readme.md for @push.rocks/smartscaf

Lightning-fast project scaffolding with smart templates and variable interpolation

What It Does

SmartScaf is a powerful TypeScript scaffolding engine that transforms template directories into fully configured projects in seconds. Think of it as your project's DNA replicator - define once, deploy anywhere with custom variables, dependency merging, and automated post-setup scripts.

Perfect for:

- □□ **Bootstrapping microservices** with consistent structure
- □□ **Creating npm packages** with standard configurations
- □□ **Spinning up frontend projects** with your tech stack
- □□ **Standardizing team workflows** with company templates
- ⚡ **Automating repetitive project setups**

Installation

```
# Install globally for CLI usage
npm install -g @push.rocks/smartscaf

# Or add to your project
npm install @push.rocks/smartscaf --save-dev
```

Quick Start

Create Your First Template

1. **Set up a template directory** with your project structure:

```
my-template/
├── .smartscaf.yml       # Template configuration
├── package.json          # With {{projectName}} variables
├── src/
│   └── index.ts         # Your source files
├── readme.md            # With {{description}} placeholder
└── .gitignore           # Standard ignores
```

2. **Configure your template** in `.smartscaf.yml`:

```
# Template defaults and configuration
defaults:
  projectName: 'my-awesome-project'
  description: 'A fantastic new project'
  author: '{{gitName}}'
  license: 'MIT'
  nodeVersion: '20'

# Merge other templates (optional)
dependencies:
  merge:
    - ../shared-config-template
    - ../company-standards-template

# Post-scaffold automation
runafter:
  - 'npm install'
  - 'git init'
  - 'git add .'
  - 'git commit -m "Initial commit from SmartScaf"'
```

3. **Use Handlebars syntax** in your template files:

```
// src/index.ts
/**
 * {{description}}
 * @author {{author}}
 * @license {{license}}
```

```
 */

export class {{className}} {
  constructor() {
    console.log('Welcome to {{projectName}}!');
  }
}
```

Scaffold a New Project

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

async function createProject() {
  // Load your template
  const template = await ScafTemplate.createTemplateFromDir('./my-template');
  await template.readTemplateFromDir();

  // Supply variables (overrides defaults)
  await template.supplyVariables({
    projectName: 'super-api',
    description: 'Revolutionary API service',
    className: 'SuperAPI',
    author: 'John Doe'
  });

  // Interactive mode - asks for any missing variables
  await template.askCliForMissingVariables();

  // Generate the project
  await template.writeToDisk('./projects/super-api');

  console.log('□ Project scaffolded successfully!');
}

createProject();
```

Core Features

□□ Smart Variable System

SmartScaf intelligently manages template variables through multiple layers:

```
const template = await ScafTemplate.createTemplateFromDir('./template');
await template.readTemplateFromDir();

// Access variable collections
console.log(template.defaultVariables);   // From .smartscaf.yml defaults
console.log(template.requiredVariables);  // Found in template files
console.log(template.suppliedVariables);  // What you've provided
console.log(template.missingVariables);   // What still needs values
```

□□ Template Composition

Merge multiple templates to create complex project structures:

```
# .smartscaf.yml
dependencies:
  merge:
    - ../base-template      # Common structure
    - ../auth-template      # Authentication setup
    - ../docker-template    # Container configuration
```

Templates are merged in order, allowing you to build sophisticated scaffolds from modular components.

□□ Dynamic File Naming

Use frontmatter to dynamically rename files during scaffolding:

```
---
fileName: {{projectName}}.config.js
---

module.exports = {
  name: '{{projectName}}',
  version: '{{version}}'
```

```
};
```

□□ Interactive CLI Mode

Let SmartScaf prompt for missing variables:

```
// Automatically prompts for any variables not yet supplied
await template.askCliForMissingVariables();
```

□□ Post-Scaffold Scripts

Automate setup tasks after scaffolding:

```
runafter:
  - 'npm install'                    # Install dependencies
  - 'npm run build'                   # Initial build
  - 'npm test'                        # Verify setup
  - 'code .'                          # Open in VS Code
```

Scripts run in order using an interactive shell, ensuring proper environment handling.

Advanced Usage

Programmatic Control

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

class ProjectGenerator {
  private template: ScafTemplate;

  async initialize(templatePath: string) {
    this.template = await ScafTemplate.createTemplateFromDir(templatePath);
    await this.template.readTemplateFromDir();
  }

  async generate(config: any, outputPath: string) {
```

```
    // Supply configuration
    await this.template.supplyVariables(config);

    // Check what's missing
    if (this.template.missingVariables.length > 0) {
      console.log('Missing:', this.template.missingVariables);
      // Handle missing variables programmatically
      // or use: await this.template.askCliForMissingVariables();
    }

    // Generate project
    await this.template.writeToDisk(outputPath);
  }
}
```

Template Validation

```
// Validate template before using
const template = await ScafTemplate.createTemplateFromDir('./template');
await template.readTemplateFromDir();

// Check template structure
if (template.templateSmartfileArray.length === 0) {
  throw new Error('Template is empty!');
}

// Verify required variables
const required = template.requiredVariables;
const defaults = Object.keys(template.defaultVariables);
const missing = required.filter(v => !defaults.includes(v));

if (missing.length > 0) {
  console.warn(`Template needs: ${missing.join(', ')}`);
}
```

Complex Variable Structures

SmartScaf supports nested object variables:

```
# .smartscaf.yml
defaults:
  app.name: 'MyApp'
  app.version: '1.0.0'
  database.host: 'localhost'
  database.port: 5432
  api.endpoints.users: '/api/users'
  api.endpoints.auth: '/api/auth'
```

Use in templates:

```
// config.js
export default {
  app: {
    name: '{{app.name}}',
    version: '{{app.version}}'
  },
  database: {
    host: '{{database.host}}',
    port: {{database.port}}
  },
  api: {
    users: '{{api.endpoints.users}}',
    auth: '{{api.endpoints.auth}}'
  }
};
```

CI/CD Integration

```
// ci-scaffold.ts
import { ScafTemplate } from '@push.rocks/smartscaf';

async function scaffoldForCI() {
  const template = await ScafTemplate.createTemplateFromDir(
    process.env.TEMPLATE_PATH
  );
  await template.readTemplateFromDir();
```

```
  // Get variables from environment
  const vars = {
    projectName: process.env.PROJECT_NAME,
    environment: process.env.DEPLOY_ENV,
    apiKey: process.env.API_KEY,
    region: process.env.AWS_REGION
  };

  await template.supplyVariables(vars);
  await template.writeToDisk(process.env.OUTPUT_PATH);
}

// Run in CI pipeline
scaffoldForCI().catch(console.error);
```

Real-World Examples

Microservice Template

```
# .smartscaf.yml for microservice template
defaults:
  serviceName: 'my-service'
  port: 3000
  dockerImage: 'node:20-alpine'
  healthPath: '/health'

dependencies:
  merge:
    - ../shared/base-service
    - ../shared/monitoring
    - ../shared/logging

runafter:
  - 'npm install'
  - 'docker build -t {{serviceName}}:latest .'
  - 'npm run test:integration'
```

React Component Library

```
# .smartscaf.yml for React library
defaults:
  libraryName: 'ui-components'
  componentPrefix: 'UI'
  useTypeScript: true
  useStorybook: true
  testRunner: 'jest'

runafter:
  - 'pnpm install'
  - 'pnpm build'
  - 'pnpm storybook:build'
  - 'pnpm test'
```

API Reference

ScafTemplate Class

```
class ScafTemplate {
  // Factory method
  static createTemplateFromDir(dirPath: string): Promise<ScafTemplate>

  // Core methods
  readTemplateFromDir(): Promise<void>
  supplyVariables(variables: object): Promise<void>
  askCliForMissingVariables(): Promise<void>
  writeToDisk(destinationPath: string): Promise<void>

  // Properties
  name: string                        // Template name
  description: string                  // Template description
  templateSmartfileArray: SmartFile[] // Loaded template files
  defaultVariables: object            // Default values from .smartscaf.yml
```

```
  requiredVariables: string[]         // Variables found in templates
  suppliedVariables: object          // Variables you've provided
  missingVariables: string[]         // Variables still needed
}
```

Best Practices

□□ Template Organization

```
company-templates/
├── base/                    # Shared foundations
│   ├── typescript/         # TS configuration
│   ├── eslint/            # Linting setup
│   └── ci-cd/             # CI/CD pipelines
├── services/               # Service templates
│   ├── rest-api/
│   ├── graphql-api/
│   └── worker-service/
└── frontends/              # Frontend templates
    ├── react-app/
    ├── vue-app/
    └── static-site/
```

□□ Security Considerations

- Never commit sensitive data in templates
- Use environment variables for secrets
- Add `.smartscaf.yml` to `.gitignore` if it contains private defaults
- Validate user input when using in automation

□□ Template Design Tips

1. **Start small** - Begin with minimal templates and grow them
2. **Use composition** - Build complex templates from simple ones
3. **Document variables** - Add comments in `.smartscaf.yml`
4. **Test templates** - Create test scaffolds before deploying

5. **Version templates** - Use git tags for template versions

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 11:12:30 UTC by foss.global Team
Updated 2026-03-28 12:19:16 UTC by foss.global Team