SOVERIUS AI

From Hours to Minutes

An AI Case Study with Sheriff

Murat Sari **Rainer Hahnekamp**

Powered by https://soverius.ai

About me

Murat Sari, **ANGULAR***architects.io / CodeRabbit.at*

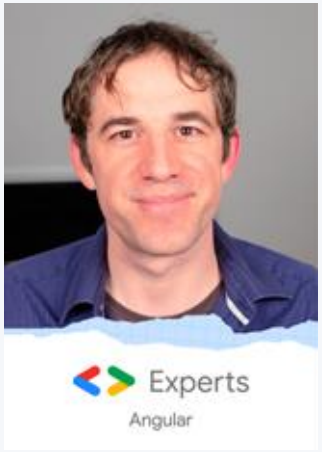
- *Software Development Services*
- *Trainer & Consultant für Angular*

Angular Workshops
and Consulting / Code Audits

Custom development
(Cloud / Iot / AI Agents)

High performance-
apps

About Me...



- **Rainer Hahnekamp**
- **Soverius.ai**
- **NgRx Core Team**
- **Developer / Trainer / Speaker**

@RainerHahnekamp

https://www.youtube.com/@RainerHahnekamp

https://www.ng-news.com

Open Source Projects

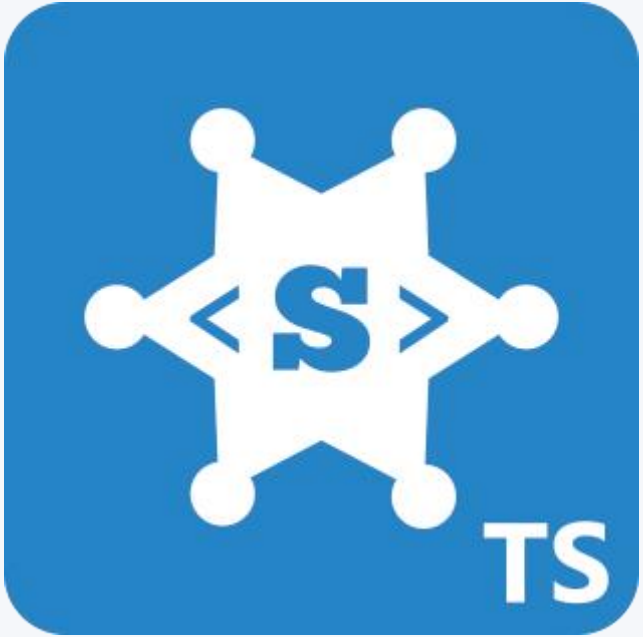
NgRx

Sheriff

NgRx Toolkit

Testronaut

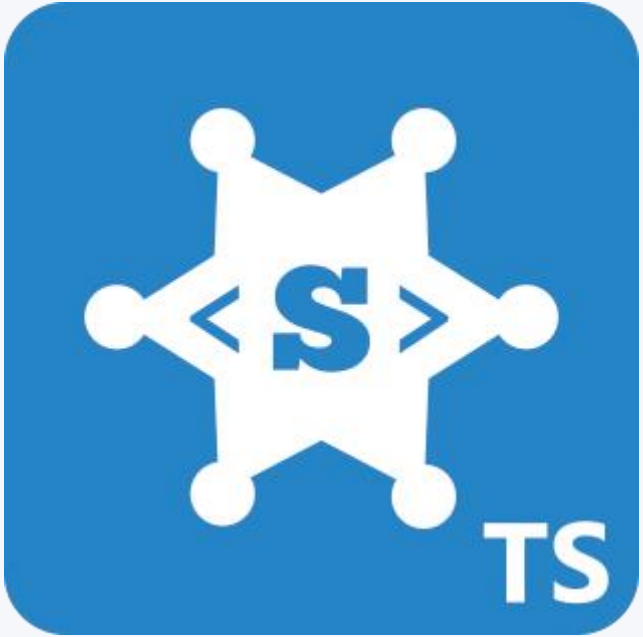
Sheriff: Modularity in TypeScript



- Module Encapsulation
- Dependency Rules

SOVERIUS AI

Sheriff: Modularity in TypeScript



- Module Encapsulation
- Dependency Rules
- Lightweight
- Convention over Configuration
- Zero Dependencies
- For all TypeScript Projects

SOVERIUS AI

From simple configurations...

```
export const sheriffConfig: SheriffConfig = {
  modules: {
    'src/app': {
      shared: 'domain:shared',
      'domains/<domain>': {
        'feature-<feature>': ['domain:<domain>', 'type:feature'],
        'ui-<ui>': ['domain:<domain>', 'type:ui'],
        data: ['domain:<domain>', 'type:data'],
        'util-<ui>': ['domain:<domain>', 'type:util'],
      },
    },
  },
  depRules: {
    root: ['*'],

    'domain:*': [sameTag, 'domain:shared'],

    'type:feature': ['type:ui', 'type:data', 'type:util'],
    'type:ui': ['type:data', 'type:util'],
    'type:data': ['type:util'],
    'type:util': noDependencies,
  },
};
```

...to something else

```ts
export const sheriffConfig: SheriffConfig = {
  entryFile: 'src/main.ts',
  enableBarrelLess: true,
  modules: {
    'src/app': {
      'shared/<shared>': ['shared', 'shared:<shared>'],
      domains: {
        diary: ['domain:diary', 'type:feature'],
        bookings: ['domain:bookings', 'type:feature'],
        '<domain>/api': ['domain:<domain>:api', 'sub-domain:api', 'type:api'],
        '<domain>/feat-<feature>': [
          'domain:<domain>',
          'sub-domain:none',
          'type:feature',
        ],
        '<domain>/sub-<name>': [
          'domain:<domain>',
          'sub-domain:<name>',
          'type:feature',
        ],
        '<domain>/sub-<sub>': {
          data: ['domain:<domain>', 'sub-domain:<sub>', 'type:data'],
          model: ['domain:<domain>', 'sub-domain:<sub>', 'type:model'],
          ui: ['domain:<domain>', 'sub-domain:<sub>', 'type:ui'],
        },
        '<domain>/<type>': [
          'domain:<domain>',
          'type:<type>',
          'sub-domain:none',
        ],
      },
    },
  },
  depRules: {
```

```ts
depRules: {
  root: [...['type:api', 'type:feature'], 'shared'],
  'domain:*': [
    sameTag, // domain:bookings -> domain:bookings
    'shared',
    ({ from, to }) => from.endsWith(':api') && from.startsWith(to), // domain:bookings:api -> domain:bookings
  ],
  'sub-domain:api': ({ to }) => to.startsWith('sub-domain'),
  'sub-domain:*': [sameTag, 'shared'],
  'type:api': [({ to }) => to.startsWith('type'), 'shared:config'],
  'type:feature': [
    ...['type:api', 'type:data', 'type:ui', 'type:model'],
    'shared:config',
    'shared:form',
    'shared:master-data',
    'shared:testing',
    'shared:ui-messaging',
    'shared:util',
  ],
  'type:data': [
    'type:model',
    'shared:config',
    'shared:ui-messaging',
    'shared:util',
  ],
  'type:ui': ['type:model', 'shared:form', 'shared:ui', 'shared:util'],
  'type:model': noDependencies,
  shared: 'shared',
  'shared:http': ['shared:config', 'shared:ui-messaging'],
  'shared:ngrx-utils': 'shared:util',
  'shared:security': 'shared:http',
  'shared:ui-messaging': 'shared:http',
  ...api('bookings', 'customers'),
},
```

Motivation

- Creating a config could be overwhelming
  - How to split - Architecture Styles
  - Transform the desired rules into a config
  - How to verify the config
- Maybe AI can automate it?
  - Ask to analyze the project
  - Suggest a style or domain cut
  - Verify the architecture
  - Visualize the architecture

```
import { noDependencies, sameTag, SheriffConfig } from '@softarc/sheriff-core';

export const config: SheriffConfig = {
  version: 1,
  tagging: {
    'src/app': {
      'shared/<type>': ['shared', 'shared:<type>'],
      bookings: ['domain:bookings', 'type:feature'],
      'customers/api': ['type:api', 'domain:customers:api'],
      '<domain>/<type>': ['domain:<domain>', 'type:<type>'],
    },
  },
  depRules: {
    root: ['type:feature', 'shared:*'],
    'domain:*': [sameTag, 'shared'],
    'domain:bookings': 'domain:customers:api',
    ....
  },
};
```

Challenges

Context Size

Data Privacy

Token Consumption

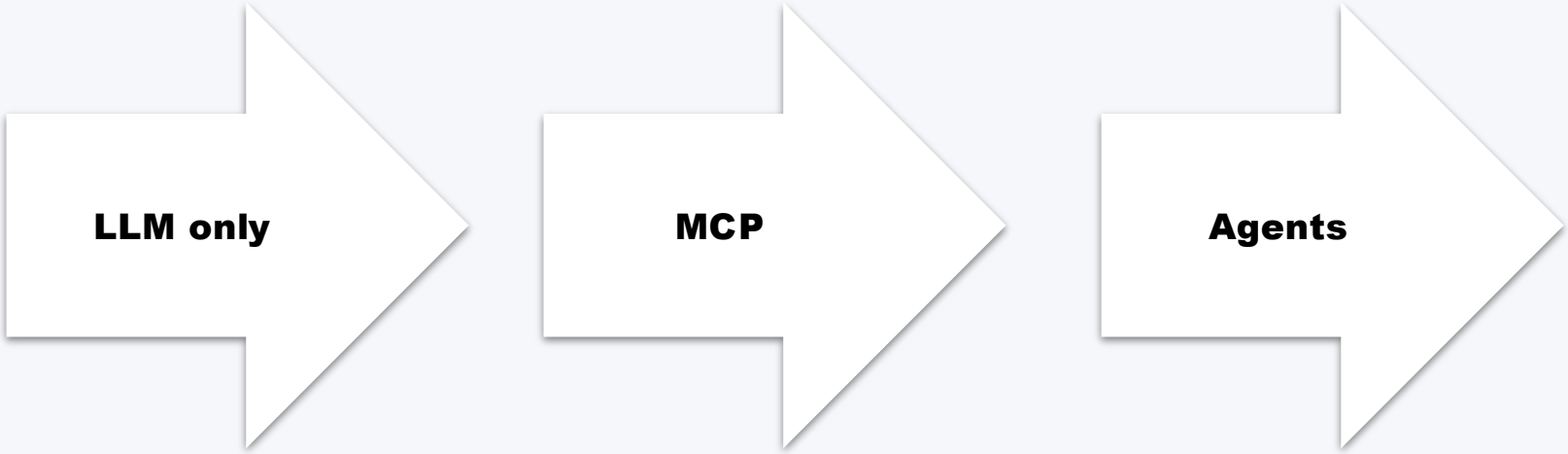
Quality

Latency

Features without AI

***Foundation is static**

SOVERIUS AI



SOVERIUS AI

Approach 1 - ICL Prompting (No MCP / No Tools)

- How can we use AI to produce a valid Sheriff configuration as a reliable starting point?
- Structured prompting with In-Context Learning (ICL)
  - The idea is to embedding structured context (documentation, examples and constraints) within the system prompt.
  - Here is our example https://hackmd.io/@wolfmanfx/SyKDmNveWx
  - Demo

Approach 2 - Multi-state agent system

- Router agent - analyzes user input and decides
- Each phase / state is specialized
  - Modular context engineering
  - Direct tools calls (no mcp overhead)
  - Structured outputs
- Final agent is doing natural language presentation

Approach 2

- State machine controlled by an “ROUTER SYSTEM PROMPT”
  - acts as a controller that routes requests to sub handler
- Each state consist of a short lived context and specific goal
  - We do not blow up our context as we always include only the data we need in each specific sub state (each state has an isolated context)
- DEMO

Approach 3 - Own LLM

- Current LLMs (Frontier / Local) do not know sheriff
  - Leads to full hallucination (if no ICL prompting is applied or MCP is used)
- Idea train the LLM to be a “Sheriff configuration expert”
  - Should prevent hallucinate incorrect answers / configs
- Traditional Fine-Tuning Problem
  - Full model: ~1.1 billion parameters (TinyLlama-1.1B-Chat-v1.0)
  - Training: Update all 1.1B parameters
- LoRA - Low-Rank Adaption
  - Freeze base model: 1.1B parameters (locked)
  - Train small adapters: 110M parameters (10%)

Approach 3 - Full Fine Tuning / LORA

- Step 1 - Create Examples (Manual / Automated)
- Step 2 - Prepare Training Data
- Step 3 - Train Model
- Step 4 - Test Model
- Step 5 - Convert for Deployment (GGUF / MLX)

Approach 3 - Full Fine Tuning / LORA / Step 1

- We have created the examples in markdown
  - Following this structure:
    - Question - “Generate a basic Sheriff config for...)
    - Additional input (Project structure)
    - Than a requirements section - “Each domain can only access...”
    - Most important the expected result (Sheriff config)
  - Why markdown
    - Easy to use
    - Human-readable and editable
  - Created ~15 manual examples and extrapolated to ~250 examples using “Ai”

Approach 3 - Full Fine Tuning / LORA / Step 2

- Data Preparation
  - Input our markdown folder
  - Output JSONL in ChatML format
- JSONL (JSON lines)
  - It means we store one valid JSON per line in the file
  - How we store our examples on disk
- ChatML (Always check the model how it expects the training data)

Approach 3 - Full Fine Tuning / LORA

- We use Transformers lib from hugging face (AutoModelForCausalLM)

```python
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",
    trust_remote_code=True,
    torch_dtype=torch_dtype,
)

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, trust_remote_code=True)

if USE_LORA:
    model.gradient_checkpointing_enable()
else:
    for param in model.parameters():
        param.requires_grad = True
```

What is @softarc/sheriff? ## Project Structure N/A - This is a question about the Sheriff package itself. ## Requirements Explain what @softarc/sheriff is and how to use it.

@softarc/sheriff is an architectural linting tool for TypeScript and Angular projects. It helps enforce module boundaries and dependency rules in your codebases. To get started, you need to install Sheriff using npm or yarn:

```bash
npm install @softarc/sheriff
# or with --save-dev flag
yarn add -D @softarc/sheriff
```

Once installed, you can configure Sheriff in your `sheriff.config.ts` file. Here's a basic configuration:

```typescript
import { SheriffConfig } from '@softarc/sheriff';
export const config: SheriffConfig = {
  modules: {
    'src/app/<domain>/<type>': ['domain:<domain>', 'type:<type>'],
  },
  depRules: {
    'root': 'noTag',
    'noTag': [
      'noTag',
      'root'
    ],
    root: ['domain:*', 'type:*'],
  },
};
```

- **modules**: Maps module paths to their tags. Keys with colons (domain and type) MUST be quoted strings with colons: `'domain:', 'type:', and 'type:data' NOT 'type': 'data'.
- **depRules**: Defines which tags can depend on which other tags. Keys with colons MUST be quoted strings with colons: `domain:*`, 'type:', and 'type:data' NOT 'type': 'data'.
- Import paths are checked against the defined depRules to prevent circular dependencies.

To use Sheriff, you need to import from modules with tags:

Send a message

sheriff-ai

What is @softarc/sheriff? ## Project Structure N/A - This is a question about the Sheriff package itself. ## Requirements Explain what @softarc/sheriff is and how to use it.

@softarc/sheriff: A High-Performance Rust Framework for Building Web Applications

Overview

@softarc/sheriff is a high-performance, modern web framework written in Rust that enables developers to build robust, scalable, and maintainable web applications. It is designed to be efficient, flexible, and easy to use, making it an attractive choice for building complex web systems.

Key Features

- **Async Support**: Sheriff is built on top of async/await, which allows for asynchronous programming and non-blocking I/O operations. This results in better performance, especially when handling high-traffic or long-running requests.
- **Fast Routing**: Sheriff uses a fast and efficient routing mechanism that minimizes overhead and optimizes the flow of incoming HTTP requests.
- **Built-in Support for WebSockets**: Sheriff includes built-in support for WebSockets, making it easy to integrate real-time communication into your application.

Getting Started

To get started with @softarc/sheriff, follow these steps:

1. Add `@softarc/sheriff` as a dependency in your `Cargo.toml` file:

```toml
[dependencies] sheriff = "0.10"
```

2. Create a new Sheriff project using the following command:

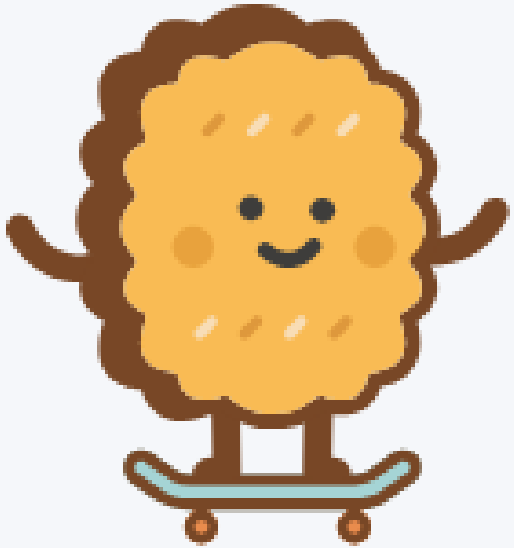
Send a message

llama3.2

Demo Model

- Video

Outlook



Summary

- AI makes a significant contribution

- AI as a helper
  - All tasks can also be done without AI
  - No dependency to AI
  - Tooling to verify the outcome (non-deterministic behavior)
- Where AI can't help us
  - Specific UI
  - Raw Import Graph
- Mixed approaches
  - No MCP
  - MCP with controlled tooling access
  - Full MCP
  - State Machine


SOVERIUS
AI
Thank you!