Jan 26 Read Time - 15 Mins.

Whitepaper

The Right-Shift Paradigm in Software Development

Dynamic Agentic Generated Applications (DAGA)

Executive Summary

The software industry is rapidly integrating Generative AI, yet this shift has focused almost exclusively on the speed of coding rather than the structure of applications. While GenAI has dramatically accelerated how software is written, it has not fundamentally changed how it is built or operated. Current implementations, primarily AI-powered pair programming, remain limited to generating static code, missing the opportunity to reimagine application architecture itself.

This paper introduces Dynamic Agentic Generated Applications (DAGA), a framework where AI agents generate user interfaces and middleware logic at runtime. We examine how shifting from static compilation to runtime generation can reduce technical debt, enable hyper-personalization, and address the limitations of traditional software delivery.

Content

AI Is Transforming Development—But Not Applications

Generative AI is reshaping how software teams work across domains, from mobile applications to large enterprise systems. Tools such as GitHub Copilot, AWS Q Developer, Windsurf, and others are revolutionizing the developer experience. They are now embedded into development workflows, assisting developers and architects in code generation, accelerating application lifecycles, and improving overall productivity.

One particularly impactful example is AI-powered pair programming, where agents collaborate with developers to generate code, fix bugs, run tests, and manage dependencies. As a result, enterprises worldwide are actively encouraging developers to adopt these AI-powered programming tools to boost efficiency and innovation.

However, this productivity leap has not translated into a fundamental shift in how applications are architected or how they behave once deployed. Architects continue to design systems using established methodologies, and developers implement these designs largely as static structures. AI accelerates coding but does not reshape the underlying application model. As a result, developers still produce large volumes of static code—often hundreds of thousands of lines, that must be maintained by application support teams, compounding operational complexity over time.

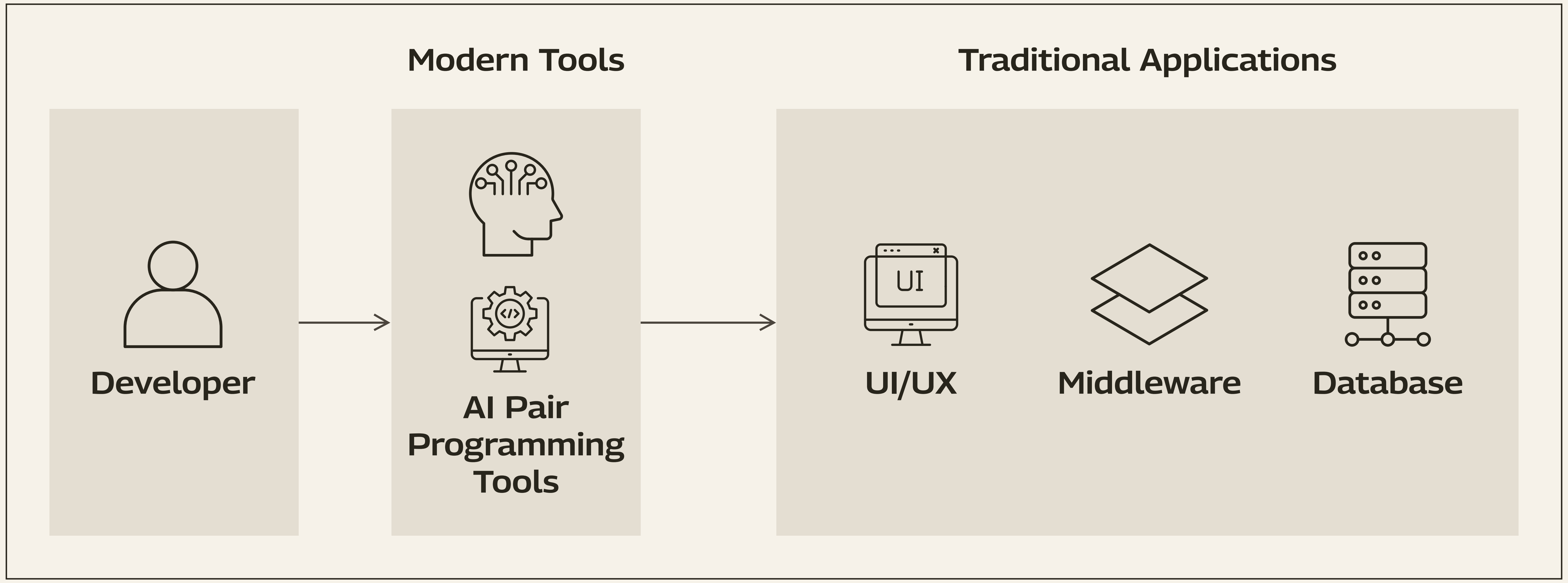

Figure 1: Developers Use AI Pair-Programming Tools to Build Traditional Applications

These dynamics give birth to several structural challenges that limit the industry's ability to realize the promise of AI in software development fully:

- Traditional application architecture remains unchanged even as AI accelerates development processes
- Developers generate significant volumes of code, increasing long-term maintenance burdens for support teams
- AI enhances development workflows but does not influence application behavior or adaptability once deployed
- The growing complexity of multi-agent coding tools results in more code to govern, debug, and support
- It is estimated that about 40% of AI generated code is unwanted code or "AI Slop" but still becomes part of the codebase
- Static UIs, forms, and middleware application programming interface (APIs) still require explicit coding, limiting dynamism and flexibility

To understand why these challenges persist, it is helpful to examine how agentic AI is being used in the market today—and where its limitations lie.

What the Market Offers Today

Currently, the agentic AI solutions available in the market primarily focus on analytics. These platforms rely on agents operating within their existing data environments to retrieve information, generate insights, and automate analytic workflows.

Snowflake Cortex AI is one such example. It provides LLM capabilities within the Snowflake Data Cloud, allowing users to analyze structured and unstructured data using natural language across SQL and Python. While powerful for analytics, Cortex does not generate application UI, workflows, or backend logic at runtime.

As a result, current offerings demonstrate how agentic AI can support analytic tasks, but do not address the architectural challenges of enterprise application development.

A New Approach to Application Development

To break this cycle, we propose a fundamental shift in how software applications are developed. We refer to this concept as Dynamic Agentic Generated Applications (DAGA), in which agentic intelligence generates UI, logic, and integrations at runtime. This approach shifts the focus from maintaining massive static codebases to designing AI agents that generate application behavior in real time.

Developers define minimal agentic logic and constraints, while the agents handle UI generation, data access, and interaction logic at runtime. Consequently, the code is generated directly in production and rendered in the browser, enabling applications to adapt in real time.

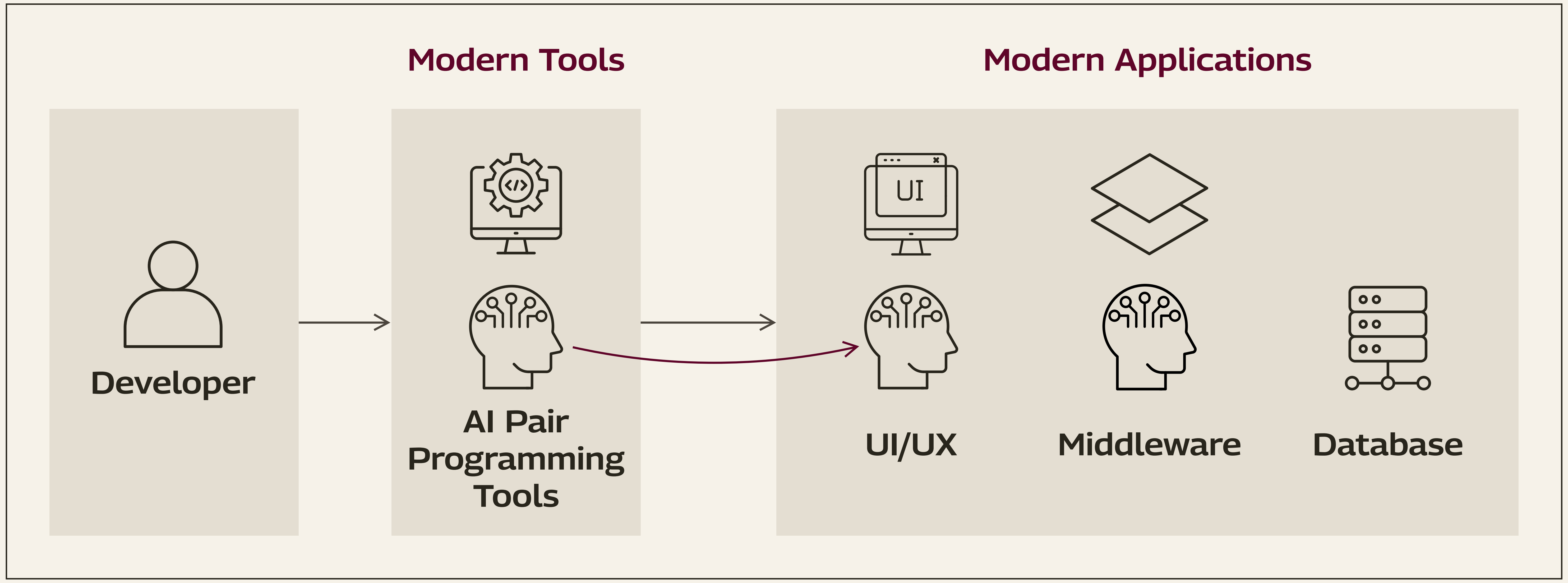

Figure 2: Shifting the code generation responsibility from agents in the "IDE tools" to the agents in the "Application"

This concept introduces several shifts in how applications are built:

Runtime Code Generation

Agents produce UI elements, validate inputs, and render the output directly in the browser based on user prompts

Minimal Static Code

Instead of coding each UI component or form, they define high-level instructions and constraints that guide agent behavior

On-Demand Interactions

Developers write lightweight agentic instructions while agents generate API or Model Context Protocol (MCP) calls dynamically to retrieve backend data, removing the burden of maintaining static middleware libraries

This shift drastically curtails code complexity and sprawl and lowers maintenance effort, resulting in a leaner system that evolves organically as business needs change.

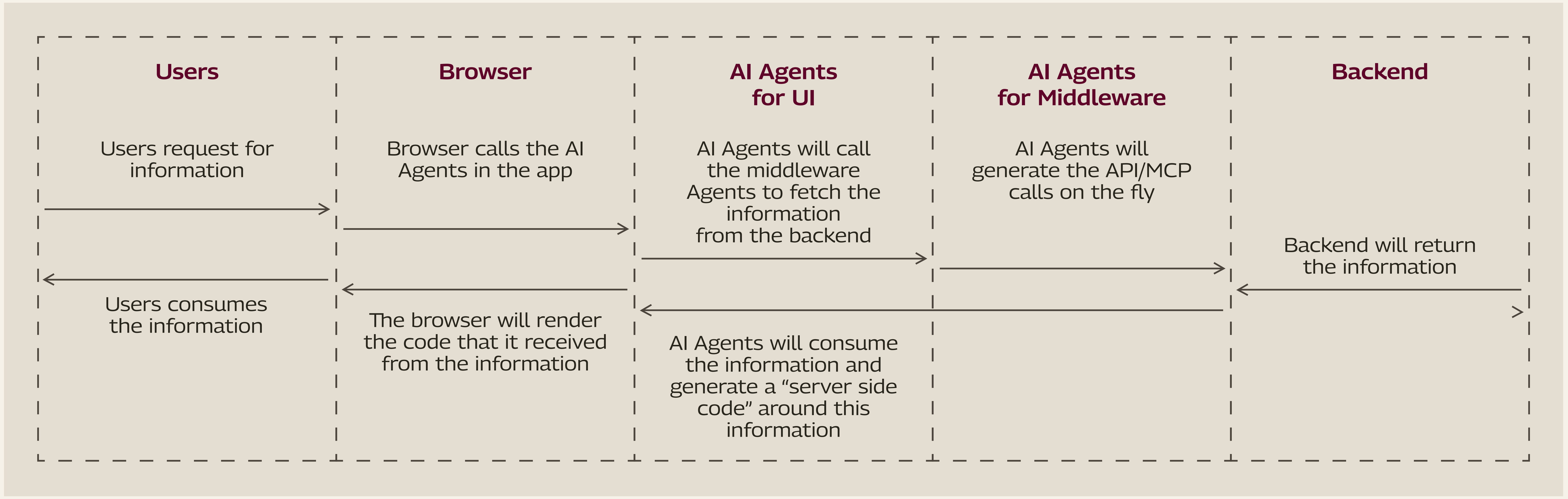
Users
Browser
AI Agents for UI
AI Agents for Middleware
Backend
Users request for information
Browser calls the AI Agents in the app
AI Agents will call the middleware Agents to fetch the information from the backend
AI Agents will generate the API/MCP calls on the fly
Backend will return the information
Users consumes the information
The browser will render the code that it received from the information
AI Agents will consume the information and generate a "server side code" around this information

Figure 3: A sequence Diagram Showing How The Process Will Work End-to-End

What Changes When Applications Become Agentic

In a DAGA-based concept, this shift allows teams to move beyond rigid architectures and focus on outcomes rather than implementation details.

The following highlights the key improvements enabled through agentic code generation:

Reduced Code Volume

Runtime "transient" code generation minimizes the need for large static codebases, drastically reducing lines of code to create and manage.

Lower Maintenance

Smaller codebases translate into lower maintenance overhead and fewer long-term dependencies for application support teams.

Dynamic and Adaptive Interfaces

Agents generate UI elements dynamically, allowing data to be presented in multiple formats like tables, graphs, or summaries, without explicit coding.

Simplified Form Creation

Complex web forms don’t need manual coding as agents generate and render form elements directly in the browser as needed.

Personalized User Experiences

Through memory and state management, agents can maintain user preferences without developers coding for the personalization aspects.

More Consistent Output

Prompting techniques such as few-shot learning, in-context learning, and chain-of-thought enable agents to produce more predictable, consistent outputs.

Key Risks and Design Considerations

As with any agentic AI application, there are inherent risks to this approach that must be carefully addressed in the design.

Type of Risk	Description	Mitigation Strategies
Agentic AI Security Risks	Agentic AI applications are vulnerable to prompt injection or prompt poisoning attacks. Inputs are manipulated, leading the backend to generate code containing sensitive data	• Use hyperlinks with predefined prompts instead of free-form chat to reduce exposure to untrusted inputs. and limit opportunities for attackers to influence agent behavior
Tools Misuse and Privilege Compromise	Attackers can misuse agents' tools to perform unauthorized actions against backend systems	• Use MCP-based tools to access backend systems. • Implement a protected MCP server with authentication and authorization controls to prevent misuse
Token Costs Overrun	High-frequency application usage may lead to excessive token consumption, creating unsustainable operational costs	• Design the agents with all strict control parameters to prevent unnecessary output generation. • Avoid automation for static or infrequently changing information to reduce token usage
Cascading Hallucination	Agents may generate plausible but incorrect information, and errors can compound when these outputs are used in subsequent steps. This risk aligns with issues highlighted in the OWASP Top 10 for LLM Applications	• Apply output validation, implement behavioral constraints, deploy multi-source validation, and ensure ongoing system corrections through feedback loops • Undergo secondary validation before AI-generated knowledge influences critical decisions, thereby helping manage risk as human oversight scales

What is needed to make this happen?

Prompt Templates

Defining prompt templates to capture various functional requirements of an application define the boundaries for the agents to operate within.

Adopting Context Engineering

This is an emerging discipline. Context engineering dynamically shapes the information AI uses to improve accuracy and relevance. It goes beyond static prompts by managing real-time, domain-specific context. This reduces hallucinations and enables personalized, trustworthy outputs. It's key for building adaptive, agentic AI applications in enterprises.

A DAGA Agentic Framework

The agentic frameworks available today are multi-purpose and capable of supporting the topic of discussion. However, there is need to develop an agentic framework exclusively for DAGA with focused capabilities.Integrations: Like traditional integrations, the integrating applications should expose their endpoints to MCPs or APIs or other methods, while ensuring proper security via authentication and trustable authorization.

Conclusion

Based on current research, this approach has not yet been widely implemented in enterprise application development. While data platforms such as Snowflake Cortex AI have laid the necessary groundwork by applying agentic AI within analytics environments, they stop short of enabling full-fledged business applications.

This opens the door to applying DAGA across enterprise domains, such as banking and human resources, where agentic capabilities extend beyond analytics to encompass application behavior and user interaction.

Adoption, however, must be deliberate. Enterprises should begin with low-risk use cases and hybrid implementations that combine agentic dynamic information with non-agentic static components. This measured approach allows teams to build confidence and address security concerns, while creating more agentic models.

As the concept matures, applications can evolve from static constructs to adaptive systems that grow with business needs.

About the Author

Imran Pachapuri

Principal Solution Architect
Tech Mahindra

Imran Pachapuri is a seasoned IT professional with more than two decades of industry experience. Imran has experience delivering technology-driven solutions for global enterprises across a wide range of industries. His expertise is defined by a blend of deep technical expertise, strategic leadership and a commitment to drive meaningful business outcomes.

About Tech Mahindra

Tech Mahindra (NSE: TECHM) offers technology consulting and digital solutions to global enterprises across industries, enabling transformative scale at unparalleled speed. With 152,000+ professionals across 90+ countries helping 1100+ clients, Tech Mahindra provides a full spectrum of services including consult-ing, information technology, enterprise applications, business process services, engineering services, network services, customer experience & design, AI & analytics, and cloud & infrastructure services. It is the first Indian company in the world to have been awarded the Sustainable Markets Initiative's Terra Carta Seal, which recognises global companies that are actively leading the charge to create a climate and nature-positive future. Tech Mahindra is part of the Mahindra Group, founded in 1945, one of the largest and most admired multinational federation of companies. For more information on how TechM can partner with you to meet your Scale at Speed™ imperatives, please visit https://www.techmahindra.com/.

TECH mahindra

www.techmahindra.com
www.linkedin.com/company/tech-mahindra
www.twitter.com/tech_mahindra