# Turning Singleton Usage into Testable Code

SEE HOW ONE PROTOCOL-BASED CHANGE MAKES YOUR SINGLETON INJECTABLE AND THE CODE FULLY TESTABLE.

**Maciej Gomółka**

# The problem

- Service uses *URLSession.shared* directly.
- Tight coupling makes unit testing impossible without real network calls.

```
struct PostsAPISerivce {
    func fetchPosts() async throws -> [Post] {
        let url = URL(string: ".../posts")!
        let (data, _) = try await URLSession.shared.data(from: url)
        return try JSONDecoder().decode([Post].self, from: data)
    }
}
```

On the next slides, you'll get step-by-step guide of making this service testable.

These same steps can be applied to all other singletons found in your code.

**Maciej Gomółka**

# Step 1 – Inspect the used API

- CMD-click on *data(from: url)* to view the documentation ⤵

```
/// Convenience method to load data using a URL, creates
/// and resumes a URLSessionDataTask internally.
///
/// - Parameter url: The URL for which to load data.
/// - Parameter delegate: Task-specific delegate.
/// - Returns: Data and response.
public func data(
    from url: URL,
    delegate: (any URLSessionTaskDelegate)? = nil
) async throws -> (Data, URLResponse)
```

**Maciej Gomółka**

# Step 2 – Define a protocol

- Create *URLSessionProtocol*
- Copy the function signature from the documentation into your protocol definition.
- Remove default arguments (not allowed in protocol).

```
protocol URLSessionProtocol {
    func data(
        from url: URL,
        delegate: (any URLSessionTaskDelegate)?
    ) async throws -> (Data, URLResponse)
}
```

**Maciej Gomółka**

# Step 3 – Conform *URLSession* to *URLSessionProtocol*

- Add the following extension to make *URLSession* conforms to your protocol ⤵

```
extension URLSession: URLSessionProtocol {}
```

**Maciej Gomółka**

# Step 4 – Inject Dependecy

- Refactor the service and inject *URLSessionProtocol* into it.

```
struct PostsAPISerivce {
    private let urlSession: URLSessionProtocol

    init(urlSession: URLSessionProtocol = URLSession.shared) {
        self.urlSession = urlSession
    }

    func fetchPosts() async throws -> [Post] {
        let url = URL(string: ".../posts")!
        let (data, _) = try await urlSession.data(from: url, delegate: nil)
        return try JSONDecoder().decode([Post].self, from: data)
    }
}
```

Now a Spy or Mock conforming to *URLSessionProtocol* can be created and injected into *PostsAPIService* to simulate API responses.

**Maciej Gomółka**

# Summary

Benefits

- No real API calls in tests ✅
- Spies and Mocks can be injected in tests to control API responses (successes & errors) ✅
- Reusable for all other components requiring *URLSession* ✅

**Remember** - These same steps can be applied to all other singletons found in your code.

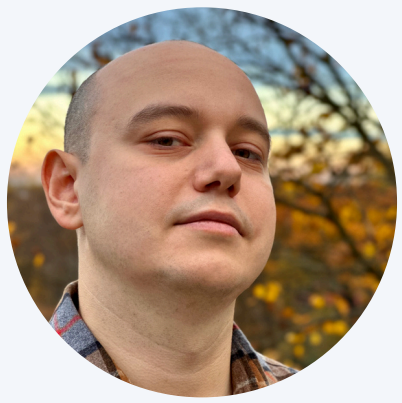
**Maciej Gomółka**

# Let’s Connect!

🔷 **Follow** for more Swift & testing tips 🔷

🔷 **Comment** your thoughts or questions 🔷

🔷 **Reshare** to help others level up 🔷



**Maciej Gomółka**