jQuery Vulnerability Mitigation & Migration Strategy

# Executive Summary

This document outlines our strategy to address the vulnerability identified in the older jQuery library during the recent VAPT (Vulnerability Assessment and Penetration Testing) report. We have taken immediate action to neutralize this client-side risk, and we have a clear, long-term plan for a full UI migration. The immediate security measures, combined with the inherent security of our application's architecture, ensure a secure and stable go-live.

# Understanding the Issue in Context

The VAPT report highlighted the use of an outdated jQuery library, which contains known client-side vulnerabilities like Cross-Site Scripting (XSS) and prototype pollution attacks. It is crucial to distinguish this from a server-side or API vulnerability.  
  
Because our application is built on JSP + Spring MVC, the business logic and database are never directly exposed to the internet — they remain behind the server. The identified jQuery issue is limited to the client-side, meaning our critical server-side components (authentication, business rules, database access) are unaffected.

# Immediate Security Measures

To ensure a secure go-live and provide a robust defense against the identified vulnerability, we have already implemented the following layers of protection:

## Content Security Policy (CSP)

A strict CSP header is now deployed. This policy controls which scripts can run on the page, effectively blocking malicious code injection and mitigating the risk of XSS attacks.

## Input Sanitization

All user input is rigorously sanitized server sides. This proactive measure prevents malicious script tags from being processed, neutralizing the payload before it can cause harm.

## Server-Side Validation

All data received from the UI is subject to a second, more rigorous layer of validation on the server. This is our definitive security check, ensuring that only correctly formatted and safe data is ever processed by the business logic or written to the database.

## Secure HTTP Headers

We have configured the server to include additional security headers such as X-Frame-Options and X-Content-Type-Options. These headers provide extra protection against attacks like clickjacking and prevent the browser from misinterpreting a file's content type.

## Strict CORS Policy

A Cross-Origin Resource Sharing (CORS) policy has been implemented to restrict API access to only a predefined list of trusted domains. This prevents unauthorized websites from making requests to our API, adding another layer of defense against malicious actors.

These measures significantly reduce the attack surface and provide a strong security posture for the current application.

# Long-Term Solution: The UI Migration Plan

While the immediate fixes are a strong defense, our long-term plan is a full migration of the UI and its technology stack to a modern framework that eliminates the vulnerable library entirely.  
  
Strategic Approach: We will adopt a phased rollout, migrating critical sections of the application first to ensure continuous security and a smooth transition.

# Conclusion

The vulnerability is a known client-side issue, and our team has implemented a robust set of security controls to mitigate it effectively for go-live. We are confident that this two-pronged approach—immediate risk reduction combined with a strategic, long-term migration plan—fully addresses the identified vulnerability and ensures the security of the platform.