International Journal of Core Engineering & Management
Volume-7, Issue-07, 2023 ISSN No: 2348-9510

INTEGRATING MODERN CLIENT APPLICATIONS WITH LEGACY BACKENDS USING AUTOMATION FRAMEWORK

Tanmaya Gaur
Bachelor of Engineering (Electronics and Telecommunication),
Birla Institute of Applied Sciences
tanmay.gaur@gmail.com

Abstract

Most modern organizations end up having a mix of legacy systems living side by side with modern applications. There is always the desire to modernize the entire ecosystem to be based on latest and greatest technology choices and patterns, achieving this 'NorthStar' however, ends up being a moving target. In situations, where these legacy systems can be abstracted by an API tier, that provides an appropriate middle ground by layering usable and more securelayers and authorization practices on top of the abstracted legacy backend. There however are always legacy backends that are only available tightly coupled to a legacy UI and not built to be available via an API standalone. This paper explores an option to create an API tier by using the legacy UI using solutions traditionally built for test automation. This solution may not work or be particularly applicable to be all scenarios, considerations which the paper will try to highlight as well.

Index Terms—Legacy Backends, Security, Usability, Modernization, Architecture

I. INTRODUCTION

There are multiple reasons, last mile challenges like usability, security amongst them which force the companies to try and move away from the legacy stack into more modern and secure solutions. While this need to modernize legacy systems is always a critical, it is not always achievable. An ideal approach is to upgrade or enhance the frontend, middleware or backend Tier as needed. This however is often not always possible, in such cases when let's say a backend cannot be migrated right away. In this case, the backend system is often placed behind a bridging on interface layer which can be appropriately secured behind the latest authentication, authorization and networking enhancements.

International Journal of Core Engineering & Management
Volume-7, Issue-07, 2023 ISSN No: 2348-9510

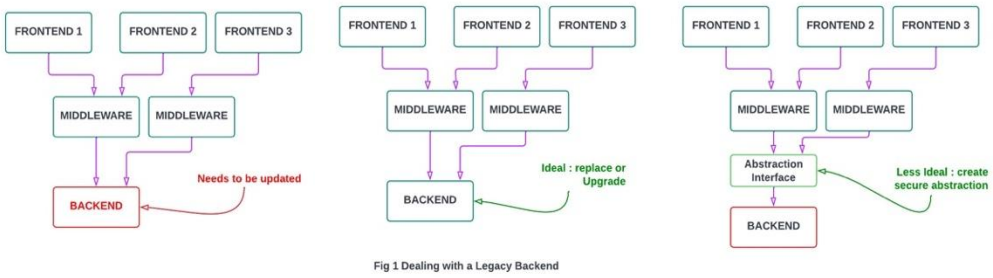

Fig 1 Dealing with a Legacy Backend

This security is not always possible for legacy systems that are either not available via traditional API(s) or are tightly coupled to a UI and cannot onboard new clients. While it sounds reasonable to perform an upgrade or complete re-write, there are often constraints which are at play. In such scenarios, businesses are often faced with a choice of having to live with the possible security and fraud risk of exposing such solutions. Dismal usability of these older stacks is often another compromise with such systems.

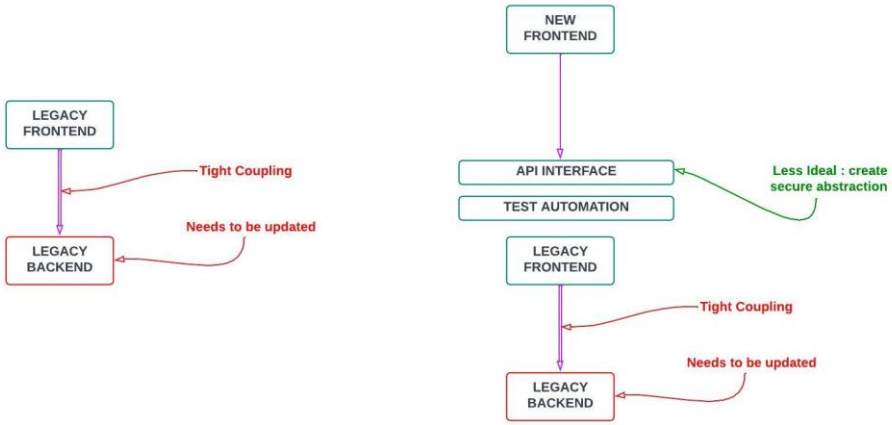

Fig 2 Dealing with a Legacy Backend with tight frontend integration

Leaving these legacy systems accessible over the network, especially directly from a user experience layer can pose significant risks, especially given the evolving security landscape. This paper will discuss some options to use tools traditionally built for test automation to be

able to resolve this challenge. For the purpose of this document, we will use a web application automation example, but similar solutions can be applied to native OS applications as well. The document will discuss challenges and considerations for this approach.

II. LEGACY SYSTEMS AND THE NEED TO UPDATE

A. The need to Update Legacy Solutions

There are often multiple reasons driving the need to upgrade or update legacy backends. Some common reasons and considerations are listed below

- Security : Legacy Systems are often not secure as per the latest standards and at times lack security updates and patches, leaving them vulnerable.
- Regulatory Compliance : Legacy systems may not always comply with the latest regulatory standards, especially in environments that desire heightened compliance. As an example, there may be industry regulations such as data privacy and security standards, often with an overhanging legal and financial compliance forcing the need to update.
- Integration Difficulty : Legacy software may have compatibility issues with modern systems which can lead to integration gaps. The integration can often be buggy even if achieved.
- Efficiency : modern systems can have better processing capabilities significantly impacting throughput or other measurable and important KPIs important to the organization.
- Productivity and Velocity: Updating legacy software can boost development productivity. Research in some studies has cited 40% or more productivity increase in productivity.
- Scalability : Legacy software may not meet the scaling demands of a growing organization or user base which may impede growth.
- Cost : There can be two aspects to cost, one of the overall maintenance cost of legacy systems. There is also the lack of development expertise on aging technologies which become niche, which raises costs in identifying talent for upkeep and enhancements. This also at times impedes team performance and velocity, which raises costs as well.
- Inexistent documentation : Legacy software may lack documentation which necessitates update but is mostly the biggest deterrent.
- Overall Usability : If this legacy system is tightly coupled to a user experience, this is another situation where the backend is now impeding user experience updates and results in usability gaps.
- New features and Improvements : Legacy software may not incorporate some latest features, enhancements and performance improvements that can benefits an organization.

B. Legacy Update Options

There are a few options available to enterprises needing to update their legacy systems.

- Updating User Interface : This is an option if the update is due to usability needs but has no broader security or other Non-functional concerns. This is often an enhancement or re-skin to update the look and feel of the application or software while not updating the core.

- Refactoring and code improvements : This implies refactor and re-organization of the codebase without changing the overall stack or versions. This may provide readability, maintainability and performance benefits. This option may at times also help improve scalability of the application. This is often a lower effort option compared to others but may not always remediate all the driving factors for modernization.

- Version upgrades and Patching : This option requires upgrading to the latest versions of software available which allows the benefits of new capabilities and features to become available. This option too may come with a scalability and performance boost as well as making sure the software gets up to date with industry standards and security patches. This however may not always be an option given support for the legacy software available. Also, there are times where upgrades or patches are not backward compatible and may need a rewrite to be eligible for upgardes.

- Modernizing infrastructure via containerization or cloud adoption : While this option is technically not a software upgrade, there are benefits that can be achieved by updating the underlying hardware. The modern containerized or cloud native deployment options often come with improved scalability and flexibility. This may also simplify the stack and reduce overall costs.

- Migration to New Software : This is the costliest of all options but becomes unavoidable at times. Transitioning to fresh software system can often offer an opportunity for drastic simplification, modernization and innovation. This option allows us to rebuild the solution with the most suitable or modern tooling which also results in significant productivity gains. There may also be advanced features or capabilities available with the new system which cannot be otherwise realized.

C. Legacy Update Gaps for Monolithic applications

It is not always possible for organizations to update their software stack. There are often reasons like costs, prioritization, lack of required expertise etc. which are holding up timely updates. There are however times when the monolithic and tightly coupled nature of the legacy stack hinder updates.

For legacy applications that are monolithic in nature, tightly coupled, they become difficult to modify or integrate with newer technologies. The monolithic nature also prevents companies from updating the applications in small decomposable bits thereby increasing the upfront

investment and overall risk manifold. Tight coupling also defeats modernization strategies like encapsulation via APIs, re-platforming, refactoring as discussed in earlier section. These also make replacement risky as it requires a complete re-write of the entire application instead of a phased update.

III. SYNTHETIC TEST AUTOMATION OPTIONS

A. Synthetic Test Automation

Synthetic tests, which may be executed against production environments, simulate requests that mimic how real users would interact with a website or application. Modern synthetic automation tools often allow easy recordings and replay of complex UI experiences. Modern synthetic automation frameworks that provide test automation capabilities across different device stypes for native, web, or hybrid mobile apps and across web, desktop, and legacy applications.

Synthetic test automation does have downsides including difficult initial setup and requirement of specialized knowledge or skill sets, the brittleness of the tests, especially for rapidly changing applications which may not be the case with legacy applications as well as difficult debugging and maintenance over time.

There are enterprise solutions which solve this space like UI path and Tosca, but they often come at a cost. Open-Source solutions like selenium or options like puppeteer and playwright require significant investment to operationalize and scale.

For our example, we will utilize a web app and puppeteer, which is open source to build an abstraction solution.

B. What is Puppeteer

Puppeteer is an open-source tool developed by Google that automates and streamlines front-end development and testing. Essentially, Puppeteer is a Node library that controls headless Chrome via the DevTools Protocol. It provides high-level APIs for automated testing, developing and debugging website features, inspecting page elements, and profiling performance.

To clarify, a headless browser operates like a regular browser but without a visible interface. It runs in the background, performing all necessary functions without displaying anything on the screen, hence the term "headless."

In our solution, we can trigger a Puppeteer script to interact with the legacy web application, retrieve specific data from the web page, or, in our case, extract a token from the response and return it as an API response.

C. Synthetic solution on a Legacy App

Let's consider a scenario where tightly coupled monolithic app which is fronted by an Apache web-gate. To access the App requires a user to access a protected resource, a request which is intercepted by the Webgate which enforces login challenge requiring username/password. The

International Journal of Core Engineering & Management
Volume-7, Issue-07, 2023 ISSN No: 2348-9510

supplied credentials are validated prior to Webgate allowing requests to be served including to data end-points required to populate the app. Webgate generates opaque local tokens which allow this access for the duration of the user session and ensure data integrity. The below illustration (figure 3) depicts this pattern.

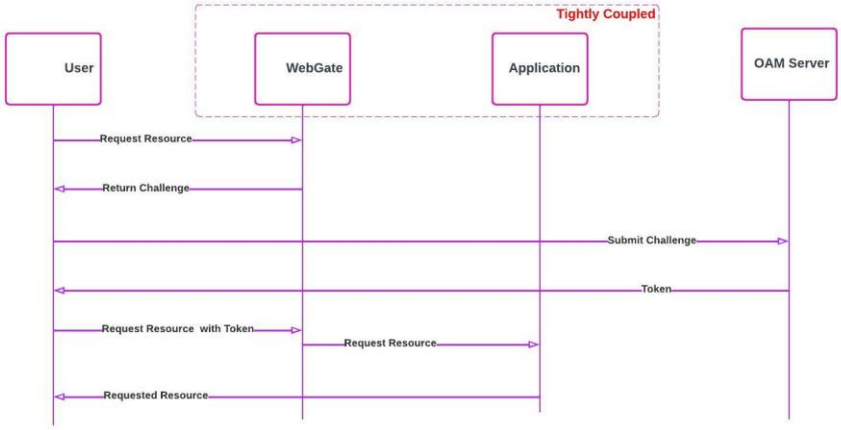

Fig 3 A monolith tightly coupled application scenario

Assuming the backend cannot be updated immediately, A modern web application layered on top of this stack with the aim of replacing legacy UI stack needs to retrieve the session token to be able to request protected resources. The below depiction (figure 4) lays out the updated integration pattern with puppeteer in place.

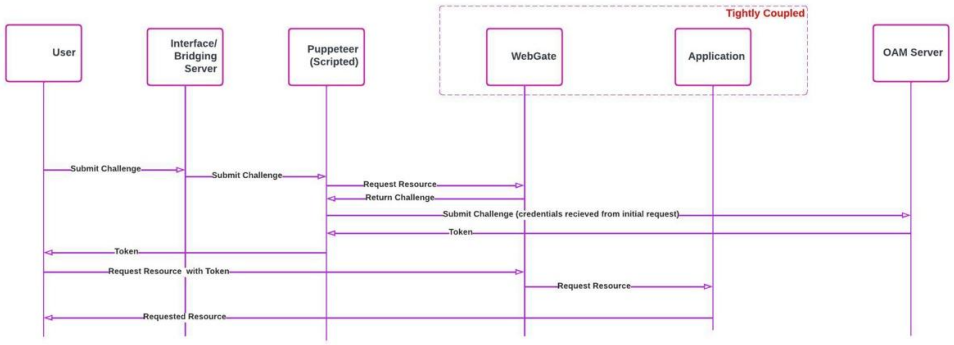

Fig 4 Simplified API Interaction with Puppeteer in the mix

International Journal of Core Engineering & Management
Volume-7, Issue-07, 2023 ISSN No: 2348-9510

D. Challenges or limitations with Puppeteer based solution

While The Puppeteer project has been widely adopted by developers as a simpler way to introduce automation into their development workflow. Most of the synthetic testing limitations discussed in section 3.1 apply to puppeteer scripts. There are performance, scalability challenges of implementing this solution at scale, especially when complex UI experiences are involved. Solutions such as puppeteer have a steep learning curve and are often limited when it comes to dynamic content or media and challenges like captcha.

E. Sample Script

Our sample stack runs on puppeteer running on an express server (refer figure 5).
The express server or similar web framework provides the external API which the modern consumer can invoke. Essentially, express allows us to turn puppeteer into a RESTful API. When this restful API receives a request, it in turn triggers the puppeteer routine. In our example, Puppeteer will first need to navigate the legacy UI, authenticate if necessary, and perform actions such as clicking buttons, filling out forms, and scraping data. Here's an example of how to use Puppeteer to automate a login process by submitting a user-name and password and then retrieving a data element from the subsequent page.

```
const puppeteer = require('puppeteer');
(async () => {
const browser = await puppeteer.launch();
const page = await browser.newPage();
await page.goto('[URL]');
// Perform login
await page.type('#username', 'your-username');
await page.type('#password', 'your-password');
await page.click('#login-button');
// Wait for navigation after login
await page.waitForNavigation();
// Scrape data
const data = await page.evaluate(() => {
return document.querySelector('#data-element').innerText;
});
console.log(data);
await browser.close();
})();
```

Fig 5 – Sample script for instantiating puppeteer and logging in

Below code (refer figure 6) establishes way to wrap this routine with an express server on port 3000 with the app.get method handling the request, opening a headless browser page and executing the script.

International Journal of Core Engineering & Management
Volume-7, Issue-07, 2023 ISSN No: 2348-9510

```
const express = require('express');
const puppeteer = require('puppeteer');
const app = express();
app.get('/api/data', async (req, res) => {
const browser = await puppeteer.launch();
const page = await browser.newPage();
await page.goto('[URL]');
// Perform login and scrape data (similar to previous example)
// ...
const data = await page.evaluate(() => {
return document.querySelector('#data-element').innerText;
});
await browser.close();
res.json({ data });
});
app.listen(3000, () => {
console.log('Server is running on port 3000');
});
```

Fig 6 – Sample script for exposing the a data via a restful interface

IV. OTHER OPTIONS FOR SOLVING SECURITY

Assuming that the risk with legacy applications has more to do with security and fraud and usability is not a concern, below options may work in putting a secure parameter around access.

A. Securing the Browser or Native run-time

While this option may not work over open internet, for legacy apps within an enterprise boundary, there may be options to force the apps to launch a certain way or within specific browser run-times with significant restrictions. The ability to manage the device and policies on the browser provide significant risk mitigation. Certain enterprise versions of the browsers come with granular controls which can be established.

B. Browser Isolation

This is an evolving technique which can provide significant security and risk abstraction. Browser isolation is a cybersecurity technique that separates a user's browsing activity from their local device and network. It's also known as remote browsing or web isolation. This approach can also face scaling issues with heavily used web-sites and may impact end user performance.

As such, and this option may not work over open internet, for legacy apps within an enterprise boundary, there may be options to force the apps to launch a certain way or within specific browser run-times with significant restrictions. The ability to manage the device and policies on the browser provide significant risk mitigation.

V. LIMITATIONS / CHALLENGES

The approach laid out in the manuscript does not work for all scenarios or will need significant customization in order to be successful.

A. Slowness

This method depends on replicating user actions and replaying them. Capturing the nuances of human user experience, such as perceived performance and page interactions, is challenging to code and systematically address. Therefore, in our web application scenario, it is advisable for this solution to wait until all JavaScript has finished executing before retrieving and returning the necessary data. However, this can lead to considerable delays in the experiences built on top of this solution.

B. Scaling

The headless solution will need to operate server-side and could require substantial compute and other resources. When dealing with complex, long-running sessions for multiple users simultaneously, scaling may become a significant concern.

C. Costs

Running the interface or bridging server, particularly at scale, can incur substantial costs. It is essential to understand and account for these expenses when considering the solution.

D. Complexity

Some application scenarios may be too intricate or verbose to automate using scripts. This complexity might not become apparent until teams encounter specific use-cases. It is recommended to conduct a pilot implementation with the more complex scenarios to validate the solution before making significant investments.

E. Learning Curve

The scripting and automation tools employed can often present a steep learning curve.

F. Support for Complex Data types

Extracting data from nested elements, such as tables, may not be well-supported. Additionally, handling composite data types like images can introduce further complexity and costs.

VI. CONCLUSION

While it is desirable to keep legacy systems up to date, various factors often prevent enterprises from doing so. The primary reasons usually involve high costs, alignment and priority issues, or timing constraints associated with migrating a legacy monolith to modern solutions. However, emerging gaps in usability, security, and other concerns place these organizations in a challenging position. Although it is advisable in the long term to refactor and rearchitect, modularizing and isolating concerns, teams often resort to short-term mitigations while the

longer-term strategy is being developed.

This manuscript explores an alternative approach using Browser Automation, such as Puppeteer, for web applications. This method offers a powerful and flexible interim solution for modernizing legacy systems by simulating web browser interactions and transforming legacy UI data into RESTful APIs. While promising, this approach comes with considerations, including hidden costs that may not be apparent upfront. When executed correctly, the benefits of improved data accessibility and integration capabilities make it a highly viable solut

REFERENCES

1. 8 Reasons to Update Software https://www.bairesdev.com/blog/update-legacy-software/
2. Update Legacy Software : Options and Best Practices https://binmile.com/blog/modernize-legacy-system/#:~:text=positive%20user%20experience.-,Best%20Practices%20for%20Updating%20Legacy%20Software,controlled%20environment%20to%20minimize%20risks
3. Modernize Your Legacy Systems for Future Growth https://imaginovation.net/services/legacy-system-modernization/
4. Puppeteer Tutorial: Complete Guide to Puppeteer Testing https://www.lambdatest.com/puppeteer
5. What is browser isolation? https://www.cloudflare.com/learning/access-management/what-is-browser-isolation/#:~:text=Browser%20isolation%20(also%20known%20as,browser%20running%20on%20local%20devices
6. Puppeteer Limitations https://webscraping.ai/faq/puppeteer/what-are-puppeteer-s-limitations
7. Orcale Fusion Middleware Security https://fusionsecurity.blogspot.com/2011/04/oam-11g-single-sign-on-and-oam-11g.html