples

```
// v3 — Class-based
import { WebRequest } from '@push.rocks/webrequest';
const client = new WebRequest();
const response = await client.request('https://api.example.com/data', { method: 'GET' });

// v4 — Function-based (fetch-compatible)
import { webrequest } from '@push.rocks/webrequest';
const response = await webrequest('https://api.example.com/data');
const data = await response.json();

// v4 — Client-based (when you need defaults)
import { WebrequestClient } from '@push.rocks/webrequest';
const client = new WebrequestClient({ timeout: 30000 });
const data = await client.getJson('https://api.example.com/data');
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:13:32 UTC by foss.global Team
Updated 2026-03-28 12:20:17 UTC by foss.global Team