

# GSOC'25 Proposal - Keploy

## TestSuite Idempotency Checker

### Personal Details

Name: Ahmed Mamdouh

Course: Bachelor's of Computer Science (Third Year).


Email: [ahmed.mamdouh8840@gmail.com](mailto:ahmed.mamdouh8840@gmail.com)

Github: <https://github.com/AHMED-D007A>

LinkedIn: <https://www.linkedin.com/in/ahmed-mamdouh-884805261/>

Phone: +201158862922

Current Country: Egypt

Link to Resume / CV:  Ahmed\_Mamdouh-resume\_0.pdf

### About Yourself

1. Please describe yourself, including your development background and specific expertise.

- I am a third year computer science undergraduate, I have been studying computers and software for about three years now. I have interest in Back-end development, Development tools and Web applications. For the last one and half years I have been preparing myself for being a Back-end developer. Before that I was focusing on basics (DS and algorithms, problem solving, OOP, Design Patterns, ...). Also wrote programs in (C, C++, JS, Python, Java, C#). I started learning go lang about a year ago. I really loved it for being simple and easy to understand and write. I have built small to medium projects with it and I really enjoyed it. For further self improvement I searched for open source to contribute to further enhance my knowledge and skills.

2. Why are you interested in the Keploy project(s) you stated above?

- Throughout my learning journey I learned about REST APIs and read a lot of articles and sections in books on standards and best practices one of the topics is the idempotency, when I first read about it, it wasn't really clear to me, when I started contributing to Keploy it was on the idempotency issue, I found myself reading more about it and searching giving me more understanding of how it will benefit me. So my interest in this project came from finding myself learning more about REST APIs that I have already read and learned about . I found that contributing to this project will give me mastery/learning that I will not get by building apps and reading on my own.

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?

- No, I haven't participated in an open-source project before.  
As I have stated in question no.2 I will get more learning/knowledge that I won't get on my own or from online courses. Also after my third year in my faculty I am obligated to have a certain number of hours as an internship in the summer and I find Keploy a great opportunity for that.

## Commitment

1. Are you planning any vacations during the GSoC period?

- No, I do not.

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

- Just the first two or three weeks of the starting coding period (June) may intersect with my finals exams, other than that I will be fully available.

3. Do you have any other employment during the GSoC period?

- No, I do not.

4. How many hours per week do you expect to work on the project and what hours do you tend to work?

- For the first two/three weeks I may be able to put two hours per day ( $2 * 7 = 14$ ), After that I will be fully available I can put six or more hours per day ( $6 * 7 = 42$ ).

## Contributions so far

- PRs merged and Unmerged
  - [#2540](#) , It was my first ever open source contribution on the GET idempotency checker issue. I tried after playing with the code for a while and came with not a very clean way. I just inserted the code where I found the test cases were being processed and saved, closed for a better one.
  - [#2581](#) , I made this as a prototype for my idea as I have found that it had many changes despite being a simple idea, and made a README for what does. I wanted to continue on it but it will get more complicated and I needed feedback.
  - [#2609](#) , As I learn about the Keploy codebase, I always come across these TODO comments. I have found some in the templatize.go file, it required simple changes which I think will not break the app.
  - [#135 \(samples-go\)](#) , It is a sample app I used for testing my code. I built it to be simple to modify and add more features from other APIs and public ones also to get a grasp on how my modifications on Keploy perform.
  - [#2628](#) , supports multiple docker-compose files and custom paths.
  - [#2630](#) , support for multiple Docker networks in container setup. Came across it working on a way for fixing errors in my request replayer addition.
- Issues, and bugs found
  - [#2631](#) , Feature for supporting multiple docker networks and compose files.
- Documentation contributions
- Other contributions

## Proposal

### Overview

- A brief overview of your proposed solution and the objectives to achieve with the project.
  - I have been trying the simple solution of replaying/resending the request of the test case multiple times, for how many times the request will be replayed will be decided by the user of the application via a config file and more configurations on that will also be provided. By comparing and operating over these responses I can decide what to be ignored and what to leave to the user to ignore or change.
  - For other noise that leads to flaky tests like session related data, what I have in mind is to have all common one saved and check them with the matched requests then give a user warning about it and make it even interactive by giving a prompt the user to decide to ignore them or added the an config file that will be updated before running tests, Also I can detect where these data are generated in a testset like a login request that gives a token and update all requests after it with this token.

- By providing a report file containing how it operates over these test cases I can then use a basic report template using Allure or Extent reports and using any python lib if I needed to have a cli-diff viewer.
- Programming languages, tools and technologies planned to be used.
  - Golang.
  - Python or Bash if needed to replay the requests outside the app to avoid docker network problems and cli-diff.
  - Allure or Extent reports.

## Detailed

### Initial Implementation Approach

After analyzing the codebase, the implementation starts in `pkg/service/record/record.go`. It will be executed within a Go routine immediately after retrieving the `testCase` from the `frames.Incoming` channel.

Since this feature requires multiple functions and report file generation, a dedicated structure, `idempotencyDB`, has been created (name subject to change). This structure will function similarly to `testDB` and `mockDB` and will be passed to the record service. This modular approach ensures that changes remain confined to a single directory dedicated to the newly added feature.

To enable easy integration with other services beyond the record service, `idempotencyDB` will be added to `cli/provider/common.go`, `cli/provider/core_service_linux.go`, and other relevant services as needed.

### Directory Structure

The new feature will be housed under the `pkg/platform` directory. This directory will contain all idempotency and noise detection-related functions. An interface will be created for the record service to call `idempotencyDB` functions and pass `testCase` objects to it.

### Feature Breakdown

The feature is divided into two primary components:

1. **Idempotency Check and Report Generation**
2. **Noise Detection and Noise Configuration**

#### 1. Idempotency Check and Report Generation

## Logic Behind Idempotency Check

- The HTTP request (`httpReq`) of the `testCase` is taken and modified by adding a custom header (`Idem-Replayer`).
- All incoming requests from `frames.Incoming` are checked for this header:
  - If found → The request is ignored.
  - If not found → The request is inserted as a normal test case.
- The custom header contains a value representing the replay number (1 for the first replay, 2 for the second, etc.). This aids logging and debugging.

## Sending Replays to the User API

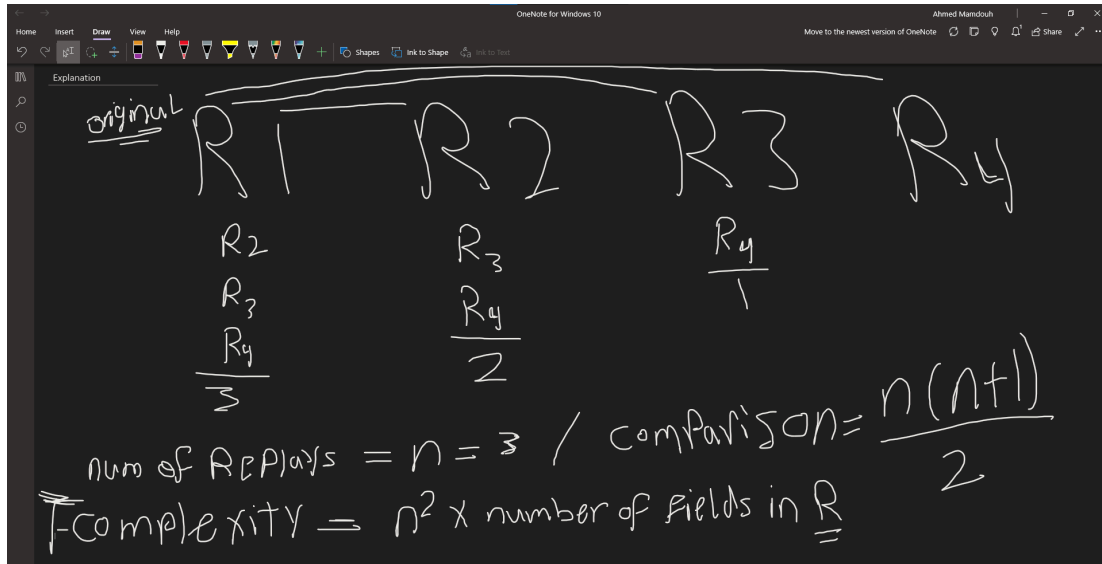
- A client sends the replayed requests to the user's API in a loop.
- The number of replays will be configurable by the user.
- A potential issue was found when testing the prototype inside Docker networks. The replay request behavior differs between internal and external networks. Keploy may or may not provide a hostname inside the Docker network.
- If Keploy does not support this, alternative solutions include using a Python or Bash script to send the requests.

## Storing Replays in `irrTestCase`

- A new type `irrTestCase` will be created to store:
  - The original `testCase`
  - The number of replays
  - The responses from the replayer
- This structure will simplify report generation and allow easy data passing between functions.

## Response Comparison and Noise Detection

- A function `compareResponses` will extract the original and replayed responses from `irrTestCase`.
- The responses will be flattened using `testdb.FlattenHttpResponse` and stored in a map for comparison.



- A **diffMap** will track differences across responses (e.g., **body.ts : 6** means the field changed across all replays  $\frac{n \times (n+1)}{2}$ ).
- Fields that are consistently different across all responses will be marked as **noise**.
- Fields that change inconsistently will trigger a **warning**.
- The user may be prompted (via a new Go routine) to decide whether to mark inconsistent fields as noise.

## Handling Non-Idempotent Methods

- For non-idempotent methods like **POST** and **PATCH**, Keploy assumes the user has implemented idempotency keys.
- Research has been conducted on best practices for implementing idempotency keys in APIs.

## Report Generation

- **irrTestCase**, detected noise fields, and warnings about inconsistencies will be saved in a report file.
- This report will be compatible with external reporting tools or the CLI-diff feature mentioned in the project description.

## 2. Noise Detection and Noise Configuration

### Prototype Status

- No working prototype has been developed for this yet.
- A partial, non-functional noise configuration method exists in a previous PR.

## Configuration Structure

- A configuration file will store:
  - Common headers/fields that frequently cause flaky tests.
  - Dynamic values (e.g., timestamps, session tokens) that should be ignored unless found in the test set.

## Functionality

- A function will detect and update dynamic fields found in test cases.
- A function will detect session-related fields (e.g., **Authorization** headers) and prioritize them in testing.
- A function will apply session-related fields to test cases before each test run.
- A Go routine will prompt the user to mark detected dynamic fields as noise or update them.

## Execution Flow

1. After performing idempotency checks, the system detects dynamic and session-related fields.
2. The detected fields are written to the configuration file.
3. The user receives a warning about dynamic fields and can choose to ignore or update them.
4. Session-related fields are extracted from requests (e.g., JWT tokens in login test cases).
5. Session-related fields are marked as prioritized in the configuration file, with an encoded test case reference.
6. During testing:
  - The configuration file is read.
  - Noise fields are ignored.
  - The user is prompted to update certain fields.
  - Prioritized fields are updated before test execution.

## Enhancements for User Experience

- Each warning will include suggestions tailored to the detected inconsistencies, improving the user experience.

## Handling Different Response Types

- Before operating on a response, its type must be identified.
- JSON responses will be handled using **FlattenHttpResponse**.
- Other response types (e.g., HTML) will require a new parsing function.
- This ensures Keploy can handle diverse API responses effectively.

## Report Generation Tools

- The project description specifies using **Allure** or **Extent** reports.
- Research will be conducted to determine the best approach for integrating these tools.

## CLI-Diff Implementation

- No previous experience with CLI-diff tools.
- Research will be done to decide whether to build a custom tool or integrate an existing one.

## Race Condition Issue

- A race condition occurs when Keploy is terminated immediately after sending a request.
- The issue arises in `pkg/core/hooks/conn/socket.go` `ListenSocket` due to a closed test case channel (`t`).
- Solution: Wait for the replayer to finish before closing the channel.

## Project Plan

- I want to state that the first 2 to 3 weeks I may not be fully available due to my final exams of the year. It usually takes place from the end of May to the start of June.
- Pre-Midterm
  - Week 1 (June 2 - June 8) - Minimal Work  
Goal: Light tasks to stay engaged while balancing other commitments.
  - Week 2 (June 9 - June 15) - Minimal Work  
Goal: Continue light tasks and prepare for full availability.
  - Week 3 (June 16 - June 22)  
Goal: Implement the core functionality for idempotency checking.
  - Week 4 (June 23 - June 29)  
Goal: Enhance noise detection and reporting.
  - Week 5 (June 30 - July 6)  
Goal: Finalize the idempotency checker and noise detection.
  - Week 6 (July 7 - July 13)  
Goal: Prepare for the midterm evaluation.
- Post-Midterm
  - Week 7 (July 14 - July 20)  
Goal: Implement reporting using Allure or Extent.
  - Week 8 (July 21 - July 27)  
Goal: Implement CLI-diff functionality.



- Week 9 (July 28 - August 3)  
Goal: Improve user experience and documentation.
- Week 10 (August 4 - August 10)  
Goal: Optimize and test the implementation.
- Week 11 (August 11 - August 17)  
Goal: Prepare for the final evaluation.
- Week 12 (August 18 - August 25)  
Goal: Buffer week for unforeseen delays or additional improvements.

## Project Plan - Preliminary Plan:

### (Community Bonding Period)

- Understand Existing Codebase.
- Discuss with the mentor the best way to go about the implementation.

Week Number	Start Date	End Date	Tasks to be completed
Week 1	June-2	June-8	<input type="checkbox"/> Set up the idempotencyDB directory structure. <input type="checkbox"/> Begin drafting the logic for replaying HTTP requests with a custom header (Idem-Replayer).
Week 2	June-9	June-15	<input type="checkbox"/> Finalize the logic for replaying HTTP requests. <input type="checkbox"/> Start implementing the compareResponses function for basic response comparison. <input type="checkbox"/> Begin researching noise parameters and detection strategies.
Week 3	June-16	June-22	<input type="checkbox"/> Complete the implementation of the compareResponses function. <input type="checkbox"/> Add logging and warnings for inconsistencies detected during idempotency checks. <input type="checkbox"/> Begin implementing a basic configuration system for noise detection (e.g., dynamic fields, session-related data).
Week 4	June-23	June-29	<input type="checkbox"/> Improve the noise detection system to handle session-related data (e.g., JWT tokens). <input type="checkbox"/> Add functionality to prioritize and update session-related fields in the test set. <input type="checkbox"/> Start generating a basic report file with detected noise and inconsistencies.
Week 5	June-30	July-6	<input type="checkbox"/> Complete the implementation of the idempotency checker and noise detection. <input type="checkbox"/> Add user prompts for dynamic fields and session-related data.
Week 6	July-7	July-13	<input type="checkbox"/> Document the implemented features and their usage. <input type="checkbox"/> Fix any bugs or issues identified during testing. <input type="checkbox"/> Prepare a demo for the midterm evaluation.
Week 7	July-14	July-20	<input type="checkbox"/> Research and integrate Allure or Extent reporting into Keploy. <input type="checkbox"/> Generate detailed reports for idempotency checks and noise detection.

Week 8	July-21	July-27	<input type="checkbox"/> Research and implement a CLI-diff tool to visualize differences in test responses. <input type="checkbox"/> Integrate the CLI-diff tool with the existing Keploy CLI.
Week 9	July-28	August-3	<input type="checkbox"/> Add user-friendly prompts and warnings for noise and inconsistencies. <input type="checkbox"/> Write comprehensive documentation for the new features. <input type="checkbox"/> Update the sample applications to demonstrate the new functionality.
Week 10	August-4	August-10	<input type="checkbox"/> Optimize the performance of the idempotency checker and noise detection. <input type="checkbox"/> Conduct extensive testing to ensure stability and reliability. <input type="checkbox"/> Fix any remaining bugs or issues.
Week 11	August-11	August-17	<input type="checkbox"/> Finalize all documentation and demos. <input type="checkbox"/> Prepare a presentation for the final evaluation.
Week 12	August-18	August-24	<input type="checkbox"/> Submit the final code and deliverables. <input type="checkbox"/> Address any last-minute issues or feedback from the mentor. <input type="checkbox"/> Polish the final deliverables and documentation.
<b>Submission</b>	August-25	September-1	<input type="checkbox"/> Final week: GSoC contributors submit their final work product and their final mentor evaluation.

## Major Milestones

- Midterm Evaluation (July 14):
  - Complete implementation of the idempotency checker and noise detection.
  - Generate basic reports for idempotency checks and noise detection.
  - Demonstrate the working prototype.
- Reporting and CLI-diff Implementation (July 27):
  - Integrate Allure/Extent reporting and CLI-diff functionality.
  - Demonstrate the enhanced reporting and visualization features.
- Final Evaluation (August 25):
  - Deliver a stable and fully documented implementation.
  - Present the final project to the Keploy community.

## Additional Information

- Why I Can Complete This Project:
  - I have a strong foundation in Golang and have already contributed to Keploy by addressing idempotency issues.
  - I am passionate about learning and solving problems, as shown by my self-driven projects and contributions.

- I am committed to dedicating sufficient time (40+ hours/week after the first two weeks) to ensure the success of this project.
- Preference:
  - I am not applying to any other organization, This is my preferred project and only project I am applying for with Keploy.

## 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 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 outlined 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.