readme.md for @push.rocks/smartenv

> □□ **Universal JavaScript Runtime Detection** - One library for Node.js, Deno, Bun, and Browser

A powerful TypeScript library that provides comprehensive runtime environment detection and safe module loading across **all major JavaScript runtimes**. Write once, run everywhere with confidence.

Why smartenv?

Modern JavaScript runs in many environments - Node.js, Deno, Bun, and browsers. Writing isomorphic code that works everywhere is challenging. **smartenv** solves this by providing:

- □ **Accurate runtime detection** - Distinguishes Node.js from Deno, Bun, and browsers without false positives
- □ **Smart module loading** - Load the right modules for each runtime automatically
- □ **Platform detection** - Detect macOS, Linux, Windows, and CI environments
- □ **Zero dependencies** (except @push.rocks/smartpromise)
- □ **Full TypeScript support** with complete type definitions
- □ **Battle-tested** - Comprehensive test suite across all runtimes

Install

```
pnpm install @push.rocks/smartenv --save
```

```
npm install @push.rocks/smartenv --save
```

Quick Start

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

const env = new Smartenv();

// Detect the runtime
console.log(env.runtimeEnv); // 'node' | 'deno' | 'bun' | 'browser'

// Load modules safely for your runtime
const pathModule = await env.getSafeModuleFor('server', 'path');
if (pathModule) {
  console.log('Path module loaded!');
}

// Check specific runtimes
if (env.isNode) {
  console.log(`Running on Node.js ${env.nodeVersion}`);
} else if (env.isDeno) {
  console.log(`Running on Deno ${env.denoVersion}`);
} else if (env.isBun) {
  console.log(`Running on Bun ${env.bunVersion}`);
} else {
  console.log(`Running in ${env.userAgent}`);
}
```

Core Features

□□ Multi-Runtime Detection

Accurately detects all major JavaScript runtimes using proper detection order to avoid false positives:

```
const env = new Smartenv();

// Runtime type - returns one of: 'node' | 'deno' | 'bun' | 'browser'
console.log(env.runtimeEnv);

// Boolean checks for each runtime
```

```
env.isNode;     // true only in Node.js
env.isDeno;     // true only in Deno
env.isBun;      // true only in Bun
env.isBrowser;  // true only in browser
```

Why detection order matters: Deno and Bun provide `process` objects for Node.js compatibility. smartenv checks for `Deno` and `Bun` globals first, then `process.versions.node`, ensuring accurate detection.

□□ Smart Module Loading

The new `getSafeModuleFor()` API lets you target specific runtimes or groups:

```
// Load for a specific runtime
const fsNode = await env.getSafeModuleFor('node', 'fs');

// Load for Deno (requires 'node:' prefix for Node.js built-ins)
const fsDeno = await env.getSafeModuleFor('deno', 'node:path');

// Load for all server-side runtimes (Node.js + Deno + Bun)
const pathModule = await env.getSafeModuleFor('server', 'path');

// Load for multiple specific runtimes
const crypto = await env.getSafeModuleFor(['node', 'bun'], 'crypto');

// Browser module loading
const jQuery = await env.getSafeModuleFor(
  'browser',
  'https://code.jquery.com/jquery-3.6.0.min.js',
  () => window.jQuery
);
```

Target options:

- `'node'` - Node.js only
- `'deno'` - Deno only
- `'bun'` - Bun only
- `'browser'` - Browser only
- `'server'` - Shorthand for `['node', 'deno', 'bun']`
- `['node', 'deno']` - Array of specific runtimes

If the current runtime doesn't match the target, the method returns `undefined` and logs a warning.

□□ Platform Detection

Detect operating systems in server-side runtimes (Node.js, Deno, Bun):

```
const env = new Smartenv();

// Async OS detection
const isMac = await env.isMacAsync();       // macOS
const isLinux = await env.isLinuxAsync();   // Linux
const isWindows = await env.isWindowsAsync(); // Windows

if (isMac) {
  console.log('Running on macOS');
}
```

□□ Version Information

Get version strings for each runtime:

```
const env = new Smartenv();

// Returns version string or 'undefined' if not in that runtime
env.nodeVersion;  // e.g., 'v20.10.0'
env.denoVersion;  // e.g., '1.40.0'
env.bunVersion;   // e.g., '1.0.20'
env.userAgent;    // Browser user agent string
```

□□ CI Detection

Detect if running in a continuous integration environment:

```
const env = new Smartenv();

if (env.isCI) {
  console.log('Running in CI environment');
```

```
  // Enable extended test suite, different build config, etc.
}
```

API Reference

Runtime Detection Properties

`runtimeEnv: TRuntimeType`

Returns the detected runtime as a string.

Type: `'node' | 'deno' | 'bun' | 'browser'`

```
const env = new Smartenv();
console.log(env.runtimeEnv); // 'node', 'deno', 'bun', or 'browser'
```

`isNode: boolean`

`true` if running in Node.js (specifically checks for `process.versions.node`).

```
if (env.isNode) {
  console.log('Node.js environment');
}
```

`isDeno: boolean`

`true` if running in Deno (checks for `Deno` global).

```
if (env.isDeno) {
  console.log('Deno environment');
}
```

`isBun: boolean`

`true` if running in Bun (checks for `Bun` global).

```
if (env.isBun) {
  console.log('Bun environment');
}
```

`isBrowser: boolean`

`true` if running in a browser environment.

```
if (env.isBrowser) {
  console.log('Browser environment');
}
```

`isCI: boolean`

`true` if running in a CI environment (checks `process.env.CI` in server runtimes).

```
if (env.isCI) {
  // CI-specific logic
}
```

Version Properties

`nodeVersion: string`

Node.js version string. Returns `'undefined'` if not in Node.js.

```
console.log(env.nodeVersion); // 'v20.10.0'
```

`denoVersion: string`

Deno version string. Returns `'undefined'` if not in Deno.

```
console.log(env.denoVersion); // '1.40.0'
```

`bunVersion: string`

Bun version string. Returns `'undefined'` if not in Bun.

```
console.log(env.bunVersion); // '1.0.20'
```

`userAgent: string`

Browser user agent string. Returns `'undefined'` if not in browser.

```
console.log(env.userAgent); // 'Mozilla/5.0...'
```

Platform Detection Methods

`isMacAsync(): Promise<boolean>`

Asynchronously checks if running on macOS. Works in Node.js, Deno, and Bun.

```
const isMac = await env.isMacAsync();
if (isMac) {
  console.log('Running on macOS');
}
```

`isLinuxAsync(): Promise<boolean>`

Asynchronously checks if running on Linux. Works in Node.js, Deno, and Bun.

```
const isLinux = await env.isLinuxAsync();
if (isLinux) {
  console.log('Running on Linux');
}
```

`isWindowsAsync(): Promise<boolean>`

Asynchronously checks if running on Windows. Works in Node.js, Deno, and Bun.

```
const isWindows = await env.isWindowsAsync();
if (isWindows) {
  console.log('Running on Windows');
}
```

Module Loading Methods

`getSafeModuleFor<T>(target, moduleNameOrUrl, getFunction?): Promise<T | undefined>`

The recommended way to load modules - supports runtime targeting with flexible options.

Parameters:

- `target: TRuntimeTarget | TRuntimeTarget[]` - Runtime(s) to load for
- `moduleNameOrUrl: string` - Module name (server runtimes) or URL (browser)
- `getFunction?: () => any` - Function to retrieve module (required for browser)

Returns: `Promise<T | undefined>` - Loaded module or undefined if runtime doesn't match

Examples:

```
// Node.js only
const fs = await env.getSafeModuleFor('node', 'fs');

// Deno only (note: use 'node:' prefix for Node.js built-ins)
const path = await env.getSafeModuleFor('deno', 'node:path');

// All server runtimes
const crypto = await env.getSafeModuleFor('server', 'crypto');

// Multiple specific runtimes
const util = await env.getSafeModuleFor(['node', 'bun'], 'util');

// Browser
const lib = await env.getSafeModuleFor(
  'browser',
  'https://cdn.example.com/lib.js',
  () => window.MyLib
);
```

`getSafeNodeModule<T>(moduleName, runAfterFunc?): Promise<T>`

Legacy method - Loads modules in server-side runtimes (Node.js, Deno, Bun).

```
const fs = await env.getSafeNodeModule('fs');

// With post-load callback
const express = await env.getSafeNodeModule('express', async (mod) => {
  console.log('Express loaded');
});
```

`getSafeWebModule(url, getFunction): Promise<any>`

Legacy method - Loads web modules via script tag in browser. Prevents duplicate loading.

```
const jQuery = await env.getSafeWebModule(
  'https://code.jquery.com/jquery-3.6.0.min.js',
```

```
  () => window.jQuery
);
```

`getEnvAwareModule(options): Promise<any>`

Legacy method - Loads environment-appropriate modules.

```
const module = await env.getEnvAwareModule({
  nodeModuleName: 'node-fetch',
  webUrlArg: 'https://unpkg.com/whatwg-fetch@3.6.2/dist/fetch.umd.js',
  getFunction: () => window.fetch
});
```

Utility Methods

`printEnv(): Promise<void>`

Prints environment information to console for debugging.

```
await env.printEnv();
// Node.js: "running on NODE" + version
// Deno: "running on DENO" + version
// Bun: "running on BUN" + version
// Browser: "running on BROWSER" + user agent
```

Real-World Examples

□□ Isomorphic Cryptography

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

const env = new Smartenv();

// Load crypto for any server runtime
const crypto = await env.getSafeModuleFor('server', 'crypto');

if (crypto) {
```

```
  const hash = crypto.createHash('sha256');
  hash.update('hello world');
  console.log(hash.digest('hex'));
} else if (env.isBrowser) {
  // Use Web Crypto API
  const encoder = new TextEncoder();
  const data = encoder.encode('hello world');
  const hashBuffer = await crypto.subtle.digest('SHA-256', data);
  console.log(Array.from(new Uint8Array(hashBuffer))
    .map(b => b.toString(16).padStart(2, '0'))
    .join(''));
}
```

□□ Cross-Runtime File System

```
const env = new Smartenv();

async function readConfig() {
  if (env.isNode) {
    const fs = await env.getSafeModuleFor('node', 'fs/promises');
    return JSON.parse(await fs.readFile('config.json', 'utf-8'));
  } else if (env.isDeno) {
    const content = await Deno.readTextFile('config.json');
    return JSON.parse(content);
  } else if (env.isBun) {
    const file = Bun.file('config.json');
    return await file.json();
  } else {
    // Browser: fetch from server
    const response = await fetch('/config.json');
    return response.json();
  }
}
```

□□ Development vs Production

```
const env = new Smartenv();

async function setupEnvironment() {
  // Different behavior in CI
  if (env.isCI) {
    console.log('CI Environment detected');
    // Skip interactive prompts, use default values
    return { mode: 'ci', verbose: true };
  }

  // Platform-specific paths
  if (await env.isMacAsync()) {
    return { configPath: '/Users/username/.config' };
  } else if (await env.isLinuxAsync()) {
    return { configPath: '/home/username/.config' };
  } else if (await env.isWindowsAsync()) {
    return { configPath: 'C:\\Users\\username\\AppData\\Local' };
  }
}
```

□□ Conditional Analytics Loading

```
const env = new Smartenv();

async function initializeAnalytics() {
  if (!env.isBrowser) {
    console.log('Analytics skipped - not in browser');
    return;
  }

  // Only load analytics in browser
  const analytics = await env.getSafeModuleFor(
    'browser',
    'https://www.googletagmanager.com/gtag/js?id=GA_MEASUREMENT_ID',
    () => window.gtag
  );

  if (analytics) {
```

```
    analytics('config', 'GA_MEASUREMENT_ID');
  }
}
```

□□ Runtime-Specific Testing

```
const env = new Smartenv();

async function runTests() {
  console.log(`Testing in ${env.runtimeEnv}`);

  // Load test utilities for current runtime
  const testLib = await env.getSafeModuleFor(
    ['node', 'deno', 'bun'],
    '@git.zone/tstest'
  );

  if (testLib) {
    // Run server-side tests
    await runServerTests(testLib);
  } else {
    // Run browser tests
    await runBrowserTests();
  }
}
```

TypeScript Support

smartenv is written in TypeScript and provides complete type definitions:

```
import {
  Smartenv,
  TRuntimeType,     // 'node' | 'deno' | 'bun' | 'browser'
  TRuntimeTarget,   // TRuntimeType | 'server'
  IEnvObject
} from '@push.rocks/smartenv';
```

```
const env: Smartenv = new Smartenv();
const runtime: TRuntimeType = env.runtimeEnv;

// Type-safe module loading
const fs = await env.getSafeModuleFor<typeof import('fs')>('node', 'fs');
if (fs) {
  fs.readFileSync('./file.txt', 'utf-8'); // Full type support
}
```

How Runtime Detection Works

smartenv uses a careful detection order to avoid false positives:

1. **Check for Deno** - `globalThis.Deno?.version` (Deno has `process` for compatibility)
2. **Check for Bun** - `globalThis.Bun?.version` (Bun also has `process`)
3. **Check for Node.js** - `globalThis.process?.versions?.node` (must be specific)
4. **Check for Browser** - `globalThis.window && globalThis.document` (fallback)

This order is critical because Deno and Bun provide `process` objects for Node.js compatibility, which would cause false Node.js detection if checked first.

Migration from 4.x to 5.x

Breaking Changes:

1. **`runtimeEnv` return type changed:**
   - **Before:** `'node' | 'browser'`
   - **After:** `'node' | 'deno' | 'bun' | 'browser'`
2. **`isNode` is now more specific:**
   - **Before:** Returns `true` for Node.js, Deno, and Bun
   - **After:** Returns `true` only for Node.js

Migration guide:

```
// Before (4.x)
if (env.isNode) {
  // This ran in Node.js, Deno, and Bun
}
```

```
// After (5.x) - Option 1: Check all server runtimes
if (env.isNode || env.isDeno || env.isBun) {
  // Works in Node.js, Deno, and Bun
}

// After (5.x) - Option 2: Use the new module loading API
const module = await env.getSafeModuleFor('server', 'path');
if (module) {
  // Module loaded successfully in any server runtime
}
```

Performance

- ⚡ **Lightweight** - Minimal overhead with lazy evaluation
- □□ **Fast detection** - Simple boolean checks, no heavy operations
- □□ **Cached results** - Detection runs once, results are cached
- □□ **Small bundle** - ~5KB minified, tree-shakeable

Browser Compatibility

Tested and working in:

- □ Chrome/Chromium (latest)
- □ Firefox (latest)
- □ Safari (latest)
- □ Edge (latest)

Node.js Compatibility

- □ Node.js 18.x
- □ Node.js 20.x (LTS)
- □ Node.js 22.x

Deno Compatibility

- Deno 1.40+

Note: When using Deno, use the `node:` prefix for Node.js built-in modules:

```
const path = await env.getSafeModuleFor('deno', 'node:path');
```

Bun Compatibility

- Bun 1.0+

License and Legal Information

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

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 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, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

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

For any legal inquiries or if you require 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 11:10:34 UTC by foss.global Team

Updated 2026-03-28 12:17:21 UTC by foss.global Team