PSA: Exam **& Exam**

Now you know your **midterm exam** score, time for the final exam and presentation. *Can you do it with a broken heart?*

Any ideas for the final exam design?
Let me know by today (this is the last chance *literally*).

Will announce **the final exam** organisation by the end of this week.

PSA: Voting **Today**

Presentation Organisation

- **Course-Organised:**
  12m Presentation
  + 8m Demo
- **Self-Organised:**
  Declare the time
  (within 20m)
  by Week 14.

Note: Course-Organised duration used the same structure as the previous one (Week 4). <u>It does not consider the time</u> for CI live-demos (if any).

Presentation Order

- **Reservation:**
  Submit early = present early.
- **Random:**
  Based on a DIY seeded random script (the randomised order will be announced **the next week**).

us.

In progress of getting *2nd semester course(s)* to lecture.

First priority is to lecture **you** in the next semester;
Don't wanna cause "gaps" due to different lecturers (some students could suffer from that).

But can't promise that **I** will get what I want.

Have the meeting on this after today's class.
Will give the update ASAP.

"One for the Road"

#2: MS Test Strategy @ 24/09

Recall: Automated Builds

TL;DR: Rather than building the system locally and manually, building the system automatically via scripts. So we can test them automatically (i.e. the next week class) and make them production-ready with minimal labours.

Automated Tests

TL;DR: Rather than testing the system locally and manually, testing the automatically built system (i.e. the last week class) automatically via scripts. So we can test them automatically and make them production-ready with minimal labours.

Automated Tests

Example: GitHub Actions

GitHub Docs | Version: Free, Pro, & Team | Search GitHub Docs

GitHub Actions / Use cases and examples

Use cases and examples

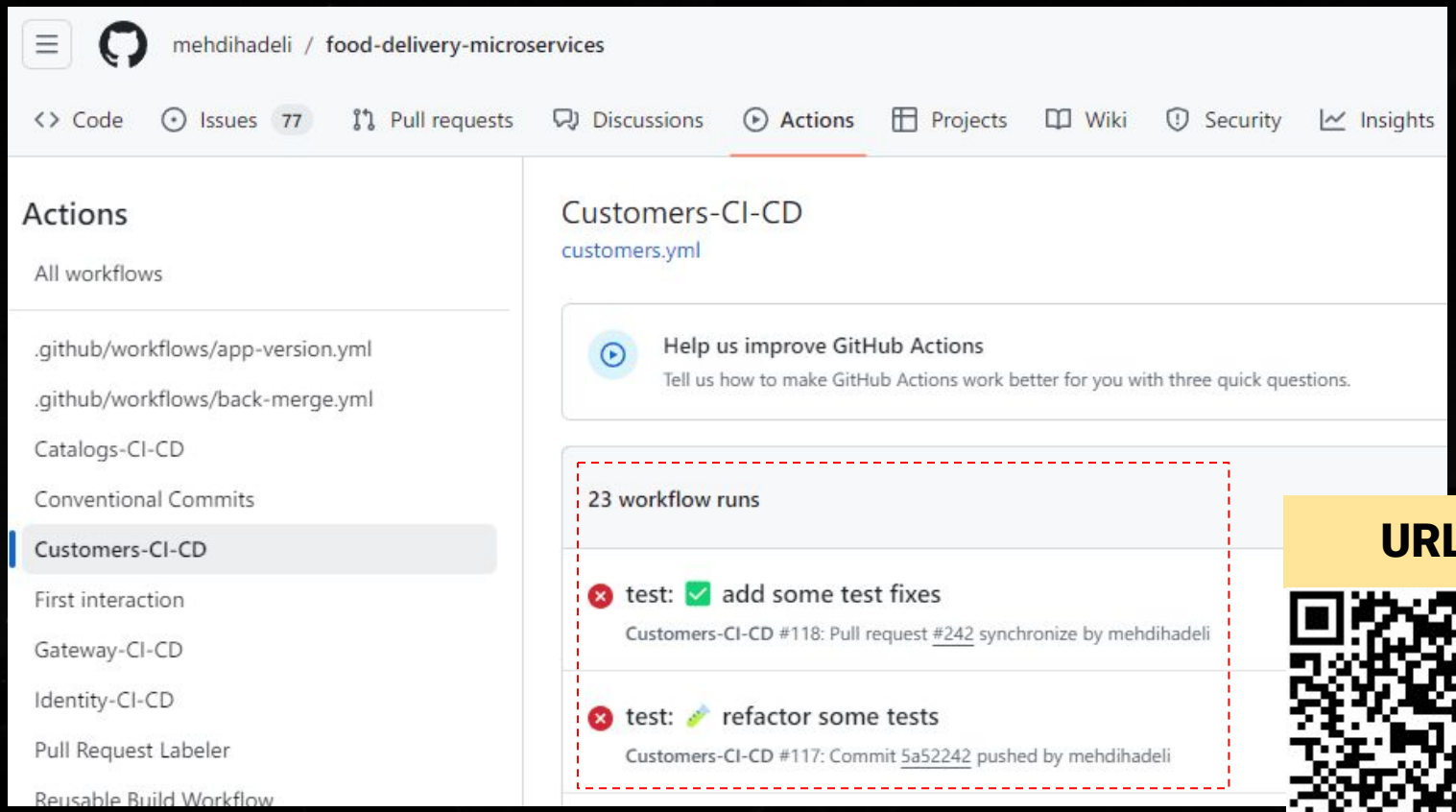
Example workflows that demonstrate the features of GitHub Actions.

- Creating an example workflow
- Building and testing
  - Building and testing Go
  - Building and testing Java with Ant
  - Building and testing Java with Gradle
  - Building and testing Java with Maven
  - Building and testing .NET
  - Building and testing Node.js
  - Building and testing PowerShell
  - Building and testing Python
  - Building and testing Ruby
  - Building and testing Swift
  - Building and testing Xamarin applications

URL:

Automated Tests

Example: Automated Tests in Action



URL:

Automated Tests

Example: Automated Unit Test Results

```
Passed!  - Failed:     0, Passed:    56, Skipped:     0, Total:    56, Duration: 866 ms - FoodDelivery.Services.Customers.UnitTests.dll (net8.0)

Calculating coverage result...
  Generating report '/home/runner/work/food-delivery-microservices/food-delivery-microservices/tests/Services/Customers/FoodDelivery.Services.Customers.UnitTests/coverage.cobertura.xml'
  Generating report '/home/runner/work/food-delivery-microservices/food-delivery-microservices/tests/Services/Customers/FoodDelivery.Services.Customers.UnitTests/coverage.info'

+------------------------------------+--------+--------+--------+
| Module                             | Line   | Branch | Method |
+------------------------------------+--------+--------+--------+
| FoodDelivery.Services.Customers.Api | 0%     | 0%     | 0%     |
+------------------------------------+--------+--------+--------+
| FoodDelivery.Services.Customers    | 24.21% | 40.8%  | 22.15% |
+------------------------------------+--------+--------+--------+

+---------+--------+--------+--------+
|         | Line   | Branch | Method |
+---------+--------+--------+--------+
| Total   | 23.81% | 38.58% | 22.08% |
+---------+--------+--------+--------+
| Average | 12.1%  | 20.39% | 11.07% |
+---------+--------+--------+--------+
```

URL (Login Required):

Automated Tests

Example: Automated Integration Test Results

✗ tests/Services/Customers/FoodDelivery.Services.Customers.IntegrationTests/TestResults/test-results.trx

21 tests were completed in **1310s** with **17** passed, **4** failed and **0** skipped.

Test suite	Passed	Failed	Skipped	Time
FoodDelivery.Services.Customers.IntegrationTests.Customers.Features.CreatingCustomer.v1.CreateCustomerTests	7 ✓			88s
FoodDelivery.Services.Customers.IntegrationTests.Customers.Features.GettingCustomerByCustomerId.v1.GetCustomerByCustomerIdTests	2 ✓			25ms
FoodDelivery.Services.Customers.IntegrationTests.Customers.Features.GettingCustomerById.v1.GetCustomerByIdTests	2 ✓			51ms
FoodDelivery.Services.Customers.IntegrationTests.Customers.Features.GettingCustomers.v1.GetCustomersTests	2 ✓			213ms
FoodDelivery.Services.Customers.IntegrationTests.RestockSubscriptions.Features.CreatingRestockSubscription.v1.CreateRestockSubscriptionTests	1 ✓			477ms
FoodDelivery.Services.Customers.IntegrationTests.Users.Features.RegisteringUser.v1.Events.External.UserRegisteredTests	3 ✓	4 ✗		1204s

URL (Login Required):

Automated Tests

Example: Automated End to End Test Results

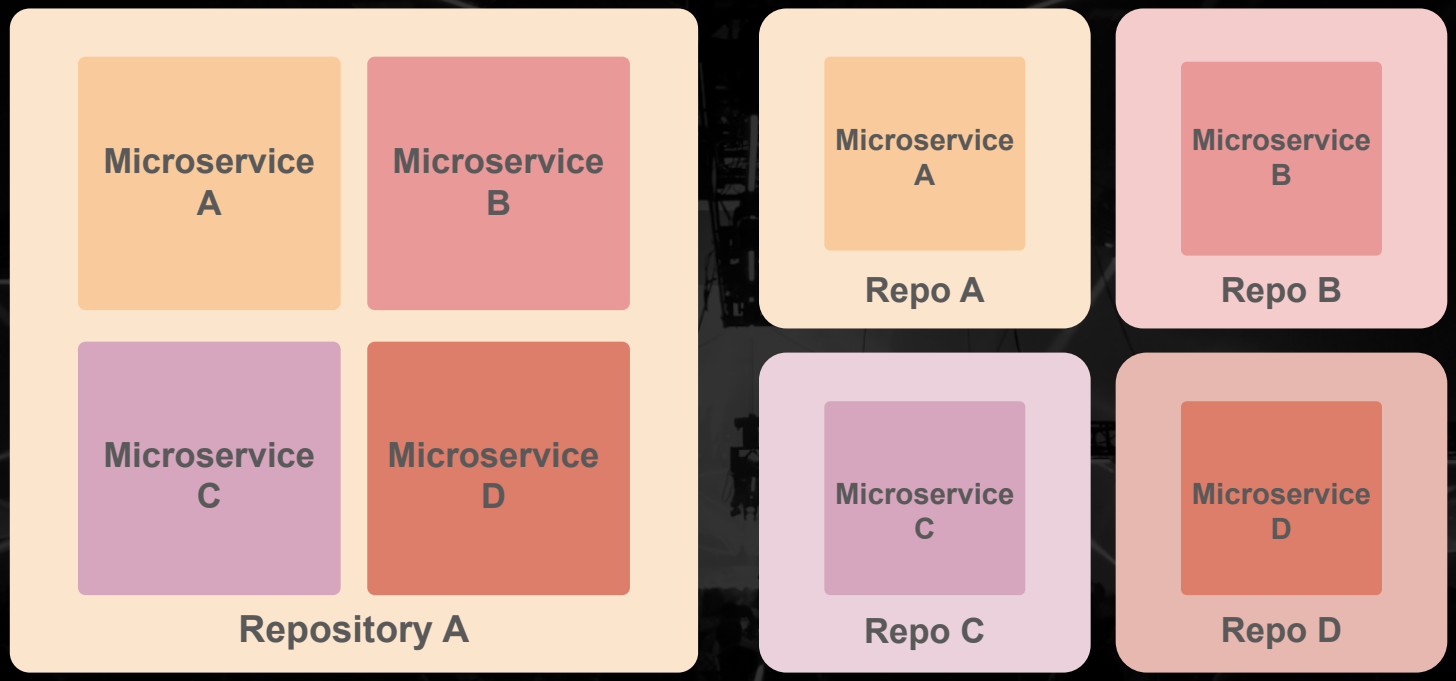
FoodDelivery.Services.Customers.EndToEndTests.Customers.Features.GettingCustomerById.v1.GetCustomerByIdTests

```
✔ can returns ok status code using valid id and auth credentials
✘ can returns valid response using valid id and auth credentials
      Microsoft.VisualStudio.TestTools.UnitTesting.AssertFailedException : Expected response to satisfy one or more model assertions, but it wasn't:

          - expectation has property x.Customer.FirstName that the other object does not have.
          - expectation has property x.Customer.LastName that the other object does not have.
          - expectation has property x.Customer.FullName that the other object does not have.
          - expectation has property x.Customer.Created that the other object does not have.


      The HTTP response was:

      HTTP/1.1 200 OK
      api-supported-versions: 1.0
      Content-Type: application/json; charset=utf-8
      Content-Length: 420

      {
        "customer": {
          "id": "d170beb7-5b16-3911-5c33-2e1273d50e33",
          "customerId": 1,
          "identityId": "2585bd39-a80e-0e1f-7c29-67807483385b",
          "email": "Johathan_Satterfield66@yahoo.com",
          "name": "Braden Adams",
          "country": "Colombia",
          "city": "Lake Garrick",
          "detailAddress": "45760 Maggio Route, Lake Eva, Yemen",
          "nationality": "Global",
          "birthDate": "2024-09-15T11:46:24.425\u002B00:00",
          "phoneNumber": "(\u002B53)3603337355",
          "createdAt": "0001-01-01T00:00:00"
        }
      }
      The originating HTTP request was:

      GET http://localhost/api/v1/customers/d170beb7-5b16-3911-5c33-2e1273d50e33 HTTP 1.1
      Accept: application/json
      Authorization: FakeBearer ***
      Content-Length: 0
```

URL (Login Required):

Repository Patterns

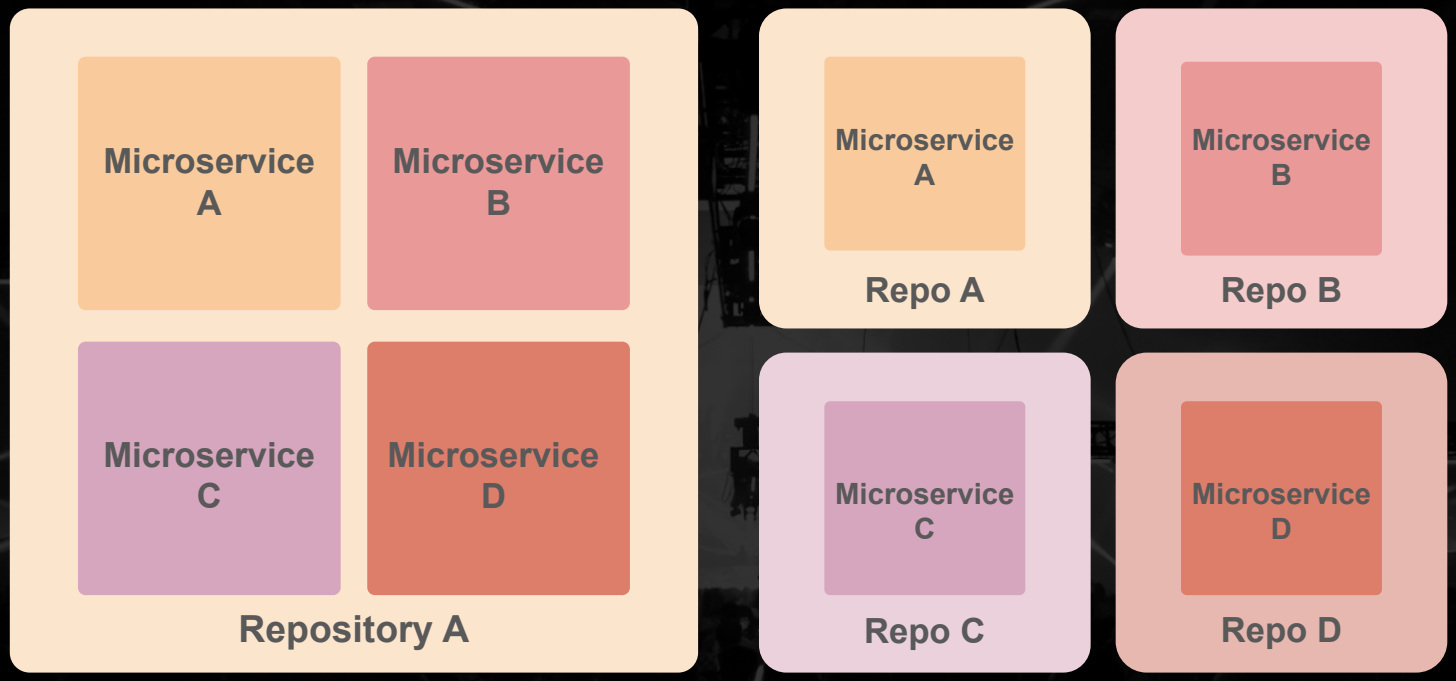
For the separation of concerns, each microservice should be able to test ***independently*** regardless of repository patterns.



A) Monorepo **B) Multi-repo**

Repository Patterns

But each microservice may depend on each other IRL (e.g. B requires a response from C & a data from A).



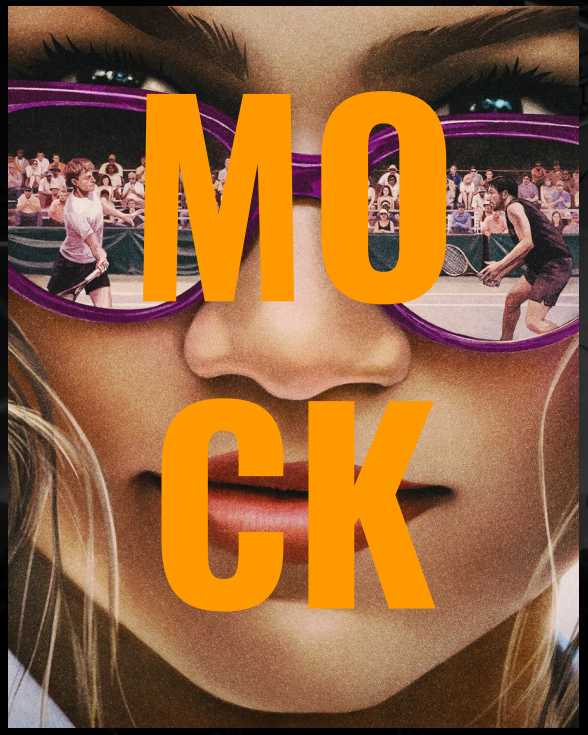
A) Monorepo **B) Multi-repo**

Testing Techniques for MS

Two Techniques to Mitigate This.
Both can be used in single project IRL:

Mock

TD;LR: Dumber version of a real request/data but smarter than Stub.



Characteristics:

- Output is **not predetermined** (e.g. random, calculated result, etc.)
- Use to verify expected **behaviour/interactions** /wo relying on other microservices/data.
- **Require technical knowledge** to create.

Mock

"Do you know what Tennis is? It's a relationship."

Real-life example: Tennis Ball Machine.

Observation:

- Dumber version of a coach/a sparring partner.
- Output is **not predetermined:** Programmable to shoot in different ways & to random positions.
- Use to verify expected **behaviour/interactions**: Rehearse how a tennis player react when the ball is approaching.
- **Require technical knowledge** to create: Only experienced tennis player can design good ones.

Mock

Example: Tennis Ball Machine.

Assuming “launchBall() : String” is a function for a Tennis Ball Machine.

Output of launchBall() from 4 Trial:

Trial 1:

“
Slice: **No**,
Shot: **Backhand**,
Position: **100,120**,
Speed(mph): **85**
”

Trial 2:

“
Slice: **Yes**,
Shot: **Forehand**,
Position: **0,50**,
Speed(mph): **65**
”

Trial 3:

“
Slice: **Yes**,
Shot: **Lob**,
Position: **-20,40**,
Speed(mph): **70**
”

Trial 4:

“
Slice: **Yes**,
Shot: **Drop**,
Position: **-100,-100**,
Speed(mph): **89**
”

Mock

Example: Mock in Action - Unit Test Case.

Assuming “launchBall() : String” is a mock function replacing in the first serve in a multiplayer tennis game.

Unit Test Case 1: Ensure that the ball machine will not launch the ball out of bound (X position-only - between -150 to 150).

Target:
x = launchBall().XPosition //Assuming this will fetch XPosition string.

Input: N/A

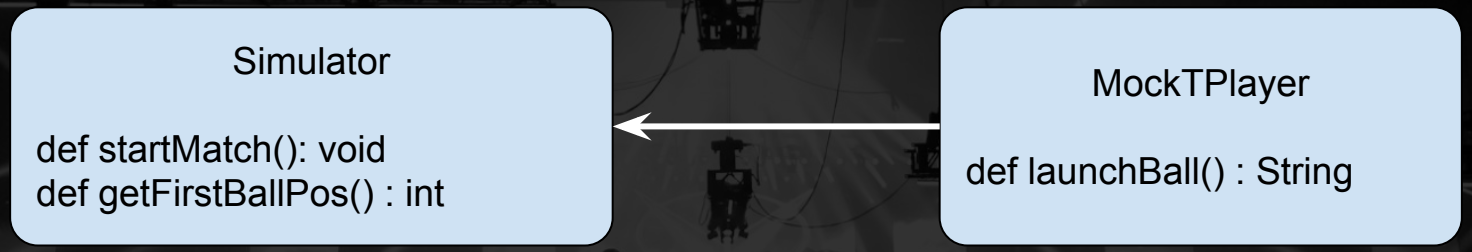
Output:
Pass: int(x) <= 150 && int(x) >= -150
Fail: int(x) > 150 || int(x) < -150

Mock

Example: Mock in Action - Integration Test Case.

Assuming “launchBall() : String” is a mock function replacing in the first serve in a multiplayer tennis game.

Integration Test Case 1: Ensure that the ball machine will not launch the ball out of bound (X position-only - between -150 to 150).



Target: x = getFirstBall@Simulator + int(launchBall().XPosition@MockTPlayer)
Input: N/A (Sending a request to MockTPlayer via startMatch()@Simulator)
Output: Pass: x <= 150 && int(x) >= -150, Fail: int(x) > 150 || int(x) < -150

Mock

Real Example: Mock in Action - AutoBogus to generate a fake email (i.e. f.Internet.Email())

food-delivery-microservices / tests / Services / Customers / FoodDelivery.Services.Customers.TestShared / Fakes / Customers / Commands / FakeCreateCustomer.cs

mehdihadeli refactor: upgrade to .net 8 and refactoring (#227) ✕ 4252a41 · 2 months

Code Blame 19 lines (17 loc) · 1.03 KB Raw

```
using AutoBogus;
using FoodDelivery.Services.Customers.Customers.Features.CreatingCustomer.v1;

namespace FoodDelivery.Services.Customers.TestShared.Fakes.Customers.Commands;

// Note: AutoBogus doesn't generate values for readonly properties (propertyInfo.Ca
// Note that, should a rule set be used to generate a type, then only members not d
// https://github.com/nickdodd79/AutoBogus#autofakert
// `Faker` has a problem with non-default constructor but `AutoFaker` works also wi
// because AutoFaker generate data also for private set and init members (not read
internal sealed class FakeCreateCustomer : AutoFaker<CreateCustomer>
{
    public FakeCreateCustomer(string? email = null)
    {
        long id = 1;
        RuleFor(x => x.Email, f => email ?? f.Internet.Email());
        RuleFor(x => x.Id, f => id++);
    }
}
```

URL (Login Required):

Stub

TD;LR: Dumber version of **Mock**.

Characteristics:

- Output is **determined** (e.g. hard-coded in the code, stored in the db, etc.)
- Monkey see = monkey do. Require **less knowledge** to create compared **to Mock**.
- Use to **replace dependency** when testing.

Stub

Real-life example: Music Player during Dance Practice.

Characteristics:

- Output is **determined:** Play a selected song to practice. Always.
- Require **less knowledge** to create compared **to Mock**: Know the song name = play that song.
- Use to **replace dependency** when testing: Make artists concentrates on dancing, less/no dependency in singing during practice.

Stub

Real-life example: Music Player during Dance Practice.

Assuming “playCreamSoda() : void” is a function to play music & “playCreamSoda().info : String” to get the info of (hard-coded) Song ID.

Output of playCreamSoda().info from 4 Trial:

Trial 1:

“Song:
Cream Soda
Artist: EXO,
Duration:
3:05
Song ID:
42h7yc9Rda1IO
MYLACVgld”

Trial 2:

“Song:
Cream Soda
Artist: EXO,
Duration:
3:05
Song ID:
42h7yc9Rda1IO
MYLACVgld”

Trial 3:

“Song:
Cream Soda
Artist: EXO,
Duration:
3:05
Song ID:
42h7yc9Rda1IO
MYLACVgld”

Trial 4:

“Song:
Cream Soda
Artist: EXO,
Duration:
3:05
Song ID:
42h7yc9Rda1IO
MYLACVgld”

Stub

Example: Music Player.

Assuming “playCreamSoda() : void” is a function to play music & “playCreamSoda().info : String” to get the info of (hard-coded) Song ID.

Unit Test Case 2: Ensure that the function play the right song.

Target:
x = playCreamSoda().info.SongID //Assuming this will fetch Song ID string.

Input: N/A

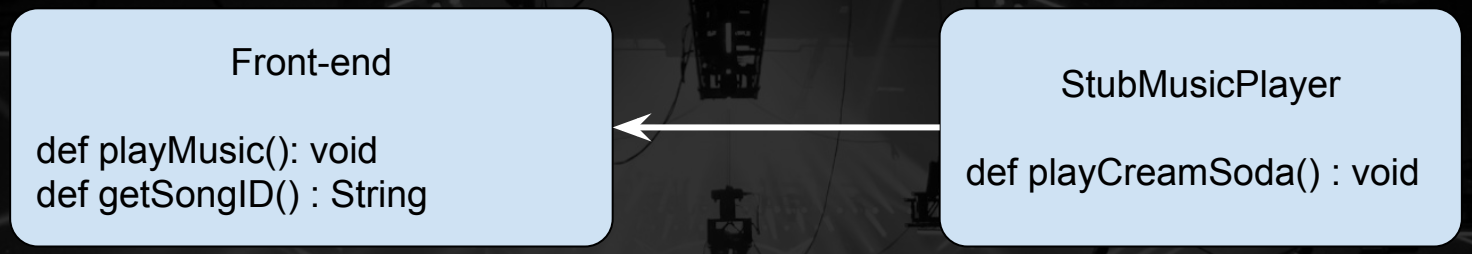
Output:
Pass: x == 42h7yc9Rda1IOMYLACVgld
Fail: x != 42h7yc9Rda1IOMYLACVgld
//This is a Spotify Song ID (no easter egg here).

Stub

Example: Music Player.

Assuming “playCreamSoda() : void” is a function to play music & “playCreamSoda().info : String” to get the info of (hard-coded) Song ID.

Integration Test Case 2: Ensure that the function play the right song.



Target: x = getSongID()@Front-End + playCreamSoda().info.SongID@StubMusicPlayer
Input: N/A (Sending a request to StubMusicPlayer via playMusic()@Front-End)
Output:
Pass: x == 42h7yc9Rda1IOMYLACVgld
Fail: x != 42h7yc9Rda1IOMYLACVgld

Stub

Real Example: Stub in Action - Using a hard-coded email (“i.e. test@example.com”) to test a validator.

food-delivery-microservices / tests / Services / Customers / FoodDelivery.Services.Customers.UnitTests / Customers / Features / CreatingCustomer / v1 / CreateCustomerValidatorTests.cs

mehdihadeli refactor: enhance building blocks codebase (#234) f5f13fa · last month History

Code Blame 60 lines (51 loc) · 1.91 KB Raw

```
using FluentAssertions;
using FluentValidation.TestHelper;
using FoodDelivery.Services.Customers.Customers.Features.CreatingCustomer.v1;
using FoodDelivery.Services.Customers.UnitTests.Common;
using Tests.Shared.XunitCategories;

namespace FoodDelivery.Services.Customers.UnitTests.Customers.Features.CreatingCustomer.v1;

public class CreateCustomerValidatorTests : CustomerServiceUnitTestBase
{
    [Fact]
    [CategoryTrait(TestCategory.Unit)]
    public void must_success_with_valid_inputs()
    {
        // Arrange
        var command = new CreateCustomer("test@example.com");
        var validator = new CreateCustomerValidator();

        var result = validator.TestValidate(command);
        result.IsValid.Should().BeTrue();
    }
```

URL:

Group **Exercise -** Week 12

Main Objective:

1. Create a test strategy for your system that help in your problem statement requirement(s).
2. Create test case(s) /w Mock based on a test strategy in 1.
3. Create test case(s) /w Stub based on a test strategy in 1.

Vote:

1. Time for Presentation: Self-Organised/Course-Organised.
2. Order for Presentation: Reservation/Random.

Submit to:
suwichak.fu(at)kmitl.ac.th

Subject:
[6622][Team Name] Group Exercise Submission