Fortis Games Prompts - Q&A

QUESTION 1 (No Coding Required)

Prompt: *Describe a situation in which you had a radically different opinion from a colleague, or strongly disagreed with a technical direction on your team. What led to the difference, and what steps did you take to resolve it?*

Answer: While working on the BRP Ecommerce team, we needed to implement performance testing to ensure our platform could handle a surge in user traffic. The initial proposal was to spend $60k USD on a consulting firm to create performance tests. I firmly believed we could avoid this expense by leveraging our in-house QA team and open-source tools.

I had a QA Specialist on my team who suggested using Gatling because he had some familiarity with it, although he had concerns about JMeter's scalability. In contrast, I believed that due to our aggressive timeline, the team's prior knowledge, and JMeter's robust ecosystem, using JMeter—potentially enhanced with Selenium to simulate user actions—was a more practical approach. My colleague doubted this combination, feeling it was unconventional and unfamiliar.

To resolve this difference in opinion, we agreed to try both approaches in parallel. While my colleague explored Gatling and Scala-based scripting, I rapidly developed a proof-of-concept using JMeter alongside Selenium. Within a few days, I was able to show a running test suite that integrated easily into our existing toolchain, required minimal extra training for the team, and closely simulated realistic user workflows.

Faced with Gatling's complexity and slower progress, my colleague was surprised and impressed by how quickly the JMeter+Selenium solution came together. Recognizing its immediate value and ease of adoption, he agreed to pivot entirely towards my approach. This allowed us to build our performance tests internally, save money, meet deadlines, and ensure the team fully understood and could maintain the solution moving forward.

To resolve this, I proposed a structured approach:

1. **Defining Evaluation Criteria:** We first aligned on what mattered most—timeliness, ease of adoption, skill set familiarity, and integration capabilities. By clearly outlining these criteria, we had objective measures rather than relying on personal preference.
2. **Proof of Concept (PoC):** We agreed to develop two quick prototypes over a set time frame: one using Gatling and another using JMeter+Selenium. This allowed us to concretely compare the complexity, setup time, and integration potential of each tool in a realistic scenario.
3. **Demonstration and Review:** After a few days, I showcased a fully functioning JMeter+Selenium setup already integrated with our pipeline and simulating user actions effectively. In contrast, the Gatling implementation struggled with its Scala-based scripting and had limited demonstrable progress in the same time frame.
4. **Final Decision:** Seeing immediate, tangible results and the quick learning curve of the JMeter+Selenium solution, my colleague recognized its advantages. We agreed to adopt JMeter+Selenium, allowing us to build our performance tests in-house, meet deadlines, and avoid unnecessary costs.

By establishing objective criteria, running parallel PoCs, and openly reviewing results, we made a data-driven decision that ultimately won my colleague's support and positioned our team for long-term success.

QUESTION 2 (No Coding Required)

Prompt: *Given the example dashboard https://free.minimals.cc/, outline how you would build the test automation framework for it. Discuss what framework you'd use and why, how you would structure tests for maintainability and reusability, where selectors would be stored, and how to provide test results.*

Answer:

1. **Choice of Framework:**
   I would choose **Cypress** as the automation framework. Cypress offers a fast and reliable way to test modern web applications. It has a simple API, automatic waiting, and great debugging tools. It also integrates well with CI/CD systems and provides built-in screenshots and videos, making it easy for the team to review results.

2. **Test Structure for Maintainability & Reusability:**
   I would structure the tests using a **Page Object Model (POM)** or a similar pattern. For each page or component of the dashboard (e.g., Product Page, Dashboard Page, User Profile Page), I'd create a dedicated file containing the selectors and actions associated with that page. Each test would then import these page objects, making tests more readable and maintainable. This structure would allow the team to easily update a single page object file if selectors change rather than updating every test file.

   - Example Structure:

     ```
     cypress/
       e2e/
         login.cy.js
         product.cy.js
         dashboard.cy.js
       selectors/
         loginSelectors.js
         productSelectors.js
         dashboardSelectors.js
       support/
         commands.js
         e2e.js
     ```

3. **Where to Store Selectors:** I would store selectors in separate `selectors/` files. For instance, `loginSelectors.js` would export all selectors used on the login page. This keeps all locators in a central place, making it easy to maintain when the UI changes. The test files or page object files would import these selectors, ensuring consistency and minimizing duplication.

4. **Capturing and Providing Test Results:** Cypress by default provides a command-line report with pass/fail status. For richer reporting, I could integrate a reporter like `mochawesome` which generates HTML reports. These reports would be part of the CI pipeline artifacts. The CI (e.g., GitHub Actions or

Jenkins) would display test results, and the HTML report could be stored as a build artifact for easy access.

Additionally, Cypress video recordings and screenshots (taken automatically on test failures) would be available for debugging. With these artifacts, the team can quickly gain insights into test failures and application quality.

QUESTION 3 (Coding Exercise)

Prompt:
Objective: Develop a small-scale automation script to validate the functionality of a random date generator website.

Guidelines:

- Create or provide a public GitHub repository to store your code and share the outcome.
- Use the automation framework of your choice.
- Base your script on this website: https://www.random.org/calendar-dates/
- Be mindful of your test organization, selector location, etc.

Task:

1. On *'Step 1 : The Dates'*, pick a total of 4 random dates.
2. In the dropdown, select the date between the *5th of January 2024* and the *25th of November 2025*.
3. Press *'Get Dates'* on Step 4.
4. Assert that the result returned 4 dates.
5. Assert that the queried date range is correctly displayed in the results line, specifically checking the displayed dates.

Answer: The implementation for this exercise can be found in my public GitHub repository. Please refer to the provided Git link for the complete solution and instructions on how to run the tests in the README.md

https://github.com/Speelcraft/fortis-qa-challenge
