privsim / **mcp-test-runner** Public

Code | Issues 1 | Pull requests 6 | Actions | Projects | Security and quality | Insights

main | 7 Branches | 0 Tags | Go to file | Go to file | Code

privsim Merge pull request #2 from punkpeye/patch-1 — 83c84ed · 6 months ago

.github	after_next	last year
src	after_next	last year
test	after_next	last year
.actrc	changed design	last year
.gitignore	after_next	last year
.node-version	init	last year
CHANGELOG.md	after_next	last year
LICENSE	init	last year
README.md	Merge pull request #2 from punkpeye/p…	6 months ago
README_flutter_syntax.md	after_next	last year
cleanup.sh	after_next	last year
jest.config.mjs	changed design	last year
package-lock.json	changed design	last year
package.json	after_next	last year
run_all_tests.sh	after_next	last year
tsconfig.json	changed design	last year

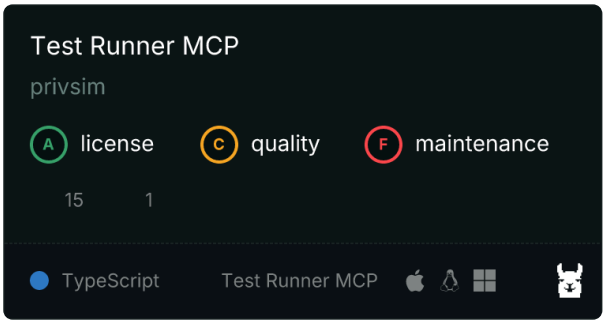
README | MIT license

Test Runner MCP

A Model Context Protocol (MCP) server for running and parsing test results from multiple testing frameworks. This server provides a unified interface for executing tests and processing their outputs, supporting:

- Bats (Bash Automated Testing System)
- Pytest (Python Testing Framework)
- Flutter Tests
- Jest (JavaScript Testing Framework)

- Go Tests
- Rust Tests (Cargo test)
- Generic (for arbitrary command execution)



Installation

```
npm install test-runner-mcp
```

Prerequisites

The following test frameworks need to be installed for their respective test types:

- Bats: `apt-get install bats` or `brew install bats`
- Pytest: `pip install pytest`
- Flutter: Follow Flutter installation guide
- Jest: `npm install --save-dev jest`
- Go: Follow Go installation guide
- Rust: Follow Rust installation guide

Usage

Configuration

Add the test-runner to your MCP settings (e.g., in `claude_desktop_config.json` or `cline_mcp_settings.json`):

```
{
  "mcpServers": {
    "test-runner": {
      "command": "node",
      "args": ["/path/to/test-runner-mcp/build/index.js"],
      "env": {
        "NODE_PATH": "/path/to/test-runner-mcp/node_modules",
        // Flutter-specific environment (required for Flutter tests)
        "FLUTTER_ROOT": "/opt/homebrew/Caskroom/flutter/3.27.2/flutter",
        "PUB_CACHE": "/Users/username/.pub-cache",
        "PATH": "/opt/homebrew/Caskroom/flutter/3.27.2/flutter/bin:/usr/local/bin:/usr/bin:/bin"
      }
    }
  }
}
```

Note: For Flutter tests, ensure you replace:

- `/opt/homebrew/Caskroom/flutter/3.27.2/flutter` with your actual Flutter installation path
- `/Users/username/.pub-cache` with your actual pub cache path
- Update PATH to include your system's actual paths

You can find these values by running:

```
# Get Flutter root
flutter --version

# Get pub cache path
echo $PUB_CACHE   # or default to $HOME/.pub-cache

# Get Flutter binary path
which flutter
```

Running Tests

Use the `run_tests` tool with the following parameters:

```
{
  "command": "test command to execute",
  "workingDir": "working directory for test execution",
  "framework": "bats|pytest|flutter|jest|go|rust|generic",
  "outputDir": "directory for test results",
  "timeout": "test execution timeout in milliseconds (default: 300000)",
  "env": "optional environment variables",
  "securityOptions": "optional security options for command execution"
}
```

Example for each framework:

```
// Bats
{
  "command": "bats test/*.bats",
  "workingDir": "/path/to/project",
  "framework": "bats",
  "outputDir": "test_reports"
}

// Pytest
{
  "command": "pytest test_file.py -v",
  "workingDir": "/path/to/project",
  "framework": "pytest",
  "outputDir": "test_reports"
}

// Flutter
{
  "command": "flutter test test/widget_test.dart",
  "workingDir": "/path/to/project",
  "framework": "flutter",
  "outputDir": "test_reports",
  "FLUTTER_ROOT": "/opt/homebrew/Caskroom/flutter/3.27.2/flutter",
  "PUB_CACHE": "/Users/username/.pub-cache",
  "PATH": "/opt/homebrew/Caskroom/flutter/3.27.2/flutter/bin:/usr/local/bin:/usr/bin:/bin"
}
```

```
// Jest
{
  "command": "jest test/*.test.js",
  "workingDir": "/path/to/project",
  "framework": "jest",
  "outputDir": "test_reports"
}

// Go
{
  "command": "go test ./...",
  "workingDir": "/path/to/project",
  "framework": "go",
  "outputDir": "test_reports"
}

// Rust
{
  "command": "cargo test",
  "workingDir": "/path/to/project",
  "framework": "rust",
  "outputDir": "test_reports"
}

// Generic (for arbitrary commands, CI/CD tools, etc.)
{
  "command": "act -j build",
  "workingDir": "/path/to/project",
  "framework": "generic",
  "outputDir": "test_reports"
}

// Generic with security overrides
{
  "command": "sudo docker-compose -f docker-compose.test.yml up",
  "workingDir": "/path/to/project",
  "framework": "generic",
  "outputDir": "test_reports",
  "securityOptions": {
    "allowSudo": true
  }
}
```

Security Features

The test-runner includes built-in security features to prevent execution of potentially harmful commands, particularly for the `generic` framework:

1. **Command Validation**
   - Blocks `sudo` and `su` by default
   - Prevents dangerous commands like `rm -rf /`
   - Blocks file system write operations outside of safe locations
2. **Environment Variable Sanitization**
   - Filters out potentially dangerous environment variables
   - Prevents overriding critical system variables

- Ensures safe path handling

3. **Configurable Security**
   - Override security restrictions when necessary via `securityOptions`
   - Fine-grained control over security features
   - Default safe settings for standard test usage

Security options you can configure:

```
{
  "securityOptions": {
    "allowSudo": false,        // Allow sudo commands
    "allowSu": false,          // Allow su commands
    "allowShellExpansion": true, // Allow shell expansion like $() or backticks
    "allowPipeToFile": false   // Allow pipe to file operations (> or >>)
  }
}
```

Flutter Test Support

The test runner includes enhanced support for Flutter tests:

1. Environment Setup
   - Automatic Flutter environment configuration
   - PATH and PUB_CACHE setup
   - Flutter installation verification
2. Error Handling
   - Stack trace collection
   - Assertion error handling
   - Exception capture
   - Test failure detection
3. Output Processing
   - Complete test output capture
   - Stack trace preservation
   - Detailed error reporting
   - Raw output preservation

Rust Test Support

The test runner provides specific support for Rust's `cargo test`:

1. Environment Setup
   - Automatically sets RUST_BACKTRACE=1 for better error messages
2. Output Parsing
   - Parses individual test results
   - Captures detailed error messages for failed tests

- Identifies ignored tests
- Extracts summary information

Generic Test Support

For CI/CD pipelines, GitHub Actions via `act`, or any other command execution, the generic framework provides:

1. Automatic Output Analysis
   - Attempts to segment output into logical blocks
   - Identifies section headers
   - Detects pass/fail indicators
   - Provides reasonable output structure even for unknown formats
2. Flexible Integration
   - Works with arbitrary shell commands
   - No specific format requirements
   - Perfect for integration with tools like `act`, Docker, and custom scripts
3. Security Features
   - Command validation to prevent harmful operations
   - Can be configured to allow specific elevated permissions when necessary

Output Format

The test runner produces structured output while preserving complete test output:

```
interface TestResult {
  name: string;
  passed: boolean;
  output: string[];
  rawOutput?: string;  // Complete unprocessed output
}

interface TestSummary {
  total: number;
  passed: number;
  failed: number;
  duration?: number;
}

interface ParsedResults {
  framework: string;
  tests: TestResult[];
  summary: TestSummary;
  rawOutput: string;  // Complete command output
}
```

Results are saved in the specified output directory:

- `test_output.log`: Raw test output
- `test_errors.log`: Error messages if any
- `test_results.json`: Structured test results

- `summary.txt`: Human-readable summary

Development

Setup

1. Clone the repository
2. Install dependencies:

```
npm install
```

3. Build the project:

```
npm run build
```

Running Tests

```
npm test
```

The test suite includes tests for all supported frameworks and verifies both successful and failed test scenarios.

CI/CD

The project uses GitHub Actions for continuous integration:

- Automated testing on Node.js 18.x and 20.x
- Test results uploaded as artifacts
- Dependabot configured for automated dependency updates

Contributing

1. Fork the repository
2. Create your feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

License

Releases

No releases published

Packages

No packages published

Contributors 2

privsim

punkpeye Frank Fiegel

Languages

TypeScript 71.2% **JavaScript** 23.8% **Shell** 3.2% **Go** 0.6% **Rust** 0.5% **Dart** 0.4% **Python** 0.3%