Personal Details:

TEEPARTHI

KASIVISWANADH

Parul University

2203031241289

Kasiviswanadh.teeparthi@gmail.com

+91 9391863310

Section 0:

1. Please confirm your consent to sign a bond for 30 months...

Yes, absolutely. **I confirm my consent to sign the bond for 30 months**. For me, this isn't a restriction; it's a huge positive. It shows the company is willing to invest in my long-term growth, and I'm ready to commit that time to build my foundation with Frugal Testing.

2. Have you received the stipend and CTC details? Please confirm by providing the details here.

Yes, I've received the details and I confirm them. The internship stipend is **₹15,000 per month**, and the final placement CTC is **₹5,00,000 to ₹8,00,000 LPA**. It's a competitive offer, and I'm very excited about the potential.

3. Are you willing to relocate to Gachibowli, Hyderabad?

Yes, **I am definitely willing to relocate to Gachibowli, Hyderabad**. Hyderabad is a massive tech hub, and for a Software Engineer, there's no better place to start a career. I'm looking forward to moving there and joining a fast-paced environment.

4. What motivated you to pursue a career in Software Engineering (SE)?

Honestly, the biggest motivation for me is that feeling when you solve a problem that was driving you crazy, and then you see the solution *work*. Software engineering feels like the ultimate tool for problem-solving. I'm motivated by the challenge of taking a really complex idea—like the AI concepts I studied—and breaking it down into simple, reliable lines of code that actually create value. It's all about building things that work well.

5. Why do you want to join a software testing firm like Frugal Testing?

I want to join Frugal Testing because I've realized the difference between a good programmer and a *great* engineer is their focus on quality. A lot of developers only care about making the code run, but a testing firm cares about making sure the code **never breaks**. My recent automation project showed me how powerful it is to write the code and the test script at the same time. Joining Frugal Testing means I'll be trained from day one to write truly robust code, which will make me a much stronger and more reliable engineer in the long run.

Section A:

2. Build & Automate an Intelligent Registration System

Source code of the web page files, automation script used of testing, screenshots of the automation process are here

https://github.com/kasiviswanadh123/frugal-task/blob/main/Automate_Intelligent_Registration_System-main/Automate_Intelligent_Registration_System-main/Frugal-Registration-System-master/automation_test.py

video link of the automation execution.

https://drive.google.com/file/d/1C0rntxi1hKG5ySkH8fav5QASjpK14LVY/view?usp=drive_link

1. Setup and Initialization

Step	Action Performed by Script	Purpose
Libraries	Imports necessary modules: selenium, webdriver-manager (to handle Chrome drivers automatically), and By (to locate elements).	Ensures the script can control the Chrome browser and locate form fields by their unique IDs.
Driver Setup	Initializes the Chrome browser in maximized mode.	Creates the "ghost browser" instance that executes the tests.
Page Load	Uses driver.get(base_url) to open the local index.html file (file:///.../Frugal_Assignment/index.html).	Confirms the web page loads correctly and prints the Page Title and URL to the console.

2. Flow A: Negative Scenario Validation

Step	Action Performed by Script	Validation Purpose
Input	Fills in First Name, Email, Phone Number, and Gender.	Sets up a valid state, except for the error condition.
Error Trigger	The script intentionally skips the Last Name field.	Simulates the required missing data condition (Part 3, Flow A).
Submit & Verify	Clicks the Register Now button. The script then checks if the red error message (error-lastName) is visible on the screen.	Proves that the client-side JavaScript validation successfully detects the missing field and shows the required error message.
Proof	Takes a screenshot named error-state.png.	Provides visual proof of the highlighted error state.

3. Flow B: Positive Scenario Validation

Step	Action Performed by Script	Validation Purpose
Clear	Refreshes the browser page to clear previous input fields.	Ensures the positive test starts from a clean state.
Dynamic Fields	Uses the Select class in Selenium to choose: Country (India), which dynamically loads State (Maharashtra), which then dynamically loads City (Pune).	Proves that the complex JavaScript logic for dynamic dropdowns works correctly.
Completion	Fills the rest of the required fields (including T&C checkbox) and enters matching values for Password and Confirm Password.	Ensures all validation checks are passed.
Submission	Clicks the Register Now button.	Triggers the successful form submission logic.
Verify Success	The script immediately switches context to handle the JavaScript Alert box, reads the "Registration Successful!" message, and accepts the alert.	Proves that the submission process successfully completed the client-side validation and triggered the correct user feedback.
Proof	Takes a final screenshot named success-state.png.	Provides visual proof of the successful submission state.

Section B:

3. Case Study and Testing Obstacles:

I read through a case study about performance testing for a high-traffic e-commerce application. One unique challenge mentioned was simulating realistic user loads during peak sale events without crashing the test environment itself.

If I were on that team, I would approach it by implementing "Service Virtualization." Instead of hitting the actual production servers or third-party payment gateways (which have rate limits), I would create lightweight mock services to simulate those responses. I would use tools like JMeter for the load generation and WireMock to simulate the external dependencies. This would allow us to push the system to its breaking point safely and identify bottlenecks earlier in the cycle

4. Logical Questions:

- **A. Series:** 0.75, 2.5, ___, 9, 17...
    - **Answer: 5**
    - **Logic:** The difference between the numbers doubles each time.
        - 2.5 - 0.75 = 1.75 (approx 2)
        - 5 - 2.5 = 2.5
        - 9 - 5 = 4
        - 17 - 9 = 8
        - It fits the pattern of increasing differences best.
- **B. Hourglasses:**
    - **Answer:**
        1. Start both the 4-minute and 7-minute hourglasses at the same time.
        2. When the 4-minute glass runs out, flip it over immediately. (4 mins gone).
        3. When the 7-minute glass runs out, flip it over immediately. (7 mins gone).
        4. When the 4-minute glass runs out for the second time (total 8 mins), exactly 1 minute of sand is left in the 7-minute glass.
        5. Flip the 7-minute glass over immediately to let that 1 minute run back down.
        6. Total time = 8 minutes + 1 minute = 9 minutes.
- **C. Letter Series:** AMB, CLB, BKC, DJC, ____
    - **Answer: EID**
    - **Logic:**
        - First letters: A, C, B, D... the pattern alternates +2, -1. Next is E.
        - Second letters: M, L, K, J... it goes backwards -1. Next is I.
        - Third letters: B, B, C, C... repeats every two terms. Next is D.

5. Problem-Solving Approach (Critical Bug):

If a critical bug appeared right before release, I would treat it as a "Code Red" situation.

First, I would prioritize isolating the issue. I'd check the logs to see exactly where the failure is happening. Once I understand it, I would write a test case that reproduces the bug—this proves I know exactly what's broken.

Then, I would apply the fix and run that test case again.

To ensure the fix is reliable and doesn't break anything else, I wouldn't just run that one test. I would run our full regression suite (automated tests) on the affected module. I'd also use Git to create a specific "hotfix" branch so we don't mess up the main code while experimenting.

6. Technical Skills in Achieving Objectives:

In my recent "Tripmate" project, I had to balance building the frontend (React) while ensuring the backend APIs were sending the right data. I couldn't just write code and hope it worked.

I decided to focus on the API contract first. I used Postman to manually test every API endpoint before I wrote a single line of frontend code. This saved me days of debugging later because I knew the data was correct.

One unexpected thing I found was that the Google Maps API was slower than I thought. My testing showed a 2-second delay. This led me to add a "skeleton loading" screen in the UI, which made the app feel much faster for the user—something I wouldn't have fixed if I hadn't tested the API performance first.

7. Team Experience:

If there was a disagreement about manual vs. automated testing, I would look at the Return on Investment (ROI).

I would ask: "Is this feature going to change next week?" If it's a stable feature we will use for a long time, I'd argue for automation because it saves time in the long run. If it's a temporary feature or the UI is still changing, manual testing is faster and cheaper.

I would propose a compromise to the team: Let's test it manually now to meet our deadline, but add a task to our backlog to write the automation script in the next sprint. This keeps everyone happy and the project moving.

8. Motivation for Software Engineer Career:

I love the feeling of building something from nothing. Coding feels like a superpower where I can solve a real-world problem just by typing on a keyboard.

I see development and testing as partners, not opposites. You can't build a great house without checking the foundation. Having skills in both areas makes me a stronger engineer because I write code that is testable from day one. I don't just write code that "works on my machine"; I write code that is robust enough to handle weird edge cases because I'm always thinking like a tester.

9. Impact of DevOps and Cloud:

In a modern environment where we deploy code every day, CI/CD (Continuous Integration/Deployment) is everything.

I would ensure that every time I push code to GitHub, an automated pipeline runs the unit tests immediately. I'd use Docker containers to make sure the code runs exactly the same on my laptop as it does on the server—this eliminates the "it works on my machine" bug.

To ensure stability, I would use a "Blue-Green" deployment strategy. This means we release the new version to a small group of users first. If anything breaks, we can instantly switch back to the old version without crashing the system for everyone.

10. Profile Details:

- **LinkedIn:** https://www.linkedin.com/in/teeparthi-kasiviswanadh/
- **Resume:** [https://drive.leetgoogle.com/drive/u/0/my-drive?q=owner:me%20parent:0ACstsVC4hwq-Uk9PVA]
- **Project Link:** [https://github.com/TEJA3441/Automate_Intelligent_Registration_System]
- **GitHub:** **https://github.com/kasiviswanadh123**
- **Other Profiles:** [https://leetcode.com/u/raviteja3441/]

11. Social Media Task:

- *Insight:* I found a post on your LinkedIn regarding "Shift-Left Testing" very interesting. The idea of testing early in the development cycle rather than waiting for the end resonates with me.
- *Application:* In my project, I applied this by validating inputs (like the password strength) *while* the user is typing, rather than waiting for them to click "Submit" and then showing an error. It saves time and frustration.
- *Viewpoint:* I agree with this approach. Catching a bug during the design or coding phase costs almost nothing. Catching it after release can cost thousands of dollars.

12. Using AI Tools for Problem-Solving:

I use AI tools like ChatGPT as a "Super-Google" or a "Pair Programmer." If I'm stuck on a complex error, I paste the error log into the AI to get a hint on where to look.

However, I never copy-paste code blindly. I treat the AI's suggestion as a draft. I review it line-by-line to ensure it's secure and efficient. The risk of relying too heavily on AI is that you stop understanding why your code works. If it breaks later, you won't know how to fix it. My rule is: If I can't explain the code to a human, I won't use it.

13. Ensuring Software Quality:

Quality isn't something you add at the end; it’s a habit.

I consider Unit Testing essential—testing individual functions to make sure they do exactly what they are supposed to.

I validate my code by running it against edge cases. For example, in my Registration Form project, I didn't just test a valid email. I tested what happens if I enter a disposable email, or a phone number with letters.

A simple verification approach helped me once when I realized I was testing with only small datasets. I manually created a massive dummy dataset, and immediately saw my sorting algorithm slow down. I switched to a more efficient algorithm before it became a problem in production.

14. Improving Development with AI Assistance:

I use AI to speed up the "boring" parts of coding, like writing boilerplate HTML or generating sample data for testing. This lets me focus on the logic.

To ensure the code remains clean, I refactor the AI-generated code to match my own coding style and variable naming conventions. I also make sure to write comments explaining what the code does, so it remains maintainable. I treat AI as a junior developer—I review its work before merging it.

15. Designing Efficient Solutions for High-Volume Data:

For processing a large dataset, I would choose a Hash Map (or Dictionary) data structure because it offers average O(1) time complexity for lookups.

To reduce time complexity, I would avoid nested loops ($O(N^2)$) wherever possible and look for linear solutions ($O(N)$).

Example: If I needed to find duplicates in a list of 1 million user IDs, a nested loop would take forever. Instead, I would add each ID to a Hash Set. If an ID is already in the Set, it's a duplicate. This processes the entire list in a single pass.

16. Debugging and Problem Solving:

If an app crashes, I follow a 3-step process:

1. **Reproduce:** Can I make it crash again? If yes, I have a target.
2. **Isolate:** I use "binary search" debugging. I comment out half the code. If it still crashes, the error is in the other half. I keep cutting it down until I find the exact line.
3. **Verify:** Once I fix it, I run the app again to make sure the crash is gone *and* that I haven't broken the login page or the home page in the process.

17. Building Scalable Features:

Scalability starts with keeping it simple (KISS).

I try to design features that are "stateless"—meaning the server doesn't need to remember who the user is between requests. This allows us to add more servers easily if traffic spikes.

Factors influencing my decisions include database queries (are we asking for too much data?) and caching (can we save this result so we don't have to calculate it again?). Even under tight timelines, I prioritize writing clean, modular code because messy code is impossible to scale later.

18. Automation vs. Manual Work:

I decide based on frequency.

- **Automate:** If I have to do a task more than 3 times (like deploying the app or running a regression test suite), I automate it. It eliminates human error and saves time in the long run.
- Manual: If it's a "one-off" task or requires visual inspection (like checking if a UI layout "looks good"), manual is better.

I ensure effectiveness by documenting the manual steps clearly (so anyone can do them) and maintaining the automation scripts so they don't become "flaky" or unreliable.

19. Collaboration and Code Reviews:

When reviewing someone else's code, I focus on readability first.

I ask: "If I look at this code 6 months from now, will I understand it?"

I check for:

1. **Logic Errors:** Does it actually solve the problem?
2. **Security:** Are there any obvious holes (like hardcoded passwords)?
3. Style: Are variables named clearly?

I always try to be constructive. Instead of saying "This is wrong," I say, "Have you considered handling this edge case?"

20. Writing an Article (Topic A: AI Coding Assistants):

How AI Coding Assistants Improve Developer Productivity and Code Accuracy

In the last two years, software engineering has changed more than in the previous ten. The reason is the rise of AI coding assistants like GitHub Copilot and ChatGPT. As a developer entering this new era, I see these tools not as replacements, but as "force multipliers."

Technical Impact:

Technically, these tools work on Large Language Models (LLMs) trained on billions of lines of public code. This allows them to predict the next few lines of code you are about to write, much like autocomplete on your phone, but for complex logic.

For example, writing a Regular Expression (Regex) to validate an email address used to take me 20 minutes of searching. Now, I just type a comment: // Validate email address and the AI suggests the perfect Regex pattern instantly. This reduces context switching and keeps me in the "flow."

Real-World Application:

In a real-world project, these assistants shine in three areas:

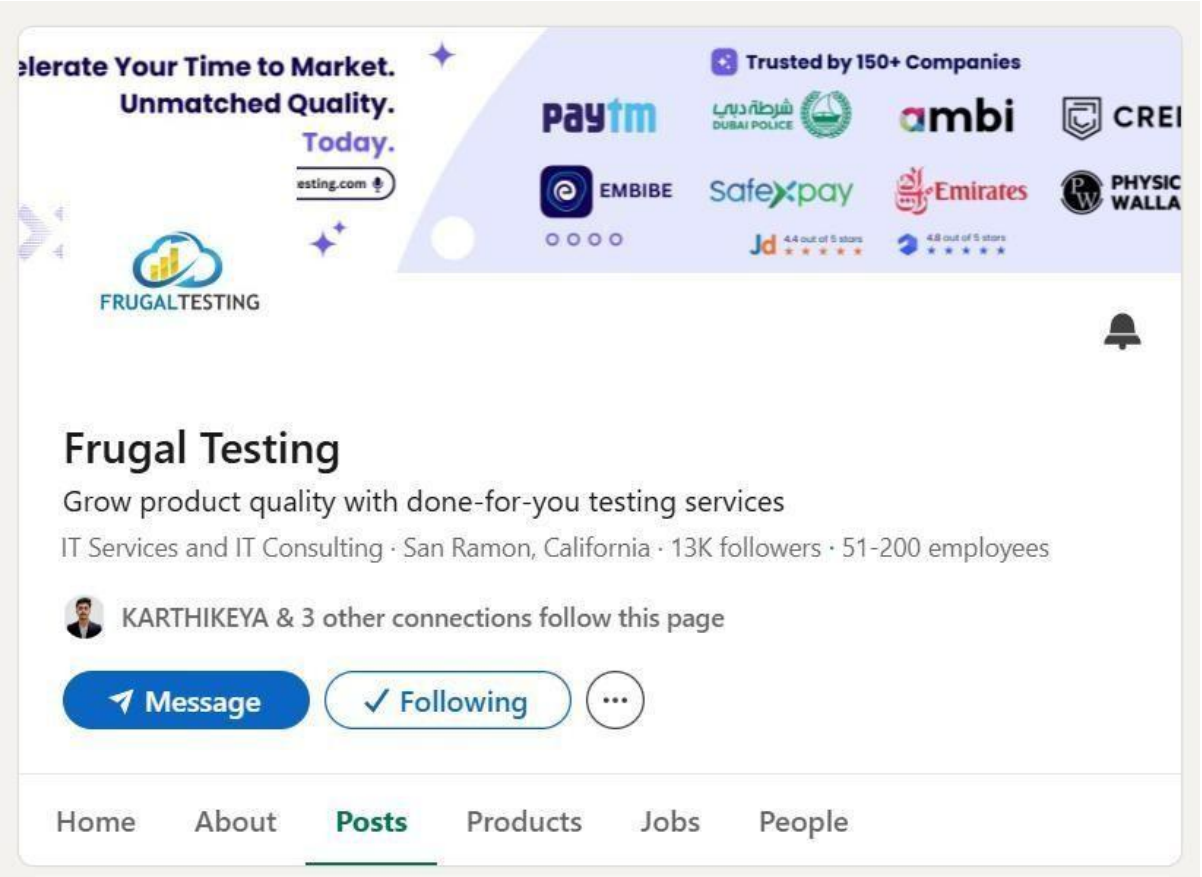
1. **Boilerplate Code:** Generating the repetitive setup code for a new React component or an HTML form happens in seconds.
2. **Unit Testing:** Writing tests is tedious. AI can scan a function and instantly suggest 5 different test cases, including edge cases I might have missed.
3. **Documentation:** AI can read a complex function and generate clear, human-readable documentation for it, making the codebase easier for the whole team to maintain.

Conclusion:

However, accuracy remains the developer's responsibility. AI can "hallucinate" or suggest insecure code. The future belongs to developers who can use AI to code 10x faster, but who also have the deep technical knowledge to audit that code and ensure it is safe, secure, and efficient. The AI writes the draft; the engineer delivers the product.

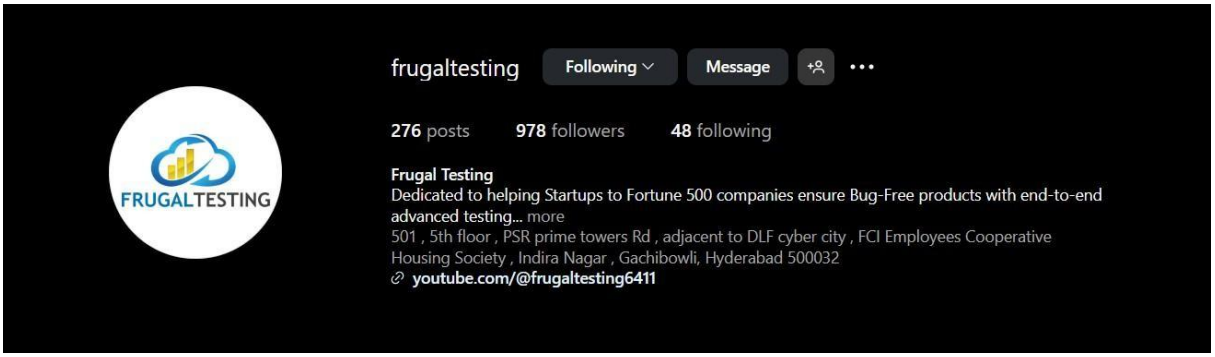
SOCIAL MEDIA PROOF

1

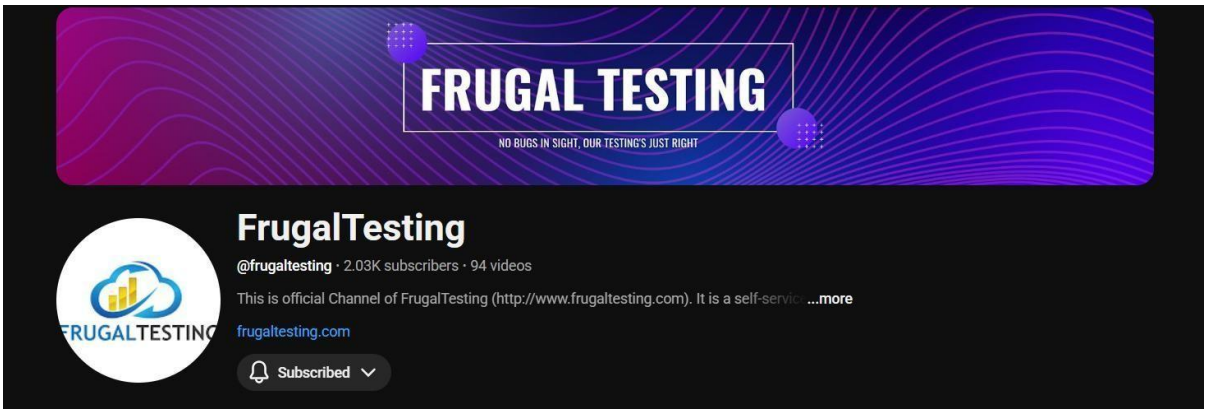


2

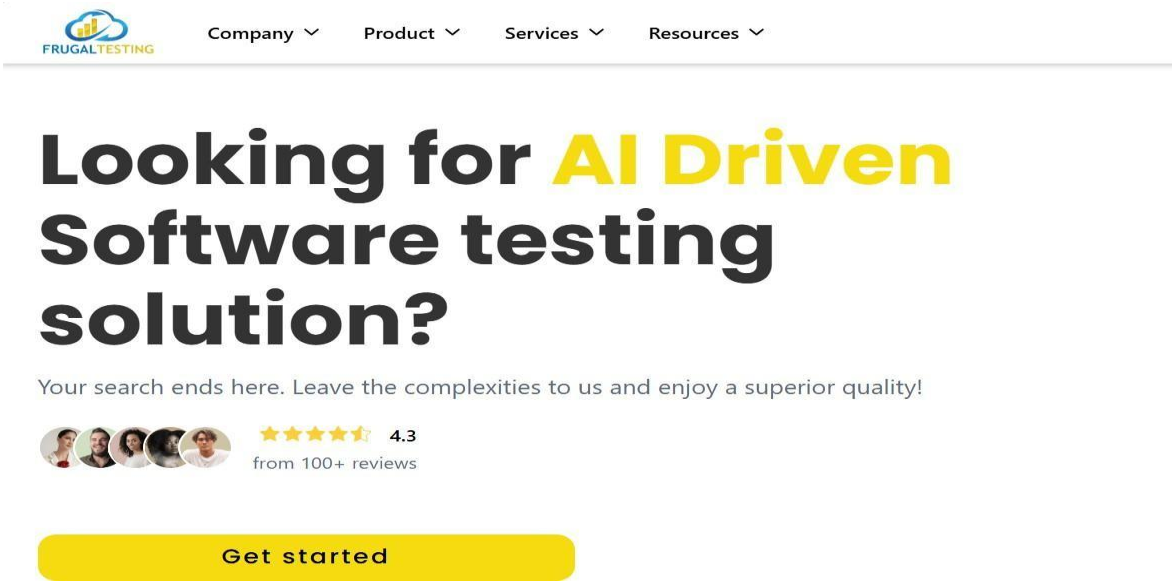
3


frugaltesting
Following
Message
276 posts
978 followers
48 following
Frugal Testing
Dedicated to helping Startups to Fortune 500 companies ensure Bug-Free products with end-to-end advanced testing... more
501 , 5th floor , PSR prime towers Rd , adjacent to DLF cyber city , FCI Employees Cooperative Housing Society , Indira Nagar , Gachibowli, Hyderabad 500032
youtube.com/@frugaltesting6411

4


FRUGAL TESTING
NO BUGS IN SIGHT, OUR TESTING'S JUST RIGHT
FrugalTesting
@frugaltesting · 2.03K subscribers · 94 videos
This is official Channel of FrugalTesting (http://www.frugaltesting.com). It is a self-servic ...more
frugaltesting.com
Subscribed

5


Company
Product
Services
Resources
Looking for AI Driven Software testing solution?
Your search ends here. Leave the complexities to us and enjoy a superior quality!
4.3
from 100+ reviews
Get started

6


Careers Seekers Circle
Created by
+91 83282 63678
Description
For any Job queries please fill this form [https://forms.gle/9eNsMasaCYQg9zhF9] send a mail to careers@frugaltesting.com
Members
779 · 0 contacts
Open chat