



# **MULTIPLE SERVICE USING KEPLOY (API CHAINING)**

## Project Proposal

**Mentors:**

Gourav Kumar

Shivam Sourav Jha

**By:**

Ahmed Mohamed Ahmed Lotfy

## ***Table of Contents***

<b>1. Overview</b>	<b>3</b>
<b>2. Goals &amp; Ideas</b>	<b>3</b>
<b>3. Personal Details</b>	<b>3</b>
<b>4. About Yourself</b>	<b>4</b>
<b>5. Commitment</b>	<b>6</b>
<b>6. Contributions so far</b>	<b>6</b>
<b>7. Qualification Tasks:</b>	<b>7</b>
Proposal	7
Overview	7
Detailed	8
Project Plan - Preliminary Plan:	13
<b>8. Major Milestones</b>	<b>14</b>
<b>9. Additional Information:</b>	<b>14</b>

## 1. Overview

This project will implement Contract Testing for Keploy, which empowers seamless interactions between services by emphasizing contract testing. This approach ensures all communications adhere to predefined expectations, boosting the reliability of interconnected systems.

Contract testing verifies interactions between microservices, applications, or systems. It guarantees proper communication through pre-defined agreements (contracts). This enables early detection of issues and reduces the complexity of end-to-end testing.

## 2. Goals & Ideas

- a. **Keploy Support:** Integrate contract testing capabilities into Keploy to enable advanced testing functionalities, enhancing its ability to validate microservice interactions effectively.
- b. **One-Go Testing:** Implement functionality within Keploy for recording tests and mocks for all related microservices simultaneously. This feature would capture both ingress and egress traffic, utilizing process identification (PID) for filtering. The primary goal is to ensure that tests and mocks remain synchronized across all services, thereby maintaining consistency and accuracy in testing environments.
- c. **Automated Change Management for API Chains:** When using Keploy with services m1, m2, and m3, consider a scenario where m2 undergoes a change, such as the addition of a new field. When this change is incorporated into m2's test cases, it could potentially disrupt the mock interactions expected by m1, necessitating updates to m1's tests to align with the new behavior. This process ensures that mocks are consistently refreshed to reflect the most current service interactions. Keploy facilitates this dynamic updating mechanism, streamlining the maintenance of test cases and mocks across services. Consequently, it simplifies end-to-end (E2E) testing, making it more efficient and reducing the complexity typically associated with testing interconnected services.

### 3. Personal Details

- **Name:** Ahmed Mohamed Ahmed Lotfy
- **Course:** Computer Engineering at Faculty of engineering Cairo University
- **Email:** ahmed.lotfy00@eng-st.cu.edu.eg
- **Github:** [Link](#)
- **LinkedIn:** [Link](#)
- **Phone:** 01097865159
- **Current Country:** Egypt
- **Link to Resume / CV:** [CV](#)

### 4. About Yourself

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

- My name is Ahmed & I'm in my last year in my college. I'm a backend engineer who worked in companies like [Ejada](#) and took internships in companies like [VMware](#), [Master Micro](#), [FGS](#).
- I took three trainings with ITI in topics "Red Hat system administration", "Flutter", "MEAN Stack".
- I worked with many programming languages like Go, C++, C, Java, Python, Javascript, Typescript and many frameworks like Angular, Expressjs, Flutter.
- **Here are the relevant college courses:**
  - ❖ - Introduction to Programming (C++)
  - ❖ - Programming Techniques (C++)
  - ❖ - Advanced Programming Techniques (Java) - Data Structures and Algorithms (C++)
  - ❖ - Design & Analysis of Algorithms (C++) - Computer Graphics (C++, OpenGL) \*
  - ❖ - Operating Systems
  - ❖ - Software Engineering
  - ❖ - Image Processing & Computer Vision (Py) - Pattern Recognition & Neural Networks (Py) \*
  - ❖ - Database Design & Management (SQL, C#) - Cryptography & Cybersecurity \*
  - ❖ - Logic Design I & II
  - ❖ - Microprocessors I & II
  - ❖ - Embedded

- |   |                               |
|---|-------------------------------|
| ❖ Systems*  | ❖ Number Theory               |
| ❖ - Computer Architecture                             | ❖ - Signal Processing         |
| ❖ Some notes:   | ❖ - Statics                   |
| ❖ - Calculus I, II & III.                             | ❖ - Dynamics                  |
| ❖ - Algebra   | ❖ - Physics I, II & III.      |
| ❖ - Linear Algebra                                    | ❖ - Mechanical Engineering    |
| ❖ - Analytic Geometry                                 | (Robotics & Thermodynamics) - |
| ❖ - Numerical methods                                 | Control Engineering*          |
| ❖ - Probability & Statistics - Discrete Mathematics - | ❖ - VLSI Design               |

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

- I'm interested in that Keploy project as Firstly, the integration of contract testing capabilities into Keploy aligns perfectly with my professional background and interests.
- I am enthusiastic about leveraging my expertise in software testing methodologies to contribute to the project's goal of validating microservice interactions effectively. By incorporating contract testing, we can not only enhance the quality of interconnected systems but also streamline the development process by identifying potential issues early on.
- The idea of implementing functionality within Keploy for recording tests and mocks for all related microservices simultaneously excites me greatly. I see immense potential in this feature to revolutionize the testing process, ensuring consistency and accuracy across different testing environments. As someone who values efficiency and precision in software development, I am eager to contribute to the development of this feature and witness its transformative impact on testing practices.

***c. 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 for OSS and I want to join Keploy this summer as contributing to an open-source project like Keploy allows me to collaborate with a diverse community of developers, mentors, and contributors from around the world. This collaborative environment fosters learning, sharing of ideas, and collective problem-solving, which are invaluable experiences for personal and professional growth.
- Participation in Keploy's open-source project offers the chance to address practical challenges in software development. By contributing to the

project, one can directly impact the reliability and efficiency of software systems, benefiting both developers and end-users. GSoC offers a platform to connect with industry professionals, mentors, and fellow students who share similar interests and goals. Through networking opportunities provided by the program, participants can build meaningful relationships, receive mentorship, and potentially explore future career opportunities in the tech industry.

## 5. Commitment

### a. Are you planning any vacations during the GSoC period?

- I have no planned absence days for the duration of the GSoC project period. If any Emergency occurs, then I will inform the Mentor.

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

- I will have my final exams for weeks 1-2.

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

- No.

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

- Time zone: UTC+02:00. I plan to work around 30-35 hours/week. I am fully committed for the whole period except for the first 3 weeks when I will have my final exams. During which, I plan to only work on the simpler tasks of the project as explained in the project schedule.
- My preferred working hours are from 9 AM to 6 PM.

## 6. Contributions so far

### a. PRs Merged:

- **#1590:** [\[feature\]: hide complete test summary in case of app not running #1590](#)
- **#1585:** [\[fix\]: bug while keploy test in user app #1585](#)

### b. PRs Unmerged:

- **#1584:** [Colorise keploy logo #1584](#)
- **#1609:** [\[fix\]: update keploy even it has the latest version #1609](#)
- **#1640:** [Addition of version argument to installation script #1640](#)
- **#1660:** [fix compare diffs in case of array of json #1660](#)
- **#1699:** [permissions for keploy folder #1699](#)

- #1702: [Validate some paths #1702](#)
- c. **Documentation contributions:**
  - #337: [\(add\): env field in debugger guide docs #337](#)
- d. **Other contributions [GSoC related issues]:**
  - #1541: [\[GSoC\]: test multiple services with keploy #1541](#)
  - #1543: [\[GSoC\]: Build an app with multiple services, demonstrating contract testing #1543](#)

## 7. Qualification Tasks:

- a. You can see my work on these issues:
  - [\[GSoC\]: Build an app with multiple services, demonstrating contract testing #1543](#)
  - [\[GSoC\]: test multiple services with keploy #1541](#)
  - [\[GSoC\]: Add support in Keploy to read from server initiated calls #1572](#)

# Proposal

## Overview

- Provide a brief overview of your proposed solution and objectives you want to achieve with your chosen project.

Overview: My proposed solution focuses on enhancing microservice testing capabilities within Keploy, a platform designed for API chaining and contract testing. By integrating advanced testing functionalities into Keploy, we aim to streamline the validation of microservice interactions and improve the reliability of interconnected systems.

## Objectives:

1. **Integrate Contract Testing:** Implement contract testing capabilities into Keploy to enable comprehensive validation of microservice interactions. This will involve defining and enforcing predefined contracts to ensure seamless communication between services.
2. **One-Go Testing Functionality:** Develop a feature within Keploy to enable recording tests and mocks for all related microservices simultaneously. This functionality will capture both ingress and egress traffic, maintaining consistency and accuracy across testing environments.
3. **Automated Change Management:** Implement automated change management for API chains to facilitate dynamic updates to tests and mocks. This will ensure that tests remain synchronized with any changes in service behaviors, simplifying maintenance and reducing the complexity of end-to-end testing.

4. Documentation and Training: Provide comprehensive documentation and training materials to support users in leveraging the enhanced testing capabilities of Keploy. This will ensure that developers can effectively utilize the platform to improve the reliability of their microservice architectures.
- Please include programming languages, tools and technologies you plan to use.
    - I will mainly focus on **Go** as it's the programming language for keploy's core.

Detailed

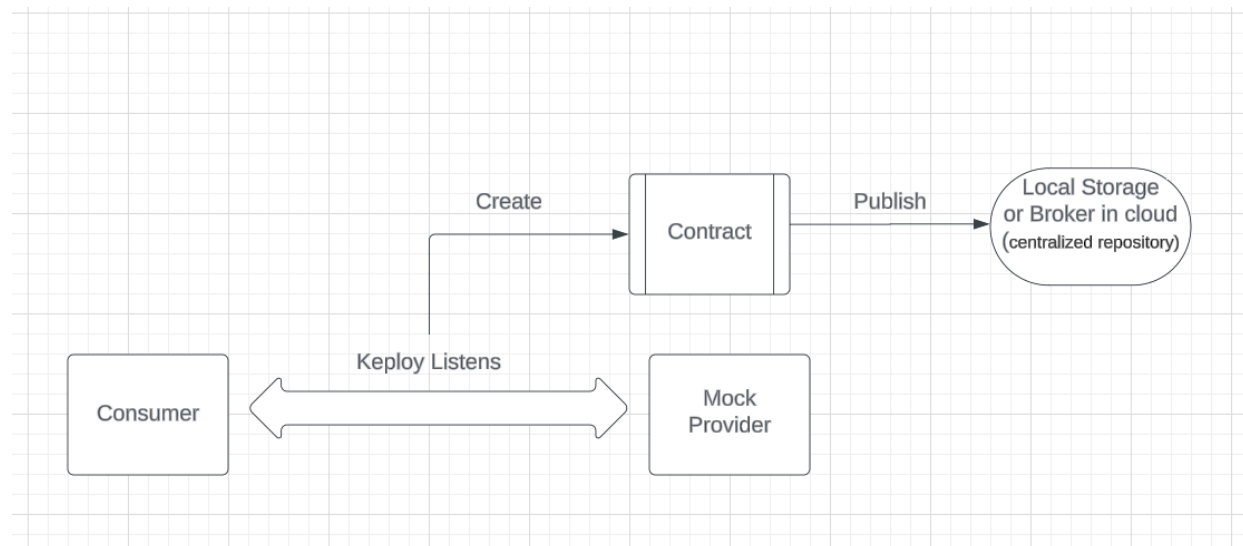
## 1. Integrate Contract Testing:

Referred to this issue: [\[GSoC\]: Build an app with multiple services, demonstrating contract testing #1543](#)

**Let's divide it into two phases:**

### 1. Producing the contract (consumer-driven contract):

- **Define Consumer Expectations:** after writing automated tests that define how the consumer service expects to interact with the provider API. These tests typically involve sending requests (e.g., GET, POST) with specific data and asserting the expected responses (status code, response body), Keploy will listen to this network interchange like what Keploy performs in record mode (outgoing and incoming channels) and create the contract instead of creating a mock and a testcase.





- **Contract Design:**

```
4 import com.fasterxml.jackson.annotation.JsonProperty;
5
6 import lombok.Data;
7 import lombok.NoArgsConstructor;
8
9 @Data
10 @NoArgsConstructor
11
12 public class Contract {
13     @JsonProperty("consumer")
14     private Participant consumer;
15
16     @JsonProperty("interaction")
17     private List<Interaction> interactions;
18
19     @JsonProperty("provider")
20     private Participant provider;
21     //create full arg constructor
22     public Contract(Participant consumer, List<Interaction> interactions, Participant provider) {
23         this.consumer = consumer;
24         this.interactions = interactions;
25         this.provider = provider;
26     }
27 }
28
29 }
```

- **Interaction Design:**

```
1 package com.example.order.models;
2
3 import com.fasterxml.jackson.annotation.JsonProperty;
4
5 import lombok.AllArgsConstructor;
6 import lombok.Data;
7 import lombok.NoArgsConstructor;
8
9 @Data
10 @NoArgsConstructor
11 @AllArgsConstructor
12 public class Interaction {
13     @JsonProperty("description")
14     private String description;
15
16     @JsonProperty("request")
17     private Request request;
18
19     @JsonProperty("response")
20     private Response response;
21 }
22
```

## ■ Request Design (may be changed later):

```
1 package com.example.order.models;
2
3 import com.fasterxml.jackson.annotation.JsonProperty;
4
5 import lombok.Data;
6
7 @Data
8
9 public class Request {
10
11     @JsonProperty("body")
12     private Object body;
13
14     @JsonProperty("headers")
15     private Headers headers;
16     @JsonProperty("URL")
17     private String URL;
18
19     @JsonProperty("method")
20     private String method;
21
22     @JsonProperty("path")
23     private String path;
24
25     @JsonProperty("query")
26     private String query;
```

## ■ Response Design(may be changed later):

```
1 package com.example.order.models;
2
3 import com.fasterxml.jackson.annotation.JsonProperty;
4
5 import lombok.Data;
6
7 @Data
8
9 public class Response {
10     @JsonProperty("body")
11     private Object body;
12
13     @JsonProperty("headers")
14     private Headers headers;
15
16     @JsonProperty("status")
17     private int status;
```

## ● Implementation Sample:

- Java code to save the contract file as a json (may be changed to yaml later).

```

// function that take response and request and create the interaction and the contract
public static void createContract([com.example.order.models.Response response, com.example.order.models.
                                Request request,String consumer,String provider]) {
    // create the interaction
    com.example.order.models.Interaction interaction = new com.example.order.models.Interaction();
    interaction.setRequest(request);
    interaction.setResponse(response);
    interaction.setDescription(description:"test interaction description");
    //check if the contract exists or not
    if(checkContractExist(consumer, provider)){
        com.example.order.models.Contract contract = loadContract(consumer, provider);
        contract.getInteractions().add(interaction);
        saveContract(contract);
    }
    else{
        // if the contract does not exist, create a new contract and add the interaction to it
        com.example.order.models.Contract contract = new com.example.order.models.Contract();
        contract.setConsumer(new com.example.order.models.Participant(consumer));
        contract.setProvider(new com.example.order.models.Participant(provider));
        List<com.example.order.models.Interaction> interactions = new ArrayList<com.example.order.models.Interaction>();
        interactions.add(interaction);
        contract.setInteractions(interactions);
        // save the contract
        saveContract(contract);
    }
}

public static Boolean checkContractExist(String consumer, String provider) {
    File file = new File("./"+consumer+"_"+provider+".json");
    return file.exists();
}

```

```

public static Boolean saveContract(com.example.order.models.Contract contract) {
    // Create an ObjectMapper to save class into it
    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.enable(SerializationFeature.INDENT_OUTPUT);

    try {
        // Convert the Contract object to a JSON string
        String contractJson = objectMapper.writeValueAsString(contract);
        // Write the JSON string to a file using try-with-resources
        try (FileWriter fileWriter = new FileWriter("./" + contract.getConsumer().getName() + " " +
                                                    contract.getProvider().getName() + ".json")) {
            fileWriter.write(contractJson);
        }
    } catch (IOException e) {
        // Handle the exception according to your needs
        e.printStackTrace();
        return false;
    }
    return true;
}

```

```

//function to load the contract
public static com.example.order.models.Contract loadContract(String consumer, String provider) {
    // Create an ObjectMapper to read the class from the file
    ObjectMapper objectMapper = new ObjectMapper();
    com.example.order.models.Contract contract = null;
    try{
        String workingDirectory = System.getProperty("user.dir");
        System.out.println("Current working directory: " + workingDirectory);
        File jsonFile = new File("/home/ahmed/Desktop/GSOC/Keploy/Issues/issue1541/8085_8084.json");

        if (jsonFile.exists()) {
            // Read the content of the file and deserialize it into Contract class
            contract = objectMapper.readValue(jsonFile, valueType:com.example.order.models.Contract.class);
            System.out.println("Contract loaded successfully from: " + jsonFile.getAbsolutePath());
        } else {
            System.out.println("File not found: " + jsonFile.getAbsolutePath());
        }
    } catch (IOException e) {
        // Handle the exception according to your needs
        e.printStackTrace();
    }
    return contract;
}

```

## 2. Verification with Provider:

- a. I will make a feature on Keploy that sets up mock consumer interactions that mimic the contract (similar to the replay feature on Keploy). This allows them to verify their implementation adheres to the agreed-upon behavior before deployment.

b. Matching will be done by the **reflect** package as done on Kepteloy.

```

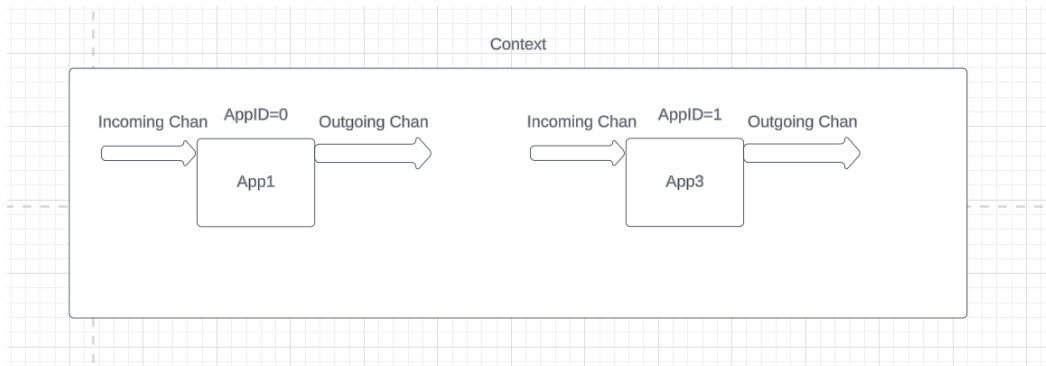
25  "go.kepteloy.io/server/v2/pkg"
26  "go.kepteloy.io/server/v2/pkg/models"
27  "go.kepteloy.io/server/v2/utlis"
28  )
29
30  type ValidatedJSON struct {
31      expected interface{} // The expected JSON
32      actual   interface{} // The actual JSON
33      isIdentical bool
34  }
35
36  type JSONComparisonResult struct {
37      matches bool // Indicates if the JSON strings match according to the criteria
38      isExact bool // Indicates if the match is exact, considering ordering and noise
39      differences []string // Lists the keys or indices of values that are not the same
40  }
41
42  func match(tc *models.TestCase, actualResponse *models.HTTPResp, noiseConfig map[string]map[string][]string, ignoreOrdering
43      bodyType := models.BodyTypePlain
44      if json.Valid([]byte(actualResponse.Body)) {
45          bodyType = models.BodyTypeJSON
46      }
47      pass := true
48      hRes := &models.HeaderResult{}
49
50      res := &models.Result{
51          StatusCode: models.IntResult{
52              Normal: false,
53              Expected: tc.HTTPResp.StatusCode,
54              Actual: actualResponse.StatusCode,
55          },
56          BodyResult: []models.BodyResult{
57              Normal: false,
58              Type: bodyType,
59              Expected: tc.HTTPResp.Body,

```

## 2. One-Go Testing:

### 1. First Solution:

a. **Base Case:** we have two services which doesn't depend on each others and we want to record them:



- ii. Each service or let's call it App has its own **PID** to capture data (network traffic, function calls) related to it, this done on Hook.
- iii. Each App has its recorder with PID (sequence number) which is passed to hook instead of this.

```
//load hooks
err = c.Hooks.Load(hookCtx, id, HookCfg{
    AppID:    id,
    Pid:      0,
    IsDocker: isDocker,
    KeployIPv4: a.KeployIPv4Addr(),
})
```

iv. Incoming Channel (**incomingChan**):

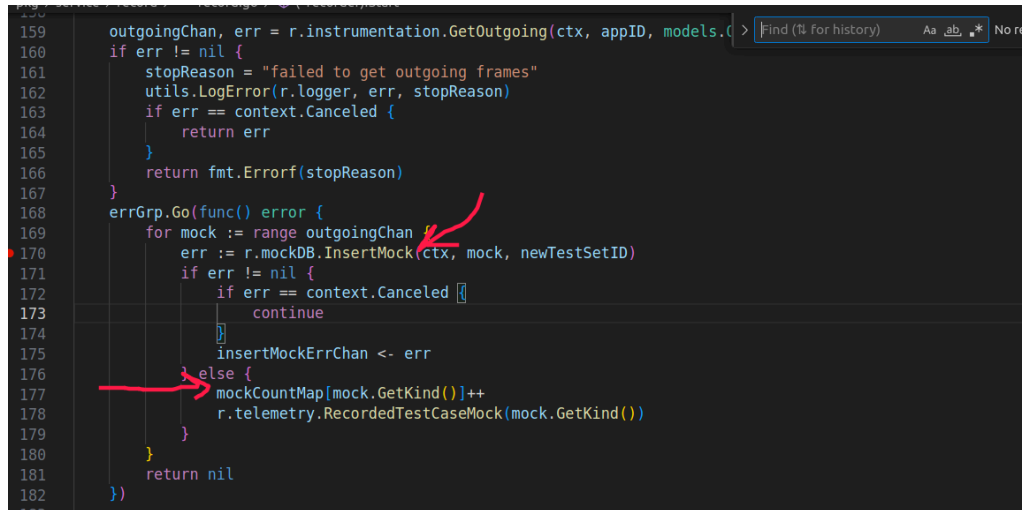
- **Content:** It now receives only those test cases generated by the specific microservice under test, as filtered by PID. This eliminates irrelevant test cases from other microservices, ensuring a focused test dataset.

v. Outgoing Channel (**outgoingChan**):

- **Content:** It now receives only those mocks intended for the specific microservice under test, as filtered by PID. This prevents mocks from being applied to unintended services, ensuring accurate and isolated testing.

- vi. Functions `InsertTestCase` and `InsertMock` are required to be synchronized as they will be called many times concurrently, so we will use **Mutexes** to ensure only one goroutine can access shared data or resources at a time.

- vii. Also the shared data structures among apps like **MockCountMap** must be synchronized (using **sync.Map** or using **sync/atomic** for atomic operations).



The screenshot shows a Go code editor with the following code snippet (lines 159-182):

```
159 outgoingChan, err = r.instrumentation.GetOutgoing(ctx, appID, models.C
160 if err != nil {
161     stopReason = "failed to get outgoing frames"
162     utils.LogError(r.logger, err, stopReason)
163     if err == context.Canceled {
164         return err
165     }
166     return fmt.Errorf(stopReason)
167 }
168 errGrp.Go(func() error {
169     for mock := range outgoingChan {
170         err := r.mockDB.InsertMock(ctx, mock, newTestSetID)
171         if err != nil {
172             if err == context.Canceled {
173                 continue
174             }
175             insertMockErrChan <- err
176         } else {
177             mockCountMap[mock.GetKind()]++
178             r.telemetry.RecordedTestCaseMock(mock.GetKind())
179         }
180     }
181     return nil
182 })
```

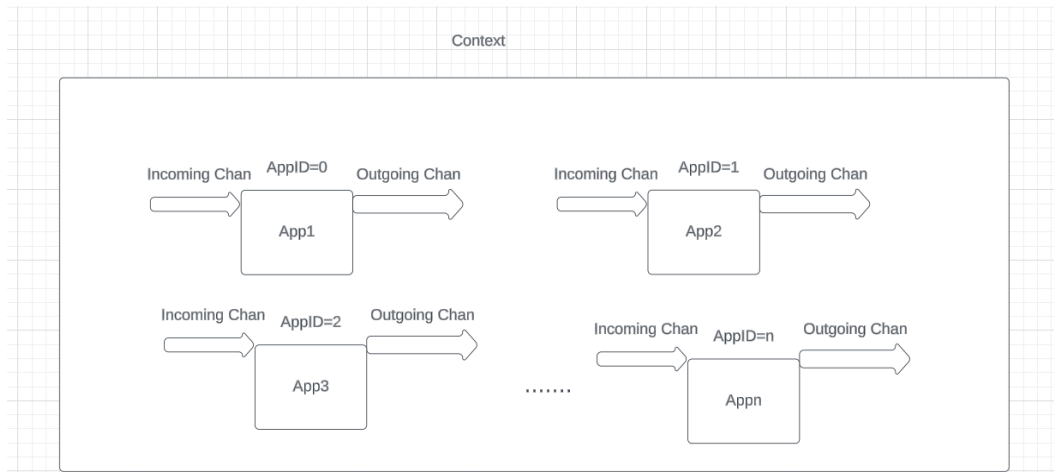
Red annotations include a curved arrow pointing from line 169 to line 170, and a straight arrow pointing from line 176 to line 177.

**b.If there is two services which one of them is dependent on the other:**

- i. Race condition on writing mocks and test cases will be solved with pid filtering & synchronization as lets say that s1 requested s2 so mock for s2 will be generated and test case also and on the same time the incoming channel of s2 would capture test case for s2 and generate it which could make a race condition but the **Mutexes** on both functions will solve this.

**c. If Number of services >2:**

- i. No problem would occur if we solved for  $n=2$



**d. In case of testing multiple apps:**

- i. Each app will interact with its test cases and mocks only. We will edit the `Start` and `BootReplay` functions to run multiple apps like what we did in record.

- e. Sample of code for recording (not necessary to be the actual implementation).

```
2
3 type App struct {
4     ID      uint64 // Unique identifier for the application
5     Command string // Command to execute the application
6     TestSetID string // Identifier for the test set associated with the application
7 }
8
9 func (r *recorder) StartRec(ctx context.Context, apps []App) error {
10     errGrp, ctx := errgroup.WithContext(ctx)
11     ctx = context.WithValue(ctx, models.ErrGroupKey, errGrp)
12
13     for _, app := range apps {
14         app := app // Capture range variable
15         errGrp.Go(func() error {
16             return r.recordApp(ctx, app)
17         })
18     }
19
20     return errGrp.Wait()
21 }
```



```

333 func (r *recorder) recordApp(ctx context.Context, app App) error {
334     var stopReason string
335     defer func() {
336         if stopReason != "" {
337             utils.LogError(r.logger, errors.New(stopReason), "Recording stopped due to error")
338         }
339     }()
340
341     // Setting up the environment for recording
342     appID, err := r.instrumentation.Setup(ctx, app.Command, models.SetupOptions{})
343     if err != nil {
344         stopReason = "failed setting up the environment"
345         return fmt.Errorf(stopReason+": %w", err)
346     }
347
348     // Starting the hooks and proxy
349     err = r.instrumentation.Hook(ctx, appID, models.HookOptions{Mode: models.MODE_RECORD})
350     if err != nil {
351         stopReason = "failed to start the hooks and proxy"
352         return fmt.Errorf(stopReason+": %w", err)
353     }
354
355     // Fetching test cases and mocks from the application and inserting them into the database
356     incomingChan, err := r.instrumentation.GetIncoming(ctx, appID, models.IncomingOptions{})
357     if err != nil {
358         stopReason = "failed to get incoming frames"
359         return fmt.Errorf(stopReason+": %w", err)
360     }
361
362     errGrp, _ := errgroup.WithContext(ctx)
363
364     // Handle incoming test cases
365     errGrp.Go(func() error {
366         for testCase := range incomingChan {
367             err := r.testDB.InsertTestCase(ctx, testCase, app.TestSetID)
368             if err != nil {
369                 return err // Exit if error occurs

```

```

364         // Handle incoming test cases
365         errGrp.Go(func() error {
366             for testCase := range incomingChan {
367                 err := r.testDB.InsertTestCase(ctx, testCase, app.TestSetID)
368                 if err != nil {
369                     return err // Exit if error occurs
370                 }
371                 r.telemetry.RecordedTestAndMocks()
372             }
373             return nil
374         })
375
376         // Handle outgoing mocks
377         outgoingChan, err := r.instrumentation.GetOutgoing(ctx, appID, models.OutgoingOptions{})
378         if err != nil {
379             stopReason = "failed to get outgoing frames"
380             return fmt.Errorf(stopReason+": %w", err)
381         }
382
383         errGrp.Go(func() error {
384             for mock := range outgoingChan {
385                 err := r.mockDB.InsertMock(ctx, mock, app.TestSetID)
386                 if err != nil {
387                     return err // Exit if error occurs
388                 }
389                 r.telemetry.RecordedTestCaseMock(mock.GetKind())
390             }
391             return nil
392         })
393
394         // Wait for all operations to complete
395         if err := errGrp.Wait(); err != nil {
396             stopReason = "error while recording app"
397             return fmt.Errorf(stopReason+": %w", err)
398         }
399
400         return nil
401     }

```

## 2. Automated Change Management for API Chains:

### 1. First solution:

- We can do this by the concept of contracts, let's take an example:

Imagine you have an e-commerce application where two services interact:





- **Product Service (m1)**: This service retrieves product information from a database and sends it to the client application.
- **Inventory Service (m2)**: This service manages product stock levels and communicates them to the Product Service.
- If **Product Service** requests **Inventory Service** to get products test case and mock are generated but the problem is when there is a major change happen in **Inventory Service** like adding a field this change won't impact the interaction with **Product Service** as there is an existing mock for **Product Service (which no longer reflects the new response format from the Inventory Service)** and it thought that everything is ok but when doing e2e testing between the two services it will fail.
- The solution is when there is any change in the provider service like **Inventory Service** in our case it will unverify all the contracts with the consumers so that they will be notified for the change in the mocks and change their test cases accordingly or we can implement a feature to automatic change the test-cases to match the expected mock by getting the different attributes by matching techniques like reflect and update the test-case.

- Sample of what pact does when there is any change in mock and there is a service dependent on it:
  - This is the contract generated between product (consumer) and retailer( the provider)

API BROWSER

## Pacts

Search

Consumer ↕	Provider ↕		Latest pact published	Webhook status	Last verified	
Example App	Example API	 	1 day ago	<a href="#">Create</a>	1 day ago	...
product	retailer	 	35 minutes ago	<a href="#">Create</a>	less than a minute ago	...





« 1 »  
2 of 2 pacts

- So any change any field name in provider test will cause the contract to be unverified

API BROWSER

## Pacts

Search

Consumer ↕	Provider ↕		Latest pact published	Webhook status	Last verified	
Example App	Example API	 	1 day ago	<a href="#">Create</a>	1 day ago	...
product	retailer	 	about 1 hour ago	<a href="#">Create</a>	1 minute ago	...

« 1 »  
2 of 2 pacts

- And consumer is notified by it's outdated testcase

```
2024-02-28 09:14:39.492 INFO 1500 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-02-28 09:14:39.518 INFO 1500 --- [o-auto-1-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 25 ms

returns a response which
has status code 200 (OK)
includes headers
"Content-Type" with value "application/json" (OK)
has a matching body (FAILED)

Failures:

0) Get item details for item id returns a response which has a matching body
$ -> Expected price=1000 but was missing

Diff:

    "name": "iPhone",
-   "price": 1000
+   "price2": 1000.0
}

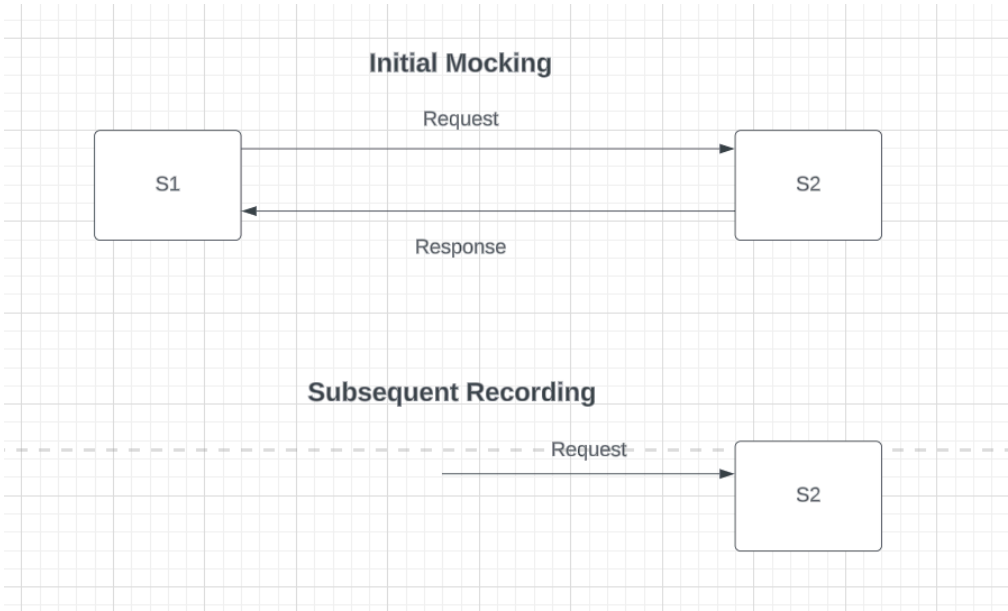
0 - $ -> [{mismatch=Expected price=1000 but was missing, diff=    "name": "iPhone",
-   "price": 1000
+   "price2": 1000.0
}]
```

- The contract must be somewhere shareable between consumer and provider services.

## 2. Second Solution:

- While recording if there any incoming requests we will search for all mocks contains this service as a provider ( we could add a **dependent\_service** parameter in the mock) and match the response with the expected response on the mock:
  - If Not matched, update it.
  - If matched , update the timestamp of the updated mock.
  - Example: initial recording between s1 and s2 causing generation of s2 mock for s1.Any subsequent recording will cause to check

the validate all dependent mocks



**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 1-3	May 27	June 16	<b>Understanding more about the project and the best practices done in go.</b> Note: I will have my final exams for weeks 1-3. That is why I picked simple tasks for them. (~ 8-10 hours/week).
Week 4-6	June 17	July 7	<b>Integrate Contract Testing into Keploy</b>
<b>First Evaluation</b>			

Week 7	July 8	July 14	Testing, Bug fixes, and Documentation
Week 8-11	July 15	August 11	Implement "One-Go Testing" feature
Week 11-13	August 12	August 25	Implement "Automated Change Management for API Chains" & Testing, Bug fixes, and Documentation
<b>Final Evaluation for Medium Projects</b>			

## 8. Major Milestones

- Week **6** : Finishing the contract testing.
- Week **11**: Implementation of the "One-Go Testing" feature.
- Week **13**: Implementation of the "Automated Change Management for API Chains" feature.

## 9. Additional Information:

- I've already implemented contract testing using spring boot so it would be good to implement it into Keploy.
- I worked for companies like Ejada,VMware so I know how to work in a team.
- I took 3 training with ITI in these topics: **MEAN stack,Red hat system administration,Flutter**
- I had many experiences in backend, frontend and testing development.
- Here is some of my projects:
  - [RedditX Backend](#): Reddit clone
  - [Alpha-Pet](#): full-stack web-application using [Angularjs](#),[Node.js](#),[Express.js](#) and MySQL which is a mimic of Vezeeta.
  - [OS Scheduler](#): Scheduling the processes by the **operating system** is one of the most important jobs of the **OS**. We will apply some scheduling algorithm in this project, like **SRTN** algorithm, **RoundRobin** algorithm, **HPF** algorithm.
  - [Book Store](#): full-stack web-application using [Angularjs](#), [Node.js](#), [Express.js](#), [MongoDB](#) and used [ngBootstrap](#), [NodeMailer](#), [Json Web Tokens](#), [Font Awesome](#), [Paypal Checkout](#), [Animate.css](#)
  - [Logic Simulator](#) : Simulator like [Logisim](#).
- Keploy is the only project I focused on and interested in.

*Thank You*