# POC : Hawaii-Cli

**Goal:** Enable the developers to manage GraphQL schemas, permissions, configurations and endpoints.

**RoadMap:** Hawaii Developer Experience (CLI) One Pager

## Why CLI?

* It is much faster and efficient than any other type of Interface.
* It requires fewer resources, reducing dependencies on other systems for GUI
* It will give more control over the app.

The Hawaii CLI tool is designed to help a developer be more productive by making app development faster.

## Why C#.NET?

* C# can run very fast, and when it comes to building CLI tools, we need to be efficient, unlike Python which is also popular for building CLIs but are slower compared to C#.
* C# is widely used within and outside Microsoft, so integrating it with other apps are much easier.
* .NET is one of the most loved framework around the world. It is simply independent and it tends to support all the platforms. It is easy to deploy and it supports many languages. Apart from that, It’s also very reliable and stable.

The Hawaii-Cli will generate a config file that will contain all the information required to connect to different DBs and query them. It will consist of the Environment, permissions and relationship between the tables as well.

There will be 2 types of Host Environment:

1. **Development**: It will be used for testing new changes before deploying it into the production environment.
2. **Production**: It will be the Production environment connecting to Production DB.

## Why are we using JSON for Config?

* JSON can be used with any programming language as it is not language dependent.
* JSON is much easier to read and understand as it is stored in a standard format.
* JSON is much better in terms of Serialization as it is supports many different data types like object, string, integer, null, and Boolean.
* YAML is another format popular for configuration files, but when it comes to handling large files JSON is much faster.

## What are the benefits of using a Config File?

* Since information like database type, connection string, host environment, permissions are all in a single file, shareability, and maintainability is increased.
* Any change in the accessibility of a table or its relationship will only require updating the config and no change in code.
* We can have different config files for different environments which will allow us to maintain parallel workflow.
* Testing and Deployment will be much faster.

## Designing the CLI tool:

* **Adding “--help” command into our Cli**

Text

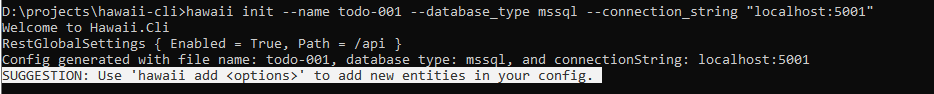
Description automatically generated

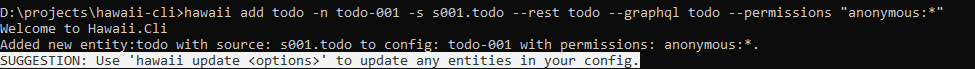
* **Human readable error messages  
    
  Text

  Description automatically generated**
* **Suggest Next Steps**

Our Cli tool will suggest the next commands that needs to be used. For example: when a user does ***“init”***, after completing the initialization step, it will suggest the next command as ***“add”.***

Similarly, after **“Add”**, it will suggest **“update”.**

****

****

## Implementation:

* The Hawaii-gql repo already has classes defined for different objects in runtime config, i.e.RunTimeConfig.cs. We will use this repo as an external library for our Hawaii-cli to reuse those classes for generating the config.
* We will organize the codes into different modules for better maintainability and reusability.

We will have directory called Classes which will contain different classes such as

**CommandLineOptions** - > It will only contain the options that are available to developers

**CommandLineHelp** -> Which will parse the command line options

**Operations** - > it will contain 3 main commands: Init, Add and Update

**Utils** -> this will contain all the utility methods such as text transformations, default value generation, and formatting, etc.

**ConfigGenerator** -> It will contain the methods to make changes to config.

* Config file will be generated in the directory where the commands are being run.
* If the user provides the name of the file, it will be considered else, it will generate a config file with default name: **hawaii-config.json**
* We can use **System.text.Json** for Serialization and De-Serialization of our JSON config file.
* **Humanizer.Core** can be used for singular and plural text generation for graphql types.

## Working:

* **Initializing the Config file with db details.**

hawaii init --name todo-001 --database\_type mssql --connection\_string "localhost:5001"

Text

Description automatically generated

* **Adding an entity to the Config file**

hawaii add todo -n todo-001 -s s001.todo --rest todo --graphql todo --permissions "anonymous:\*"

Text

Description automatically generated

* **Adding another entity to the config file**

hawaii add books -n todo-001 -s s001.books --rest books --graphql books --permissions "anonymous:create,read,update" --fields.include "id,type,review"

A screenshot of a computer

Description automatically generated with low confidence

* **Adding one more Entity in the Config file with different values of flags**

hawaii add review -n todo-001 -s s001.reviews --rest true --graphql true --permissions "anonymous:update" --fields.exclude "id,time"

Text

Description automatically generated

* **Updating an Entity in the config using “update” command**

hawaii update review -n todo-001 -s s001.review --rest review --graphql review --permissions "authenticate:create" --fields.include "content,rating"

A screenshot of a computer screen

Description automatically generated with medium confidence

* **Updating an entity by adding a relationship between two entities**

Adding 1:N relationship between books and reviews(A book can have multiple reviews)

hawaii update reviews add relationship books --target.entity reviews --cardinality many --mapping.fields "id:book\_id"

hawaii update books add relationship reviews --target.entity books --cardinality one --mapping.fields "book\_id:id"

