happiest minds
The Mindful IT Company
Born Digital . Born Agile

Adaptive Automation:

Empowering Seamless Testing Through Self-Healing Locators and LCS Algorithm

Abstract:

Healenium emerges as an advanced self-healing framework revolutionizing the stability and resilience of automated testing for mobile apps with Appium and web-based applications with Selenium. Healenium is a game-changer that easily integrates with a wide range of programming languages, (Java, Python, JavaScript, C#), and Appium for mobile app automation. This allows automated test suites to adapt to the constantly changing dynamic user interface (UI) of both web and mobile environments.

Its foundational architecture, orchestrated via Docker-compose, encompasses pivotal services including a PostgreSQL database housing crucial data, a proxy intermediary managing client requests, and a selector imitator streamlining the integration of healed locators within test scripts. This modular and scalable infrastructure ensures efficient communication among services, promoting modularity, scalability, and compatibility across both Selenium-supported web applications and Appium-supported mobile app testing scenarios.

Healenium offers a cohesive self-healing mechanism that strengthens the stability and reliability of automated test suites by aligning seamlessly with Selenium for web-based applications and Appium for mobile apps This transformative approach optimizes the testing process, ensuring consistent and reliable test results in the midst of dynamic UI changes, thereby revolutionizing the landscape of both web and mobile app automation testing.

I .Background

The Evolution of Automated Testing

The evolution of software testing has been propelled by the promise of automation—efficiency, reliability, and speed in validating software applications. With the advent of Selenium and Appium frameworks, the industry witnessed a paradigm shift towards automated testing for both web-based applications and mobile apps.

Locator Instability: A Pervasive Issue

Locators, crucial components in test automation scripts, serve as navigational markers, enabling the identification and interaction with elements within an application's user interface. Despite their fundamental role, locators are highly susceptible to instability. Dynamic changes within the UI, whether in web applications or mobile apps, disrupt the functionality of locators, leading to a cascade of test failures, false positives, and an increased burden of maintenance efforts.

Historical Efforts in Locator Stability

Numerous efforts have been undertaken within the testing community to mitigate the challenges posed by locator instability. Traditional approaches involve static locators such as XPath or CSS selectors hardcoded within test scripts. But when the UI changes dynamically, these static locators become less flexible and resilient, requiring more frequent updates and maintenance, which eventually undermines the efficiency and dependability that automated testing promises.

The Significance of Locator Stability:

In the context of automated testing, locator instability is a critical problem. Instable locators not only impede the efficiency of test automation but also contribute significantly to increased maintenance overheads, prolonged testing cycles, and unreliable test results. The inability to adapt to UI changes directly impacts the core principles of automation—accuracy, speed, and reliability—undermining the very essence of automated testing frameworks.

People, Technology, and the Testing Landscape

The dynamic nature of user interfaces, the proliferation of web and mobile applications, and the increasing demand for continuous integration and delivery have collectively exacerbated the challenges surrounding locator stability. Testing professionals, developers, and quality assurance teams grapple with the persistent issue of maintaining robust test suites amidst ever-evolving applications and interfaces.

Considering these challenges, the search for a solution that goes beyond the limitations of conventional locator handling mechanisms continues to be a priority. Healenium, an innovative self-healing framework, emerges as a ray of light for resolving the historical issue of locator instability in the context of mobile apps enabled by Appium and web-based applications evaluated by Selenium. This paper further explores the significance, implementation, and transformative impact of Healenium's self-healing mechanisms in the context of the prevailing challenges of locator instability in automated testing landscapes.

II. Introduction

In the realm of software testing, the quest for stability and reliability in automated testing remains a persistent challenge, particularly in the dynamic landscapes of web and mobile applications. The evolution of technology has ushered in a constant flux of changes within user interfaces, necessitating adaptive and resilient testing frameworks. It is within this context that Healenium emerges as a pioneering solution, transcending the boundaries of traditional testing methodologies and paving the way for a transformative approach to automated testing.

Healenium's Promise:

At the forefront of revolutionizing automated testing stands Healenium, a self-healing framework designed to fortify the stability and resilience of test suites across both web and mobile application domains. It transcends the limitations of conventional testing methodologies by introducing adaptive mechanisms that dynamically respond to UI changes. Healenium acts as a catalyst in fostering a symbiotic relationship between Selenium for web applications and Appium for mobile apps, ushering in a unified approach towards stable and reliable automated testing.

Unified Self-Healing Approach:

The core tenet of Healenium lies in its self-healing prowess. It operates as an adaptive layer, seamlessly integrated between testing scripts and the Selenium or Appium servers. Through an innovative architecture orchestrated via Docker-compose, Healenium orchestrates a suite of pivotal services: a PostgreSQL database storing critical data, a proxy intermediary managing client requests, and a selector imitator simplifying the integration of healed locators within test scripts. This unified self-healing approach stands as a testament to Healenium's adaptability and transformative capabilities, marking a paradigm shift in the testing landscape.

The Challenge of Locator Instability:

The heart of automated testing lies in the stability and reliability of locators within test scripts. However, the dynamic nature of user interfaces in both web-based applications and mobile apps poses a significant challenge—locator instability. Changes in UI elements, attributes, or structures lead to a ripple effect, causing locators to become invalid, resulting in test failures and false positives.

Healenium's Self-Healing Solution:

Healenium introduces a groundbreaking solution—a self-healing mechanism that dynamically responds to UI changes, ensuring the adaptability and resilience of test scripts. Integrated seamlessly into Selenium-based web applications and Appium-based mobile app automation, Healenium acts as an intelligent layer capable of identifying and healing invalidated locators in real-time.

Reduced Maintenance Efforts:

Healenium significantly alleviates the maintenance burden associated with locator updates. By autonomously healing locators, it reduces the need for manual intervention, allowing testing teams to focus on creating new tests rather than constantly revising existing ones.

Adaptable Locator Identification with Healenium's Modified LCS Algorithm:

The modified Longest Common Subsequence (LCS) algorithm within Healenium operates as a dynamic analytical tool tailored to cope with the inherent variability in web and mobile application interfaces. It forms the backbone of Healenium's adaptability by focusing on identifying the most extended sequence common to various attributes within the UI elements. This algorithm stands out for its advanced methodology, which incorporates weighted relevance for many factors like IDs, tags, classes, values, and other UI element features.

These weights are strategically assigned to reflect the varying significance of attributes, ensuring a nuanced understanding of their relevance in UI element identification.
In practice, Healenium's modified LCS algorithm acts as a detective, meticulously assessing the current state of the web or mobile application's UI. When confronted with a shift in the DOM structure or alterations in element attributes, this algorithm's adaptive nature comes into play. It dynamically recalculates its strategy on the basis of the changed attributes, giving the attributes given higher weights more weight.

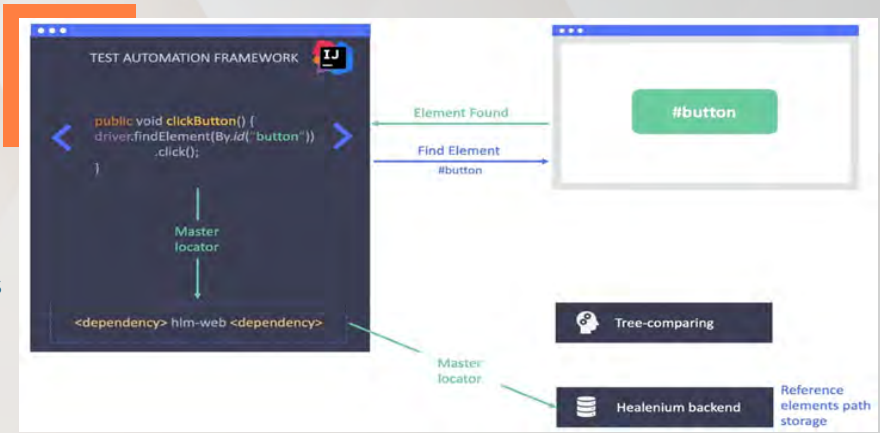
Consequently, when elements experience changes in position or receive new identifications, the algorithm intelligently generates a series of "healed" locators. These newly crafted locators encapsulate the algorithm's adaptive response, embodying the necessary adjustments in attributes and their weighted significance to maintain accurate and stable locator identification.
Healenium's LCS algorithm ensures that the generated healed locators remain resilient and reliable, offering a robust foundation for automated test scripts. Its ability to discern and prioritize attributes based on weighted importance empowers Healenium to adapt swiftly to changes in the UI, ensuring the continued stability and accuracy of locator identification despite the ever-evolving nature of web and mobile application interfaces. This nuanced approach not only mitigates the impact of UI alterations but also significantly enhances the adaptability and effectiveness of automated testing within diverse and dynamic application environments.

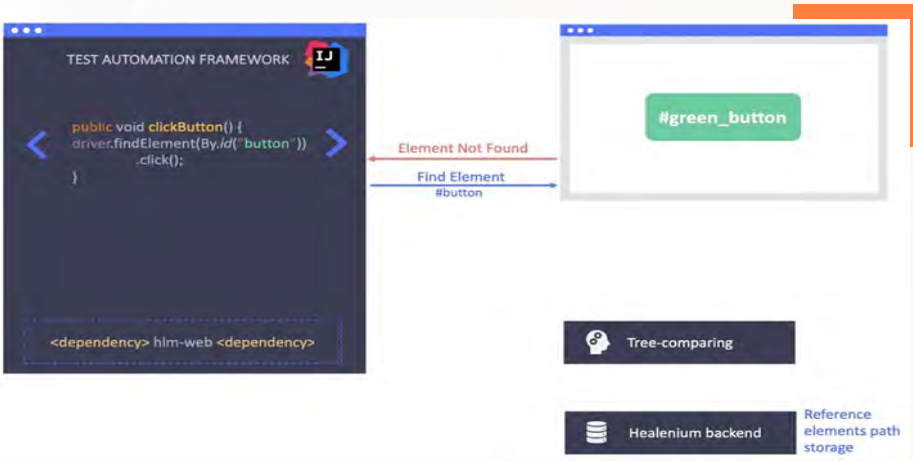
Healenium's Detailed Workflow

1. Automated Test Scenario Demonstration:

Scenario:
A test script attempts to locate an element on the UI using a specific locator, say `#button`.

Healenium Action:
Upon successful identification, Healenium stores this locator as a reference point for subsequent test executions.





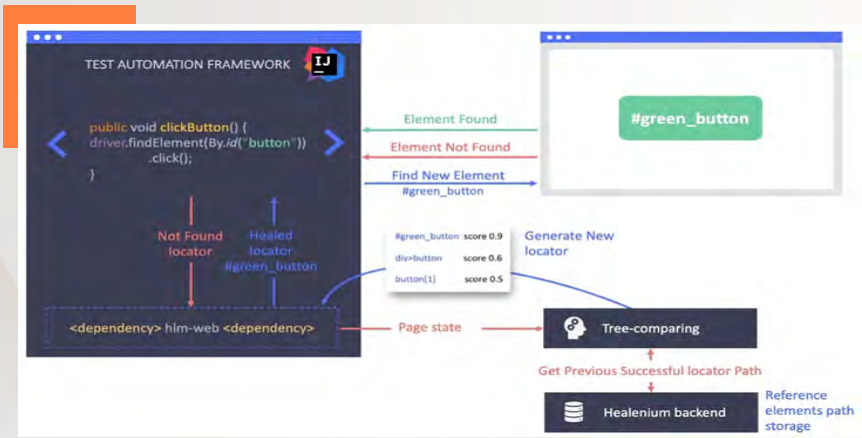
2. Triggering Healing Mechanism:

Scenario:
The UI undergoes changes, assigning a new ID, for instance, `#green_button`, unbeknownst to the test team who hasn't updated the automation code.

Healenium Response:
When the test script tries to locate the element with the old ID `#button` and fails, Healenium intercepts the `NoSuchElement` exception. It triggers its Machine Learning algorithm.

3. Machine Learning Algorithm Operation:

Healenium's Algorithm:
Utilizes a modified Longest Common Subsequence (LCS) algorithm, assigning extra weight to attributes like tag, ID, class, value, and other attributes. It dynamically identifies and matches possible locators, addressing changes in DOM structure or new element IDs



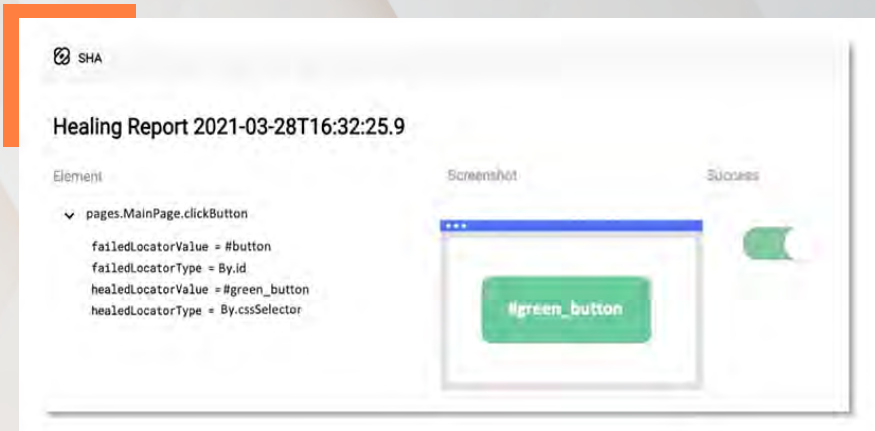
4. Locator Selection and Action Execution:

Healenium's Decision-making: Compares the current page state with the stored successful locator path. Generates a list of healed locators and selects the most suitable one based on scoring.

5. Reporting and Feedback Loop:

Healenium's Report Generation:
Post-test execution, Healenium generates a comprehensive report.

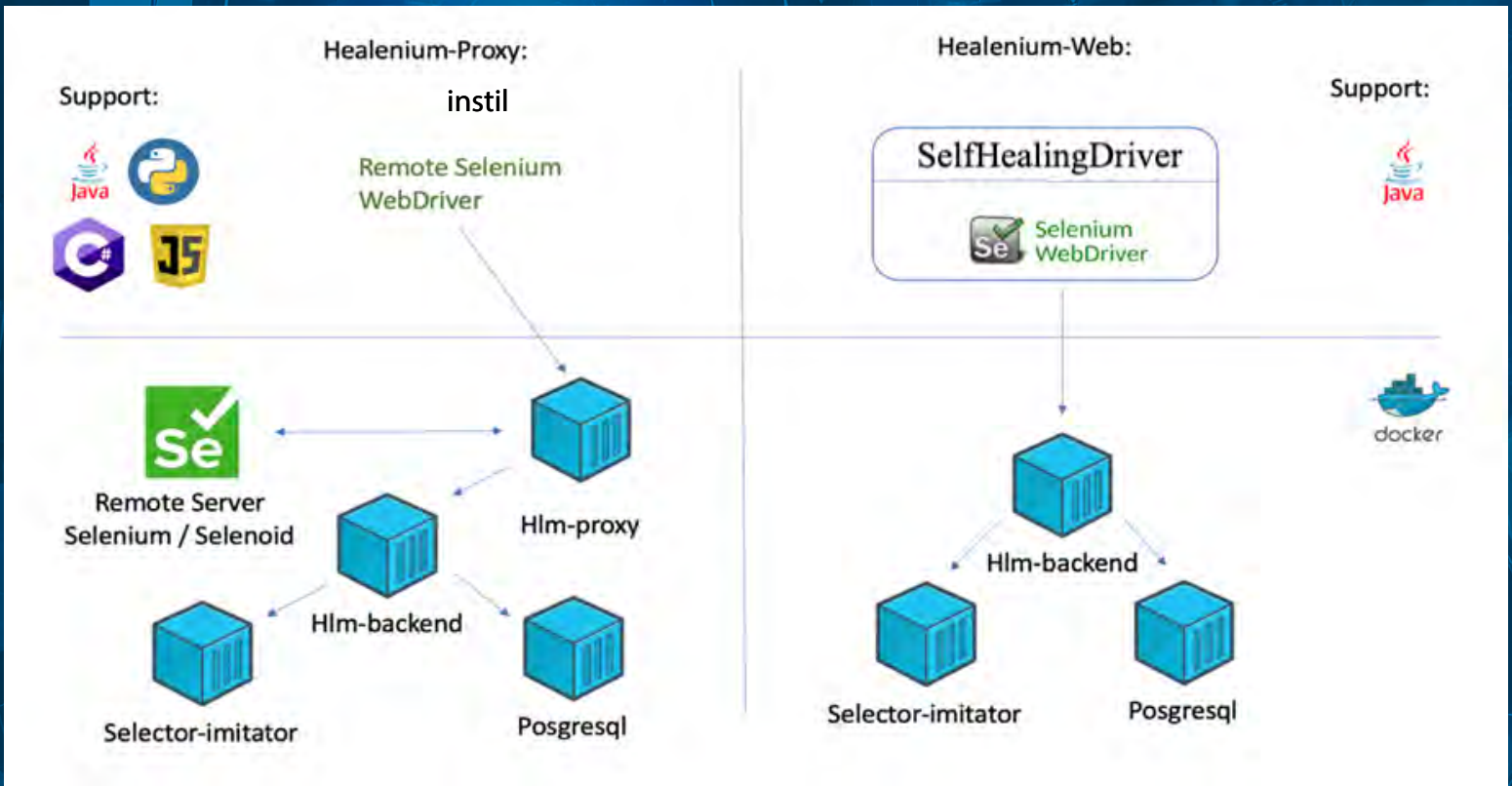
This report includes detailed information about the healed locator, screenshots capturing the successful action, and a feedback button for healing success.



6. Integration with Healenium Idea Plugin:

Plugin Functionality: The Healenium Idea plugin scans for healing results and seamlessly updates the test codebase with the healed locator, simplifying the incorporation of healed locators into automated test code.

IV. Architecture



Healenium Infrastructure Overview:

Components:

1. **postgres-db:**
   Description: PostgreSQL database dedicated to storing essential data including etalon selectors, healing records, and comprehensive reports.

2. **hlm-proxy**
   Role: Acts as an intermediary or proxy between the client (test scripts) and the Selenium server.
   Function: Manages and redirects client requests to the appropriate components within the Healenium infrastructure.

3. **hlm-backend:**
   Function: Provides the CRUD (Create, Read, Update, Delete) service functionalities, facilitating operations related to the stored data in the PostgreSQL database.

4. **selector imitator:**
   Purpose: Converts healed locators into a more user-friendly or convenient format, simplifying their utilization within the test automation scripts.

5. **selenoid/selenium-grid:**
   Role: Hosts the Selenium server, enabling the execution of automated tests in various browsers and environments.
   Function: Provides the underlying infrastructure for running tests, managing browser instances, and executing test commands.

Docker-Compose Services:

Interconnectivity: The Docker-compose configuration ensures seamless communication and interaction among these services, enabling them to work together as an integrated ecosystem.

Modularity and Scalability: The usage of Docker-compose allows for a modular setup, where each service operates independently while collaborating efficiently. Additionally, it facilitates scalability by enabling easy addition or removal of services as per the testing requirements.

Integration with Selenium:

Selenium Compatibility: Healenium is seamlessly integrated with Selenium, enabling compatibility with various programming languages supported by Selenium (such as Java, Python, JavaScript, C#).

Proxy Functionality: Acts as a proxy layer between the client and the Selenium server, intercepting and managing the requests and responses, thereby implementing the self-healing mechanisms.

This infrastructure design ensures that Healenium operates effectively as a self-healing framework, leveraging Selenium's capabilities while introducing the necessary components and services to facilitate the self-healing process within the test automation framework.

V. Advantages

Certainly, based on the provided information, your paper could encompass the following overall content:

Healenium's Business Value:
The integration of Healenium into automated testing frameworks brings forth transformative benefits, revolutionizing the landscape of test automation. Its impact extends across various critical facets, providing substantial value to businesses and test engineering teams alike.

Time and Resource Optimization:
Healenium's self-healing capabilities significantly reduce the time and resources traditionally invested in maintaining test automation code. This optimization empowers testing teams to allocate their efforts more efficiently focusing on crafting new tests and expanding test coverage, rather than constantly revising existing scripts.

Stability Ensured in Runtime:
One of Healenium's core advantages lies in its provision of runtime stability. By shielding automated End-2-End tests from the impact of UI changes, it ensures consistent and reliable test results, enhancing the overall stability of testing suites.

Impact on Quality Assurance:
Healenium's contribution to the accuracy of the quality feedback loop is substantial. By stabilizing tests and mitigating false positives due to UI alterations, it elevates the assurance that failures detected in the Continuous Integration (CI) pipeline are more likely to signify genuine product-related issues, ultimately improving overall product quality.

Cost Reduction and ROI:
The cost-effectiveness of Healenium stems from its ability to curtail test failures caused by UI changes. This reduction in false positives optimizes resource utilization, thereby amplifying the return on investment (ROI) in test automation endeavours.

Business Continuity and Confidence:
Healenium's provision of consistent and reliable results fosters confidence in the testing process, ensuring its reliability as a cornerstone in the software development lifecycle. This continuity boosts overall business confidence in the quality of the software.

Competitive Advantage:
Positioning Healenium as a facilitator of business agility, it enables organizations to navigate software development cycles swiftly. By mitigating the impact of UI changes on testing, Healenium facilitates faster releases, thereby establishing a competitive edge in the market.

VI. Case Study

Enhancing Testing Efficiency with Healenium's Self-Healing Locators

Context:
A leading software company, grappling with frequent UI changes impacting their automated testing suites, sought to bolster the stability and efficiency of their testing processes. Implementing Healenium's self-healing locators and Longest Common Subsequence (LCS) integration became a strategic imperative to address these challenges.

Implementation:
The company integrated Healenium into their existing Selenium-based testing framework across various web applications. The LCS algorithm, bolstered with attribute-weighted importance, dynamically adapted to UI alterations. Healenium's self-healing locators swiftly responded to changes, ensuring tests remained stable despite evolving UI elements.

Comparative Analysis:

Pre-Healenium Scenario:
Before Healenium, the company faced a 30% test failure rate due to UI changes, resulting in an average of 15 hours per week dedicated to code maintenance.

Post-Healenium Implementation:
After deploying Healenium, the test failure rate due to UI changes plummeted to 5%, representing a notable 25% reduction. The time allocated to code maintenance dropped to a mere 3 hours weekly, marking an 80% efficiency gain.

Efficiency Gains and Performance Improvements:

Stability Enhancement:
Healenium's intervention drastically stabilized the automated testing suites. The self-healing locators seamlessly adapted to UI alterations, ensuring consistent and reliable test results.

Resource Optimization:
The reduced need for code maintenance liberated the testing team's time. They redirected efforts towards crafting new tests, ultimately expanding overall test coverage.

ROI and Cost Reduction:
The minimized test failure rate and decreased code maintenance hours resulted in a notable cost reduction, amplifying the return on investment in test automation endeavours.

VII. Conclusion

Healenium proves to be a crucial tool in the field of test automation, significantly transforming the environment by tackling the enduring problems associated with user interface instability. Its introduction not only streamlines the efficiency of testing processes but also revolutionizes the very approach towards maintaining stability in automated testing frameworks. The substantial reduction in time and resources dedicated to code maintenance, thanks to Healenium's self-healing mechanisms, enables testing teams to redirect efforts towards innovation, fostering an environment conducive to the creation of new tests and the expansion of overall test coverage. Furthermore, the runtime stability Healenium affords shields automated tests from the turbulent impact of UI alterations, ensuring consistent and dependable outcomes, a cornerstone of any robust testing suite.

Moreover, the profound impact on quality assurance cannot be understated. Healenium's ability to weed out false positives resulting from UI changes significantly enhances the accuracy of the quality feedback loop, identifying genuine product-related issues and thereby elevating overall product quality. This, in turn, translates to improved cost-effectiveness, reducing the frequency of test failures due to UI alterations and optimizing resource utilization, ultimately amplifying the return on investment in test automation initiatives.

The resilience and reliability Healenium introduce in the testing process instils confidence in its continuity as a trusted pillar within the software development lifecycle. By promoting business continuity and enabling faster releases, Healenium bestows a distinct competitive advantage upon organizations, enabling them to navigate agile software development cycles with ease, positioning them as frontrunners in a rapidly evolving market landscape. In essence, Healenium's integration marks a paradigm shift in test automation, offering a holistic solution that empowers stability, efficiency, and innovation within testing framework.

ABOUT THE AUTHOR

POORNESH H

Automation Test Engineer,PDES

Poornesh is currently employed as an Automation Test Engineer. With over 1.5 years of experience, his expertise lies in automation testing using Java, Selenium, Appium, and Android application development with Kotlin. Additionally, he is proficient in GIT and TestNg. As a passionate mobile automation enthusiast, he has successfully built automation testing frameworks and has actively contributed solutions in this field. Poornesh finds great joy in problem-solving and possesses a deep interest in exploring new technologies, particularly in the areas of data structures and algorithms.

happiest minds
The Mindful IT Company
Born Digital . Born Agile

ABOUT HAPPIEST MINDS

Happiest Minds Technologies Limited (NSE: HAPPSTMNDS), a Mindful IT Company, enables digital transformation for enterprises and technology providers by delivering seamless customer experiences, business effciency and actionable insights. We do this by leveraging a spectrum of disruptive technologies such as: artificial intelligence, blockchain, cloud, digital process automation, internet of things, robotics/drones, security, virtual/ augmented reality, etc. Positioned as 'Born Digital. Born Agile', our capabilities span Product & Digital Engineering Services (PDES), Generative AI Business Services (GBS) and Infrastructure Management & Security Services (IMSS). We deliver these services across industry sectors such as automotive, BFSI, consumer packaged goods, e-commerce, EdTech, engineering R&D, healthcare, hi-tech, manufacturing, retail, and travel/transportation/hospitality. Happiest Minds is headquartered in Bangalore, India with operations in the U.S., UK, Canada, Australia, and the Middle East.

For more information, write to us at business@happiestminds.com