@push.rocks/smartexpect

A testing library to manage expectations in code, offering both synchronous and asynchronous assertion methods.

- readme.md for @push.rocks/smartexpect
- changelog.md for @push.rocks/smartexpect

readme.md for @push.rocks/smartexpect

Manage expectations in code with precise, readable assertions

Install

To install `@push.rocks/smartexpect`, use the following command in your terminal:

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

This will add `@push.rocks/smartexpect` to your project's dependencies. Make sure you're inside your project directory before running this command.

Why SmartExpect?

SmartExpect is designed to be a minimal, promise-first assertion library with clear, descriptive messaging and easy extensibility:

- **Zero-config asynchronous support**: chain `.resolves` / `.rejects` directly off `expect()`, add timeouts with `.withTimeout(ms)`, and get clear errors if you mix sync matchers with non-Promises.
- **Lean, modular footprint**: only depends on `fast-deep-equal`, `@push.rocks/smartpromise`, and `@push.rocks/smartdelay`; pure ESM, tree-shakable, works in Node & browser.
- **Better out-of-the-box messaging**: automatic `.not` inversion (e.g. “Expected 5 not to be greater than 3”) and unified “Expected… / Received…” JSON diffs for object/array mismatches.
- **Rich built-in matchers**: numbers (`toBeNaN`, `toBeWithinRange`), objects (`toHaveKeys`, `toHaveOwnKeys`, shorthand `toHaveOwnProperty`), strings/arrays (`toBeEmpty`, `toInclude`, `toHaveLength`), functions (`toThrowErrorMatching`, `toThrowErrorWithMessage`), dates, and more.
- **Plugin-style extensibility**: add custom matchers with `expect.extend({ myMatcher })` without monkey-patching.

- **First-class TypeScript support**: full `.d.ts` declarations, generic types for sync vs async chains, and autocomplete in editors.

Usage

`@push.rocks/smartexpect` is a TypeScript library designed to manage expectations in your code effectively, improving testing readability and maintainability. Below are various scenarios showcasing how to use this library effectively across both synchronous and asynchronous code paths.

Getting Started

First, import `@push.rocks/smartexpect` into your TypeScript file:

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

Synchronous Expectations

You can employ `expect` to create synchronous assertions:

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

// Type assertions
expect('hello').toBeTypeofString();
expect(42).toBeTypeofNumber();
expect(true).toBeTypeofBoolean();
expect(() => {}).toBeTypeOf('function');
expect({}).toBeTypeOf('object');

// Negated assertions
expect(1).not.toBeTypeofString();
expect('string').not.toBeTypeofNumber();

// Equality assertion
expect('hithere').toEqual('hithere');

// Deep equality assertion
```

```
expect({ key: 'value' }).toEqual({ key: 'value' });

// Regular expression matching
expect('hithere').toMatch(/hi/);
```

Asynchronous Expectations

For asynchronous code, use the same `expect` function with the `.resolves` or `.rejects` modifier:

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

const asyncStringFetcher = async (): Promise<string> => {
  return 'async string';
};

const asyncTest = async () => {
  // Add a timeout to prevent hanging tests
  await expect(asyncStringFetcher()).resolves.withTimeout(5000).type.toBeTypeofString();
  await expect(asyncStringFetcher()).resolves.toEqual('async string');
};

asyncTest();
```

Navigating Complex Objects

You can navigate complex objects using the `property()` and `arrayItem()` methods:

```
const complexObject = {
  users: [
    { id: 1, name: 'Alice', permissions: { admin: true } },
    { id: 2, name: 'Bob', permissions: { admin: false } }
  ]
};

// Navigate to a nested property
expect(complexObject)
  .property('users')
  .arrayItem(0)
```

```
  .property('name')
  .toEqual('Alice');

// Check nested permission
expect(complexObject)
  .property('users')
  .arrayItem(0)
  .property('permissions')
  .property('admin')
  .toBeTrue();
```

Advanced Assertions

Properties and Deep Properties

Assert the existence of properties and their values:

```
const testObject = { level1: { level2: 'value' } };

// Property existence
expect(testObject).toHaveProperty('level1');

// Property with specific value
expect(testObject).toHaveProperty('level1.level2', 'value');

// Deep Property existence
expect(testObject).toHaveDeepProperty(['level1', 'level2']);
```

Conditions and Comparisons

Perform more intricate assertions:

```
// Numeric comparisons
expect(5).toBeGreaterThan(3);
expect(3).toBeLessThan(5);
expect(5).toBeGreaterThanOrEqual(5);
expect(5).toBeLessThanOrEqual(5);
expect(0.1 + 0.2).toBeCloseTo(0.3, 10); // Floating point comparison with precision
```

```
// Truthiness checks
expect(true).toBeTrue();
expect(false).toBeFalse();
expect('non-empty').toBeTruthy();
expect(0).toBeFalsy();

// Null/Undefined checks
expect(null).toBeNull();
expect(undefined).toBeUndefined();
expect(null).toBeNullOrUndefined();

// Custom conditions
expect(7).customAssertion(value => value % 2 === 1, 'Value is not odd');
```

Arrays and Collections

Work seamlessly with arrays and collections:

```
const testArray = [1, 2, 3];

// Array checks
expect(testArray).toBeArray();
expect(testArray).toHaveLength(3);
expect(testArray).toContain(2);
expect(testArray).toContainAll([1, 3]);
expect(testArray).toExclude(4);
expect([]).toBeEmptyArray();
expect(testArray).toHaveLengthGreaterThan(2);
expect(testArray).toHaveLengthLessThan(4);

// Deep equality in arrays
expect([{ id: 1 }, { id: 2 }]).toContainEqual({ id: 1 });
```

Strings

String-specific checks:

```
expect('hello world').toStartWith('hello');
expect('hello world').toEndWith('world');
expect('hello world').toInclude('lo wo');
```

```
expect('options').toBeOneOf(['choices', 'options', 'alternatives']);
```

Functions and Exceptions

Test function behavior and exceptions:

```
const throwingFn = () => { throw new Error('test error'); };
expect(throwingFn).toThrow();
expect(throwingFn).toThrow(Error);

const safeFn = () => 'result';
expect(safeFn).not.toThrow();
```

Date Assertions

Work with dates:

```
const now = new Date();
const past = new Date(Date.now() - 10000);
const future = new Date(Date.now() + 10000);

expect(now).toBeDate();
expect(now).toBeAfterDate(past);
expect(now).toBeBeforeDate(future);
```

Debugging Assertions

The `log()` method is useful for debugging complex assertions:

```
expect(complexObject)
  .property('users')
  .log() // Logs the current value in the assertion chain
  .arrayItem(0)
  .log() // Logs the first user
  .property('permissions')
  .log() // Logs the permissions object
  .property('admin')
  .toBeTrue();
```

Customizing Error Messages

You can provide custom error messages for more meaningful test failures:

```
expect(user.age)
  .setFailMessage('User age must be at least 18 for adult content')
  .toBeGreaterThanOrEqual(18);
```

Custom Matchers

You can define your own matchers via `expect.extend()`:

```
expect.extend({
  toBeOdd(received: number) {
    const pass = received % 2 === 1;
    return {
      pass,
      message: () =>
        `Expected ${received} ${pass ? 'not ' : ''}to be odd`,
    };
  },
});

// Then use your custom matcher in tests:
expect(3).toBeOdd();
expect(4).not.toBeOdd();
```

- Matcher functions receive the value under test (`received`) plus any arguments.
- Must return an object with `pass` (boolean) and `message` (string or function) for failure messages.

Full Matcher Reference

Below is a comprehensive list of all matchers and utility functions available in `@push.rocks/smartexpect`.

Modifiers and Utilities

- `.not` Negates the next matcher in the chain.

- `.resolves` Switches to async mode, expecting the promise to resolve.
- `.rejects` Switches to async mode, expecting the promise to reject.
- `.withTimeout(ms)` Sets a timeout (in milliseconds) for async assertions.
- `.timeout(ms)` (deprecated) Alias for `.withTimeout(ms)`.
- `.property(name)` Drill into a property of an object.
- `.arrayItem(index)` Drill into an array element by index.
- `.log()` Logs the current value and assertion path for debugging.
- `.setFailMessage(message)` Override the failure message for the current assertion.
- `.setSuccessMessage(message)` Override the success message for the current assertion.
- `.customAssertion(fn, message)` Execute a custom assertion function with a message.
- `expect.extend(matchers)` Register custom matchers globally.
- `expect.any(constructor)` Matcher for values that are instances of the given constructor.
- `expect.anything()` Matcher for any defined value (not null or undefined).

Basic Matchers

- `.toEqual(expected)` Deep (or strict for primitives) equality.
- `.toBeTrue()` Value is strictly `true`.
- `.toBeFalse()` Value is strictly `false`.
- `.toBeTruthy()` Value is truthy.
- `.toBeFalsy()` Value is falsy.
- `.toBeNull()` Value is `null`.
- `.toBeUndefined()` Value is `undefined`.
- `.toBeNullOrUndefined()` Value is `null` or `undefined`.
- `.toBeDefined()` Value is not `undefined`.

Number Matchers

- `.toBeGreaterThan(value)`
- `.toBeLessThan(value)`
- `.toBeGreaterThanOrEqual(value)`
- `.toBeLessThanOrEqual(value)`
- `.toBeCloseTo(value, precision?)`
- `.toEqual(value)` Strict equality for numbers.
- `.toBeNaN()`
- `.toBeFinite()`
- `.toBeWithinRange(min, max)`

String Matchers

- `.toStartWith(prefix)`
- `.toEndWith(suffix)`
- `.toInclude(substring)`
- `.toMatch(regex)`
- `.toBeOneOf(arrayOfValues)`
- `.toHaveLength(length)`

- `.toBeEmpty()` Alias for empty string.

Array Matchers

- `.toBeArray()`
- `.toHaveLength(length)`
- `.toContain(value)`
- `.toContainEqual(value)`
- `.toContainAll(arrayOfValues)`
- `.toExclude(value)`
- `.toBeEmptyArray()`
- `.toBeEmpty()` Alias for empty array.
- `.toHaveLengthGreaterThan(length)`
- `.toHaveLengthLessThan(length)`

Object Matchers

- `.toEqual(expected)` Deep equality for objects.
- `.toMatchObject(partialObject)` Partial deep matching (supports `expect.any` and `expect.anything`).
- `.toBeInstanceOf(constructor)`
- `.toHaveProperty(propertyName, value?)`
- `.toHaveDeepProperty(pathArray)`
- `.toHaveOwnProperty(propertyName, value?)`
- `.toBeNull()`
- `.toBeUndefined()`
- `.toBeNullOrUndefined()`

Function Matchers

- `.toThrow(expectedError?)`
- `.toThrowErrorMatching(regex)`
- `.toThrowErrorWithMessage(message)`

Date Matchers

- `.toBeDate()`
- `.toBeBeforeDate(date)`
- `.toBeAfterDate(date)`

Type Matchers

- `.toBeTypeofString()`
- `.toBeTypeofNumber()`
- `.toBeTypeofBoolean()`

- `.toBeTypeOf(typeName)`

Best Practices

- **Human-readable assertions**: The fluent API is designed to create tests that read like natural language sentences.
- **Precise error messages**: When tests fail, the error messages provide detailed information about what went wrong, including expected vs. actual values.
- **Property path navigation**: Use the property path methods to navigate complex objects without creating temporary variables.
- **Comprehensive testing**: Take advantage of the wide range of assertion methods to test various aspects of your code.
- **Debugging with log()**: Use the `log()` method to see intermediate values in the assertion chain during test development.

License and Legal Information

Trademarks

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.

changelog.md for @push.rocks/smartexpect

2025-05-23 - 2.5.0 - feat(Assertion)

Add missing alias methods for length and emptiness checks and update documentation

- Updated readme.hints.md with detailed project hints, feature overview, and common patterns
- Added direct alias methods in smartexpect.classes.assertion.ts for string/array length checks (toHaveLength, toBeEmpty)
- Introduced additional delegations for numeric and array namespace methods (toHaveLengthGreaterThan, toHaveLengthLessThan)
- Enhanced function namespace with direct matchers for error checking (toThrowErrorMatching, toThrowErrorWithMessage)

2025-05-01 - 2.4.2 - fix(cleanup)

Remove unused scratch files

- Deleted scratch-alias.js, scratch-alias2.js, scratch-alias3.js, scratch-alias4.js, scratch-alias5.js, and scratch.js
- Clean up temporary alias and scratch test files

2025-04-30 - 2.4.1 - fix(Assertion)

Improve toHaveProperty alias by forwarding arguments correctly for intuitive object property assertions

- Updated the toHaveProperty method in the Assertion class to check the number of arguments and call the appropriate object matcher
- Added several scratch alias files to demonstrate and test the alias usage
- Enhanced test cases in test/propertypath to cover alias behavior

2025-04-30 - 2.4.0 - feat(object)

add toHaveOwnProperty method and improve property-path matching in object assertions

- Added 'toHaveOwnProperty' as a direct method on Assertion to check for own properties
- Enhanced property path evaluation in 'toHaveProperty' to handle nested keys more robustly
- Renamed test file to maintain consistent naming for expect.any tests
- Introduced scratch.js for manual testing and debugging of property matchers

2025-04-30 - 2.3.3 - fix(tests)

Fix test file naming inconsistencies

- Rename 'test/test.diffOutput.ts' to 'test/test.diffoutput.ts' to standardize filename casing
- Rename 'test/test.propertyPath.ts' to 'test/test.propertypath.ts' for consistent file naming

2025-04-30 - 2.3.2 - fix(object)

Update toHaveProperty matcher to support nested property paths using dot notation

- Changed toHaveProperty implementation to split property strings on '.' and traverse nested objects
- Fixed value comparison for nested properties by comparing the final drilled value instead of direct property access
- Added tests for nested property access in test/propertyPath.ts

2025-04-30 - 2.3.1 - fix(readme)

Improve README documentation with detailed 'Why SmartExpect' benefits section

- Added detailed 'Why SmartExpect' section outlining zero-config async support, modular footprint, enhanced messaging, rich built-in matchers, plugin extensibility, and TypeScript support
- Clarified installation instructions regarding dependency setup
- Updated changelog and commitinfo version to 2.3.1

2025-04-30 - 2.3.1 - fix(readme)

Improve README documentation with detailed 'Why SmartExpect' benefits section

- Added a detailed section outlining zero-config async support, modular footprint, enhanced messaging, rich built-in matchers, plugin extensibility, and TypeScript support
- Clarified installation instructions regarding dependency setup

2025-04-30 - 2.3.0 - feat(object-matchers)

Add object key matchers: toHaveKeys and toHaveOwnKeys; remove obsolete roadmap plan file

- Implemented toHaveKeys matcher to check for both own and inherited properties
- Implemented toHaveOwnKeys matcher to check for own properties only
- Added tests for key matchers in test/test.keys.ts
- Removed readme.plan.md as it is now obsolete

2025-04-29 - 2.2.2 - fix(license-files)

Remove legacy license file and add license.md to update file naming.

- Removed the old 'license' file.
- Added 'license.md' with updated file structure.

2025-04-29 - 2.2.1 - fix(readme)

Update usage examples and full matcher reference in README

- Removed deprecated 'expectAsync' from import example
- Fixed formatting in documentation
- Added comprehensive full matcher reference section

2025-04-29 - 2.2.0 - feat(generics)

Improve assertion and matcher type definitions by adding execution mode generics for better async/sync support

- Updated ts/index.ts to import and use TExecutionType in the expect function
- Modified Assertion class to use a generic execution mode (M) for improved type inference
- Revised all matcher namespaces (array, boolean, date, function, number, object, string, type) to accept the new generic parameter
- Enhanced async/sync distinction for assertion methods like resolves and rejects

2025-04-29 - 2.1.2 - fix(ts/index.ts)

Remove deprecated expectAsync function and advise using .resolves/.rejects on expect for async assertions

- Deleted the redundant expectAsync export in ts/index.ts
- Users should now call expect(...).resolves or expect(...).rejects for asynchronous assertions

2025-04-29 - 2.1.1 - fix(Assertion)

Improve chainability by fixing return types in assertion methods

- Update runCheck method to explicitly return the correct chainable type for both async and sync assertions
- Ensure customAssertion propagates the chainable Assertion instance
- Refactor internal promise handling for clarity and consistency

2025-04-28 - 2.1.0 - feat(core)

Add new matchers and improve negation messaging

- Added expect.any() and expect.anything() matchers for enhanced object pattern matching
- Introduced new number matchers: toBeNaN(), toBeFinite(), and toBeWithinRange()
- Implemented alias toBeEmpty() for both string and array matchers
- Enhanced function matchers with toThrowErrorMatching() and toThrowErrorWithMessage()
- Improved negation messaging to provide clearer failure messages (e.g. 'Expected 5 not to be greater than 3')
- Enhanced object assertions with a toHaveOwnProperty() shorthand that outputs unified diff-style messages

2025-04-28 - 2.0.1 - fix(assertion-matchers)

Refactor matcher implementations to consistently use customAssertion for improved consistency and clarity.

- Updated ArrayMatchers, BooleanMatchers, DateMatchers, FunctionMatchers, NumberMatchers, ObjectMatchers, StringMatchers, and TypeMatchers to use customAssertion directly.
- Aligned Assertion class aliases to delegate to the namespaced matchers with the new customAssertion pattern.

2025-04-28 - 2.0.0 - BREAKING CHANGE(docs)

Update documentation and examples to unify async and sync assertions, add custom matcher guides, and update package configuration

- Added packageManager field in package.json
- Revised documentation in readme.md to use .resolves/.rejects instead of expectAsync
- Included detailed examples for custom matchers and updated API usage
- Added readme.plan.md outlining the future roadmap
- Updated tests to import the built library from dist_ts

2025-03-04 - 1.6.1 - fix(build)

Corrected package.json and workflow dependencies and resolved formatting issues in tests.

- Fixed incorrect global npm package reference for tsdoc installation in workflow file.
- Updated dependencies in package.json for consistency in package naming.
- Resolved inconsistent formatting and spacing in test files.

2025-03-04 - 1.6.0 - feat(assertion)

Enhanced the assertion error messaging and added new test cases.

- Improved error messages by incorporating path and value/placeholders in assertions.
- Added detailed testing of new assertion functionalities.
- Additional test cases for comprehensive coverage of new features.

2025-03-04 - 1.5.0 - feat(Assertion)

Add toBeTypeOf assertion method

- Introduced a new assertion method `toBeTypeOf` allowing checks for expected data types.
- Updated devDependencies and dependencies to their latest versions.

2024-12-30 - 1.4.0 - feat(Assertion)

Add log method to Assertion class

- Introduced a log method in the Assertion class to output assertion context.

2024-12-30 - 1.3.0 - feat(Assertion)

Refactor Assertion class for better error handling and code clarity

- Improved method runCheck to better handle async and sync execution
- Enhanced getObjectToTestReference to handle undefined or null values gracefully
- Refactored error message logic for clarity and added more descriptive fail messages
- Added arrayItem method for better handling of array index access
- Improved structure by integrating consistent error handling in assertion methods

2024-08-24 - 1.2.1 - fix(Assertion)

Refactor methods for setting failure and success messages

- Renamed 'withFailMessage' to 'setFailMessage' for better readability and consistency.
- Renamed 'withSuccessMessage' to 'setSuccessMessage' to align with the naming convention.

2024-08-24 - 1.2.0 - feat(assertions)

Add custom fail and success messages for assertions

- Implemented withFailMessage method in Assertion class to customize fail messages
- Implemented withSuccessMessage method in Assertion class to customize success messages
- Enhanced error messages to use custom fail messages when provided

2024-08-17 - 1.1.0 - feat(assertion)

Add toBeDefined assertion method

- Added the toBeDefined method to the Assertion class for checking if an object is defined

2024-05-29 - 1.0.21 - General Updates

General updates and maintenance.

- Updated description
- Updated tsconfig
- Updated npmextra.json: githost

2023-08-12 - 1.0.20 to 1.0.21 - General Fixes

General fixes and update.

- Fixed core updates (multiple instances)
- 1.0.21 release

2023-07-10 - 1.0.15 - Organization Update

- Switched to new org scheme

2023-06-22 - 1.0.14 to 1.0.19 - General Updates

General fixes and updates.

- Fixed core updates (multiple instances)
- 1.0.18 to 1.0.16 releases

2022-02-02 - 1.0.8 to 1.0.13 - General Fixes

General fixes and update.

- Fixed core updates (multiple instances)
- 1.0.14 release

2022-01-20 - 1.0.1 to 1.0.7 - Initial Releases

Initial core updates and fixes.

- Fixed core updates (multiple instances)
- 1.0.7 release