GSoC'23 Proposal - Keploy

<Keploy CLI Refactoring>

Personal Details

Name: Cássio Milczareck
Course: Computer Science
Email: cassiomilczareck@gmail.com
Github: https://github.com/cassiozareck
LinkedIn: https://www.linkedin.com/in/cassio-ferreira-milczareck-256107261/
Phone: 5551995270656
Current Country: Brazil

Link to Resume / CV:
https://docs.google.com/document/d/13pQHxane6knqnn7fsJAtAt8NEW6rVEmd0_RB8Dq2tOU/edit?usp=sharing

About Yourself

1. Please describe yourself, including your development background and specific expertise.
   I'm mainly a self-taught person that finds a hobby in programming and always wanted to work on an open source project. I started to learn programming at 14 years old. This and last year I've been learning mocks, tests, rest apis and getting deeper knowledge at go and its ecosystem (docker, kuber…), also learned about algorithms like bst, hashmaps, tortoise and hare, sorting… Also I've used postgres in some projects I've made for myself just for practice reasons.
   Also I know how to manage time and write reliable goals as I've already made a project that lasts more than 3 months. I'll say more about it at the end.

2. Why are you interested in the Keploy project(s) you stated above?
   The CLI refactor is the project where I can use my strongest points as it relies directly on apis, go and algorithms.

3. Have you participated in an open-source project before? If so, please send us URLs to your profile pages for those projects or some other demonstration of the work that you have done in open-source. If not, why do you want to work on an open-source project in GSoC this summer with Keploy?
   I've never participated in an open-source project before. But there's a lot of reasons of why Keploy:
   1. **Keploy's idea has a future.** i've never seen such a project before and I think with time it will be widely used.
   2. **I see Keploy as a project where I can be very useful.** As I said before my strongest abilities are go, tests, rest apis…
   3. **Keploy community is very good.** Every doubt I had there was always someone to answer me.
   4. **It's a relatively new project**, so there's still a lot to do in it and a lot to grow and I'm here to help Keploy in it.

Commitment

1. **Are you planning any vacations during the GSoC period?**
   No. In fact, it is winter during this time (kinda ironic) so there's not much to do if you're not in a warm room at your pc contributing to Keploy.

2. **How many classes are you taking during the GSoC period?**
   3.

3. **Do you have any other employment during the GSoC period?**
   No.

4. **How many hours per week do you expect to work on the project and what hours do you tend to work?**
   As my classes are at night I'll start to work at 9:30 am, as most mentors are from India this time will be at 6:00pm there. The time estimation for this project is 175 hours which is half as the others. But I will work 35 hours / week to guarantee I'll finish the project in the given time.

Contributions so far

- PRs merged and Unmerged
  - Improve test coverage: https://github.com/keploy/keploy/pull/471
  - Beautiful diff output: https://github.com/keploy/keploy/pull/492
  - Pipe keploy outputs to file: https://github.com/keploy/keploy/pull/523

- Issues, bugs found
  NA

- Documentation contributions
  The PR I've made are documented
- Other contributions
  In both Beautiful diff output and Pipe keploy outputs PR I’ve refactored a bit the old code to be more simple and clean. Also I showed Keploy to some people and helped them to integrate into their projects.

Proposal

Overview

The aim of this proposal is to help Keploy server be more **pragmatic, intuitive for new-comers, and more powerful for debugging (as it will can run individual tests). Also to extend its capabilities by adding features such as commands, configuration for pipe files and even reduce its complexity (as there will be a new config file that will help a command to run even without Keploy needing to be running or making complicated api calls)**. The project will be divided into 4 parts. They are listed below:

- **Part 1: Add support for basic commands.** Here we will add support for 6 basic new commands: keploy **new**, keploy **select**, keploy **run**, keploy **listen**, keploy **get**, keploy **help**
- **Part 2: Rename tests and add comments to the yaml file.** We will rename generated test-cases to method_endpoint and also add a comment to whether it's a mock or a test.
- **Part 3: Add a configuration inside Keploy where it's possible to pipe logs to a file.** so users can use it as they want. **Also add a link to show the test report generated at the end of the log.**
- **Part 4: Add beautiful log diff viewer for actual and expected response**. In this part we will create a new way to visualize failed tests, where the response doesn't match, where the difference may appear in a hard-to-see part of the response.

To accomplish this we will use Golang of course and write new features inside Keploy server code and whenever we can I will integrate new features with already existing functions so we don't end up duplicating code and take advantage of more mature functions. In the third part I will write some new handlers for the Keploy api to achieve the third part. Also at the course of this project we will need to add some packages as they will help us achieve our tasks and are always less bug-prone than rewriting the wheel ourselves. These news features will be rigidly tested to make sure we don't break anything. **This way we will turn Keploy into a more powerful tool than it already is.**

Details

1. About Keploy support of commands

- **Problem:** Keploy currently does not support commands and only offers a binary to initialize a server, typically on port 6789. This lack of command support can leave users feeling lost about its functionality and limit their ability to interact with the tool. Users have to navigate through YAML files to view test details and cannot run individual test cases, which can be time-consuming and not what the user may want.

- **Solution:** Introduce five new commands (new, select, get, run, and help) using the Cobra library to make Keploy more intuitive, user-friendly, and practical for daily use. These commands will transform Keploy into a more interactive software, allowing users to easily run individual test cases, view specific test details, and access help documentation.

- **Implementation:** To add support for commands without affecting existing code, we will do these steps:
    - Install and integrate Cobra into the project as well define the root command, which will serve as the main entry point for the application.
    - Create separate files under cmd folder for each new command (new, select, get, run, and help), as recommended by Cobra's conventions.
    - Implement the desired functionality within the 'Run' function of each command file object.
    - Include subcommands that require additional input, such as test ID or App ID, and sanitize these inputs to ensure they are valid.

- **Code example (under cmd/new.go):**

```
func init() {
    rootCmd.AddCommand(newCmd)
}

// newCmd represents the "new" command
var newCmd = &cobra.Command{
    Use:   "new",
    Short: "Create a new Keploy application reference",
    Long: `Create a new Keploy application reference with the specified
App ID and add it under config.yaml...`,
    Args: cobra.ExactArgs(1),
    Run: func(cmd *cobra.Command, args []string) {
        appID := args[0]

        // Sanitize the input App ID
        sanitizedAppID, err := SanitizeInput(appID)
        ...

        // Creates a new application reference based on the
       // given ID
        err = newApplication(sanitizedAppID)
        if err {
        ...
}
```

- **Benefits of using Cobra:**
    - Provides a clear structure for organizing commands and subcommands.
    - Simplifies input validation and command execution.
    - Offers built-in support for generating help documentation.
    - Widely used and well-documented, making it easier for developers to maintain and extend the code.

- **Result:** By implementing the new commands using the Cobra library, Keploy will become more user-friendly and offer a streamlined, organized approach to managing its commands functionality.

1.1 About keploy new

Let's start by “keploy new” command as it **is crucial for enhancing Keploy's usability,** making it **simpler and more flexible for users**, making possible the functionality of commands like **get** and **run**, and to expand the horizons of Keploy by storing crucial data in a persistent way that can be used in N different ways.

- **Problem:** With the new commands being added we need a simple and flexible way to manage the communication with the commands and the associated application. Without the ability to gather information from an offline way, users must rely on initializing the server, which increases complexity and makes it difficult to use other commands like **get** and **run**.

- **Solution:** Introduce the **keploy new** command and the **config.yaml** file with the application list to improve Keploy's simplicity and flexibility, making it easier to manage communication between different applications.
    - **Advantages:**
        - Allows Keploy to gather information from a local file without initializing the server, reducing complexity.
        - Enhances the functionality of other commands like "get" and "run."
        - Simplifies application association by creating and storing a list of application references.

- **Implementation:**
    - Check if the config.yaml file exists when running the **keploy new** command, creating one if necessary.
    - Add **an application reference** referent to the application user wants to work into a **list of applications references**. The **application reference** will have these informations:
        - App ID.

        - Mock and test path based on the folder which **keploy new** was executed.
    - The **list of applications references** will be a config.yaml file within the keploy_config folder, ensuring data persistence.

- **Basic usage: keploy new <appid>**
    - The App ID is important for storing debug information and distinguishing between different applications.
    - The command must be run within the application that the user wants to work with.
- **Result:** Now the Keploy server will always have information about the applications and its paths. We can run **keploy get** without the server being running because he already has the information it needs. It's in fact good even for **keploy run** because now if someone wants to test just one case it can without have to handle a keploy server request for one test to the sdk (generally who sends request is the sdk so it would be difficult to handle an **keploy run** with the sdk as it implies the requests now come from the server). In the end of the day we would have a **config.yaml** file like this:

```yaml
# This is the app id
sample-url-shortener:
  path: /home/user/echosql/
  testPath: /home/user/echosql/keploy/tests
  mockPath: /home/user/echosql/keploy/mocks

other:
```

1.2 About keploy select

- **Problem:** Keploy needs a way where it can associate commands with **one** application and change the application if the user wants to.

- **Solution:** mark a label in the **config.yaml** file that **indicates** which application the Keploy commands will be for.

```yaml
...

selected: sample-url-shortener
```

- **Basic usage:** keploy select <App ID>

1.3 About keploy get

- **Problem:** Users currently cannot easily view specific test cases without navigating through YAML files, making it difficult to quickly access individual test details.

- **Solution:** Implement the "keploy get" command to enable users to view a specific test case generated by keploy by its ID, using the **config.yaml** file to obtain the test-cases path and the selected application's App ID.

- **Implementation:**
    - Refactor the existing Get function in testCase.go and add a Read() function in fs/mock.go.
    - Modify the Get function to read a single test case from the YAML files located at keploy/tests folder under the selected application and display a user-friendly output on the screen.

- **Usage:** keploy get <id of the test (e.g., 2)>

1.4 About keploy run

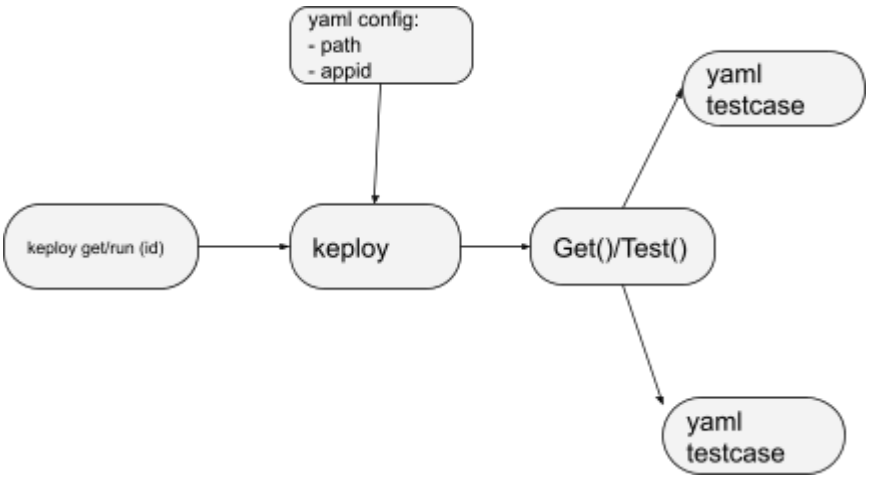
- **Problem:** Users currently cannot run individual test cases and view the results directly on the terminal, making it challenging to focus on specific tests.

- **Solution:** Implement the "keploy run" command to allow users to execute a specific test case by ID and display the results on the terminal.

- Implementation:
    - Use the config.yaml file to obtain the chosen application's App ID and paths.
    - Execute the existing Test function in pkg/service/regression.go, providing the appropriate data to run the specific test case.

- Usage: keploy run <id of the test (e.g., 2)>

A little graph to show both “keploy run” and “keploy get” intern working



1.5. About keploy help

Well generally when we start to have multiple commands for one software it's always a great idea to have a way to help new users of what they should do and what these commands do. This command will basically output some information that can explain what each cited command does. To accomplish this we can even reuse some of the text used in this proposal as it has some explanation about new commands.
As we're using Cobra to power the commands implementation this job will be easy. Cobra's root cmd function is a built-in help output that can be easily modified
Usage: **keploy help**

1.6. About keploy listen

This one will do what **keploy already listens**. Basically it runs the server. I've chosen to make it a command too because if the user just types **keploy** and it runs, new users may not know there are different commands.
Usage: **keploy listen**

2. About Keploy yaml tests refactor

As mentioned at the project ideas for the CLI 2023, one of the ideas is to refactor the name of the tests file.
Currently Keploy export tests using yaml files, the name of it is test/mocks + id.yaml, but one better idea is to change its name to **method_endpoint** as it is more clearer for the user what it really is and kind of create more relation between mocks and tests as they are in fact related (many Keploy functions are reutilized both for mocks and tests).
But also the user must know where it is a mock or a test so we will add KMocks comments to those who are mocks and KTests to those yamls who are tests.

2.1 Renaming files

To accomplish it we can refactor these lines in testCase.go as every http or grpc call passes through it.

```
    id = fmt.Sprintf("test-%v", lastIndex+1)
```

```
    ...
    m    :=    "mock-"    +    fmt.Sprint(lastIndex+1)    +    "-"    +
strconv.Itoa(i)
```

Change both strings to “method_endpoints” like so:

```
    id = fmt.Sprintf("method_endpoint-%v", lastIndex+1)
    ...
    m  :=  "method_endpoint-"  +  fmt.Sprint(lastIndex+1)  +  "-"  +
strconv.Itoa(i)
```

After doing it I will change the tests at the testCases_test.go so they can match these new names.

2.2 Adding comments

- **Objective:** Clearly distinguish between mock and test YAML files by adding comments, "KMocks" for mocks and "KTests" for tests.

- **Implementation:**
    1. In the fs/mock.go file, identify the section where Keploy parses the Go Object representing the test case into a byte array that represents the YAML file.
    2. Before the parsing, insert bytes that represent the comment, either "#KTest" or "#KMock", depending on the type.
    3. Ensure the comment appears at the beginning of the YAML file, making it easy for users to identify the file type.

3. Output features: pipe logs and test-reports output

3.1 About keploy server pipe

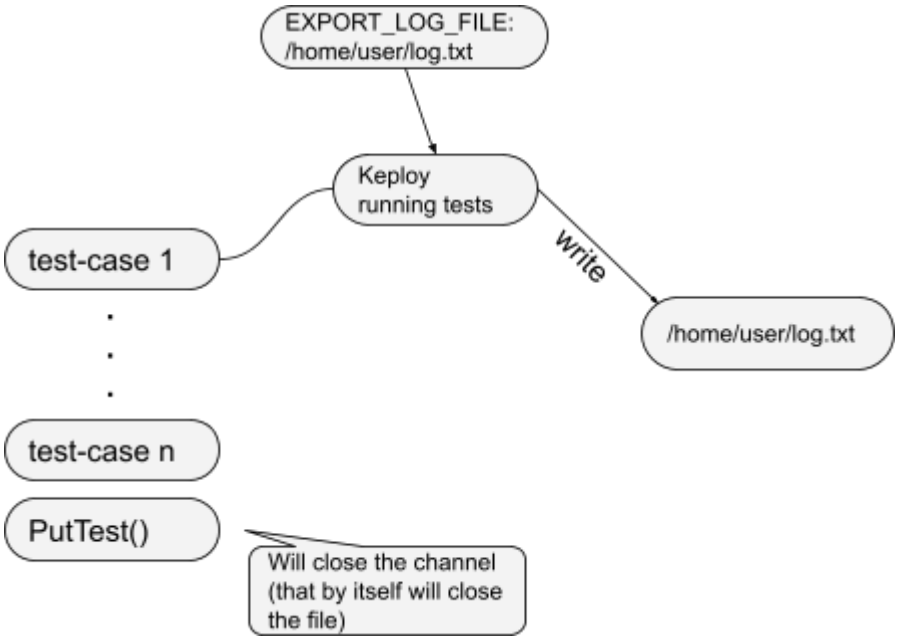
- **Problem:** Keploy server doesn’t have an option to pipe output from the server CLI to a file which is bad if someone wants to save the output in a file.

- **Solution:** As Keploy already structures its logs in one variable called logs at test run phase, we can Implement an environment variable called EXPORT_LOG_FILE that allows users to specify a file path for the output. We can then redirect the logs to the file. Also as Keploy makes use of Zap to be its logging library we can use it to redirect its logging output to a file just by defining a new core for Zap.

- **Implementation:**
    - When the EXPORT_LOG_FILE variable contains a file path, Keploy will pipe its output for both gRPC tests and HTTP tests to the specified file.

    - Use channels operations to handle multiple test cases and close the channel after the test run is complete.
    - We can close the channel after the test run is done at the end of PutTest as it is the final summary.
- **Issues with PrettyPrinter (PP):** PP uses ASCII to handle colors, which causes issues when piping output to a file (as mentioned in PR https://github.com/keploy/keploy/pull/457).
    - Solution: Use normal logs during the test run, and at the end, either print logs with PP or pipe them to a file based on the EXPORT_LOG_FILE variable.

Graph explaining its implementation:



3.2 Test-reports static link

- **Problem:** Currently, test reports are generated within the test-reports/ folder with a given UUID, but there is no easy way for users to access these reports to view the test results.

- **Solution:** Generate a static link at the end of the logs, which redirects users to the newly created test case report, making it more convenient to view the test results.

- **Implementation:**
    1. Create a new handler that, when called, searches for the test-reports with the specified UUID.
    2. Return a string representation of the YAML file to the client, such as a browser, that made the request.

3. Display the generated static link at the end of the logs so users can easily access the test case report.

Addition: If the user uses the "/" route we can easily return a list of hyperlinks of test-reports based on the folder they are being generated, so the user can get any test-report he or she wants.

4. Log json diffs

- **Problem:** currently Keploy doesn't actually calculate the difference between the expected and actual response, it just highlights fields that are unmatched. If we have a long body difference it can be difficult to visualize the difference. The problem is bigger when most of the applications are in Rest APIS formats, there's no way to visualize the difference without a proper formatting to adapt to this kind of communication.

- **Solution:** Implement a way where Keploy can compare two jsons and print it side-by-side highlighting the difference between those two. Also doing it in a beautiful way where it becomes very eye-friendly.
  We can do it by using mature, already-existent packages that could help us parsing the json and finding the difference between them, also we need a package for printing it side-by-side in a beautiful and proper way. **Gojsondiff** and **tablewriter** are both good packages in this situation. The first would take care of the diffs of the jsons, the second would put it into a table where the left-side is the expected and the right-side is the actual response.

- **Implementation:**
  - Integrate **tablewriter** and **gojsondiff** packages into the project
  - Compare the jsons using **gojsondiff**
  - Iterate over the output and separate unmatched parts at **expect** and **actual** strings
  - Use **tablewriter** to create a table where we can beautifully print those strings side-by-side.

- **Result:**

Observations: as Keploy is a tester software it can compare even fields that are not in Rest APIS format. In that case the best thing to do is to use a generic diff comparer that I've made (https://github.com/keploy/keploy/pull/492). This generic diff comparer can calculate where the expected and actual response starts to become unmatched and highlight it.

Testing Plan

Of course each of those features will be tested so we can deliver a high-quality project that can really contribute to the future of keploy. Some of units tests we can do:

- keploy select: Test the function responsible for mark a label in the config.yaml
- keploy get: Retrieve a specific test by the ID and check if it matches with the yaml, also test the new Read() at mock.fs to see if its really returning right outputs…
- keploy run: Test if its proper calling the Test() function with appropriate data
- Keploy server log pipe: See if the function responsible for writing to a file is correctly using the LOG_FILE variable and writing the right output based on the data sent to it.

Some of integration test we can do:

- Verify that subsequent Keploy commands use the selected application as the context for their operations.
- Check that the **keploy run** command works correctly with the selected application from the config.yaml file.

Also we can do some tests using Keploy itself as in the third part we'll be adding new handlers and Keploy can do a great job there.

Most of the tests need to be written after the tasks are finished because integration tests will of course rely on other already existent features. This is why at the

beginning the best thing to do is just to write small unit tests, and after it we can start to work with more robust tests.

About documentation and readability

Every new function added under this project will be well documented and refactored through the course of it to be the most readable possible, helping other programmers that will one day read the code doesn't suffer in figure out what the function does.

Project Plan

I'll be working constantly with Keploy and there's no plan for vacation in the middle. If for some reason something happens and I need to stay 2 or 3 days off I'll notify a mentor as soon as possible and work more hours a day to make up for days I'll not be able to work.

About the project plan, given the fact the program has 10-12 weeks I'll give myself 9 weeks to do the entire project. Four weeks for the pre-midterm and four weeks for the post-midterm.

At the pre-midterm I'll be working mostly on part 1 and it should be finished till the end of it as it's the most difficult-to-implement part because there'll be new 6 commands and each of them needs to be well integrated and tested within Keploy.

The next half (post-Midterm) I'll be working on both part 2, 3, 4 (rename tests, add comments, pipe log, test-report and also in the log diffs visualization) and they should be done very fast, especially part 2 where we'll just rename the tests and add comments on it which isn't so time-consuming.

Project Plan - Preliminary Plan:

Week Number	Start Date	End Date	Tasks to be completed
Week 1	05/22	05/28	Working to implement keploy new command and config yaml file
Week 2	05/29	06/04	Working to implement select, simple unit tests
Week 3	06/05	06/11	Implement get and run test case
Week 4	06/12	06/18	Continue implementing run test case as well keploy help, keploy start (they are implemented very fast) and write some tests to ensure last features are working
Week 5	09/19	06/25	Refactor yaml test-cases, add comments and rename it, as well start with the logging pipe

Week 6	06/26	07/02	Continue with the logging pipe work and start to work on the test-report static link
Week 7	07/03	07/09	End test-report static link and work with the log diffs
Week 8	07/10	07/16	End the work with log diffs and unit tests new features
Week 9	07/17	07/23	Refactor to a more clean code, document these features and write some integration tests as well keploy tests
Submission	07/24	07/30	Evaluate the work and progress done as well evaluate mentor-contributor communication.

Major Milestones

I divided the project into 4 parts because each feature inside each part is related to each other but the parts itself have different features and are not so relatable, so completing one of them is a milestone for the project because it means a complete task is done independently of the others. The milestone are when these projects will end and they will have approximately these dates:

1 - **Keploy support of commands:** 06/12

1 - **Keploy yaml tests refactor:** 06/18

3- **New output features:** 07/03

4- **Log diff viewer:** 07/10

Additional Information

I'm the kind of punctual person and disciplined, in fact as I said I've been self-taught for years and I never passed more than 1 week where I was not studying programming or something related. Even if I see the thing is getting complicated and I would need to be fast to end some task in a given time I would spend the entire day and night wake up to solve this thing in the given time, especially if it is a project that I work for and was given for me real tasks to accomplish, like I'll do with Keploy.

As fifteen I developed a game using C# and Unity. I was newbie at this epoch in programming but I gave myself till the end of the year to end this game this would give 2 months but it was more hard than I though and I start to work more in it like 5 / 6 hour a day and at the end I finished it a bit more late than I thought but making this project make me have a vision of how to manage time and makes reliable goals. Unfortunately I didn't find the game at play store and I hadn't used any versioning tool because it was too heavy, I just have a backup saved on a pendrive.

Last year I worked with Jest and Express just to know how they worked and I ended up learning about mocks, unit tests and improved my knowledge of rest apis. Nothing too fancy. I have the links of these projects on my old github account but they are just a draft for knowledge reasons and not a real project. After it I decided to go back to Go and learn how to make the same on it. Also at university I'm improving my abilities of team work as almost all computer science schoolworks is with a team.

What to expect from your mentor (and what your mentor expects from you)

If you are selected to GSoC with Keploy, you can expect the following:

- We recognize that the goals may change during the project, and the mentors will accept modifications to the goals at any time. But they are also expecting to see the reasonable effort go into the initial project timeline. Any changes to your goals or plan are expected to be immediately communicated to your mentor.
- The scope of the project might change so as to fit in the duration of GSoC
- Your mentor will establish a weekly, synchronous check-in with you.
- In addition to that check-in, your mentor will discuss with you any specific status updates or any other regular communication they expect from you as well as which methods they prefer for documentation and collaboration (Google Docs, wiki, etc.).
- The project plan and timeline you set forth in your application will also form a significant part of your midterm and final evaluations.

What to expect from GSoC at Keploy

We want you to have a productive, engaging summer. To that end:

- We will schedule several events throughout the summer where you can interact with other GSoC students and the rest of the Keploy community.
- You will have the opportunity to present your work to this broader community.