Answers – Tekna QA Assignment

> Prepared and completed by **Joyce Cervantes**.

Key Testing Points Identification

What are the main aspects you would validate when testing the Cost Calculation feature?

When testing the Cost Calculation feature, the first thing I would validate is whether the calculation formula itself is accurate and based on the correct logic, combining daily distance, fuel efficiency, and fuel prices. I would also test how the system handles decimal values and rounding, especially in borderline cases that could slightly shift the final results.

Another key point is ensuring the calculation updates instantly when a user changes input values, this is essential for a smooth and transparent experience. I'd also verify if unit conversions (km-miles or BRL-USD) are consistent and clearly communicated, especially if the platform is used in different regions. Finally, I'd test how the system behaves under poor connectivity or heavy usage, since users may access the platform on mobile or slower networks.

Identify two potential issues when testing the Savings Estimation feature.

One potential issue is **unrealistic savings due to invalid input**. For example, if someone enters 5,000 km/day, the estimated savings would spike to an absurd value, which could mislead the user or damage the platform's credibility. To prevent this, I'd recommend setting logical input boundaries and showing helpful messages when values exceed real-world scenarios.

Another issue is how the results are presented. A statement like "You save $1,800" means little without context. Users might wonder: "per year? Compared to what?" I'd suggest adding breakdowns (monthly/annually), side-by-side comparisons, and even visual elements like graphs or badges to improve clarity and engagement.

Implementation Planning

Here is my proposed two-phase development and testing approach, designed to prioritize stability and early validation.

Phase 1 – Inputs & Selection

- **What to implement:**
  - Vehicle Selection
  - Driving Habits Input

- **Why:** These features form the foundation of the platform and enable early testing of user interactions and data validation before deeper logic is introduced.

- **QA focus:**
  - Manual smoke tests for input flows
  - Validation for required fields and input formats
  - UI responsiveness on different screen sizes

Phase 2 – Processing & Output

- **What to implement:**
  - Cost Calculation
  - Savings Estimation

- **Why:** These features depend on clean, validated input and represent the platform’s main value to the user.

- **QA focus:**
  - Regression testing after each build
  - Cross-checking results with manual calculations or spreadsheets
  - Beginning automation for repeatable and critical calculations

Critical Analysis and Improvements

Would you add any tests or validations not mentioned? Why?

Absolutely. I would include:

- Accessibility checks to ensure the platform supports screen readers and keyboard navigation. Many QA teams overlook this, but I believe it's part of delivering inclusive products.
- Mobile responsiveness testing, especially since many users will likely access the platform via mobile devices.
- Negative testing, such as inputting letters in numeric fields, leaving fields blank, or submitting partial data.
- Visual regression testing to make sure UI updates don't break alignment, layout, or responsiveness.

- Usability feedback from non-technical users to catch pain points that automation can’t detect.

What problems could arise from using user-entered data? How would you prevent them?

Working with user input always brings unpredictability, and that’s where good QA shines. Users might type text where numbers are expected (“thirty” instead of “30”), enter decimal points with commas (common in Brazil), or just leave fields blank. I’ve even seen users add units like "km" inside a numeric input, and the system accepted it.

To reduce these risks, I’d apply strong input validation using regex, masks, and limiters. I’d also design validations to guide, not block, with real-time error messages, placeholder hints, and contextual help. Combining clear messaging with input thresholds helps ensure data integrity while keeping the experience smooth and frustration-free.

Automation Process

To automate regression tests properly, I’d start by identifying the core flows that users repeat most: vehicle selection, driving habits input, cost calculation, and savings estimation. These are not just the backbone of the platform, they’re also the most likely to be affected by future updates and refinements.

I would use Cypress as the primary automation framework. It’s fast, intuitive, and integrates well with JavaScript-based frontends, and I’ve been exploring its real-time debugging and selector strategies in my own QA learning projects. With Cypress, I’d cover both happy paths and edge cases, ensuring that validation, visual rendering, and result accuracy all hold up under automation.

I'd also integrate the tests into a CI/CD pipeline using GitHub Actions, so the regression suite runs automatically on every commit or pull request. This continuous feedback loop reduces the risk of pushing bugs to production and improves team velocity.

Lastly, I’d include automated visual checks to detect UI changes that might go unnoticed, especially in the comparison panels where clarity is key.