Transforming Legacy Monolithic Applications into Microservices with AI Driven Tool

Introduction:

In the fast-evolving software development landscape, legacy monolithic applications often pose significant challenges in terms of scalability, agility, and maintainability. To address these challenges, we embarked on a project to develop an innovative AI-driven tool capable of parsing monolithic applications and seamlessly transforming them into microservices architecture. This case study delves into the journey of conceptualizing, developing, and implementing this transformative product.

Client Overview:

We, a leading software development firm, recognized the pressing need to enhance the performance of migration and maintenance processes for legacy applications. With a vision to streamline software modernization efforts, we initiated the development of a product aimed at automating the conversion of monolithic applications into microservices.

Project Overview:

The project's core objective was to create an AI-powered tool capable of parsing monolithic application code and restructuring it into a microservices architecture. Phase 1 focused on parsing the code from configured paths, analysing interdependencies between modules, and segregating them into separate code bases.

Challenges Faced:

The project's core objective was to create an AI-powered tool capable of parsing monolithic application code and restructuring it into a microservices architecture. Phase 1 focused on parsing the code from configured paths, analysing interdependencies between modules, and segregating them into separate code bases.

Proposed Solution:

To address these challenges systematically, the project team devised a multi-stage approach. Each stage involved supporting a specific programming language, architecture, or design pattern in the code parsing mechanism. Leveraging a combination of open-source AI models and proprietary ML algorithms, the team aimed to achieve comprehensive code parsing capabilities.

Technological Stack:

The entire application was developed using Java Spring Boot for the backend and Angular for the front end. The project utilized a variety of open-source AI models for code parsing, supplemented by custom-built ML models, including neural networks. Notably, the decision to adopt Neo4j as the database was instrumental in efficiently storing and querying the complex network of parsed code modules and their interdependencies.

Key Features and Functionality:

The AI tool offered a user-friendly interface, allowing developers to load the source code path for analysis within a click. Upon execution, the tool meticulously parsed through the code, identified distinct modules, and established interlinks between them.

Furthermore, it classified files such as DAOs, DTOs, service classes, and utilities, facilitating a granular understanding of the application structure.

Visualization and Data Insights:

Utilizing Neo4j's graph database capabilities, the tool provided visually rich representations of code interdependencies, empowering developers to gain insights into module relationships. The ability to visualize past projects' structures facilitated knowledge sharing and informed decision-making during software modernization efforts.

Key Outcomes and Impact:

The key outcomes of the project, aimed at converting monolithic applications into microservices, reflect the efficacy and significance of the developed AI tool in addressing complex challenges within the software development landscape. Here's an in-depth elaboration on each key outcome:

1. Parsing Diverse Application Architectures:

The first significant outcome is the tool's capability to parse applications built using various technologies and architectural styles. It parses applications developed with EJB (Enterprise JavaBeans), Servlet, and Spring frameworks. This versatility demonstrates the tool's adaptability to different legacy systems, regardless of the technologies they were built with. The tool proves its utility across a broad spectrum of legacy applications by accommodating diverse architectures.

2. Support for MVC-based Applications in C#:

Another noteworthy achievement is the tool's ability to parse Model-View-Controller (MVC)- based applications written in C#. MVC is a widely used architectural pattern in software development, particularly in the Microsoft ecosystem. By extending its support to C# MVC applications, the tool ensures that a broader range of legacy systems can benefit from migrating to microservices architecture. This capability is crucial for organizations with heterogeneous technology stacks seeking to modernize their software infrastructure.

3. Significant Reduction in Migration Timeline:

One of the project's most compelling outcomes is the substantial reduction in the timeline required for migrating monolithic applications to microservices architecture. The tool's efficiency and automation contribute to a reduction of approximately 60% in the migration timeline. This dramatic improvement in efficiency translates to significant cost savings and faster time-to-market for organizations undergoing digital transformation initiatives. By streamlining the migration process, the tool enables companies to swiftly modernize their applications without sacrificing quality or stability.

4. Enhanced Visualization of Interdependencies:

Another key outcome is the tool's visualization capabilities, which enable users to comprehend the interlinkages between different modules and submodules within the existing application easily. By visualizing the complex dependencies between various components, the tool empowers developers and architects to make

informed decisions during migration. Identifying modules most impacted by changes becomes more straightforward, facilitating risk assessment and ensuring smoother transitions to the microservices architecture. This enhanced visibility into the application's structure enhances overall project management and mitigates potential risks associated with migration.

5. Seamless Integration with Neo4j Database:

Leveraging Neo4j as the underlying database for storing parsed information is a strategic decision that further enhances the tool's functionality. Neo4j's graph database model aligns well with the inherently interconnected nature of microservices architectures, making it an ideal choice for storing and querying complex interdependencies. By utilizing Neo4j, the tool ensures efficient storage and retrieval of parsed data and enables advanced querying and analysis capabilities. This integration enhances the tool's scalability and performance, efficiently supporting the management of large-scale migration projects.

Conclusion:

In conclusion, the development and deployment of the AI-driven tool marked a significant milestone in streamlining the modernization of legacy monolithic applications. By leveraging advanced AI and ML techniques, coupled with a robust technological stack, the project team successfully addressed the complex challenges associated with code parsing and migration. The transformative impact of the tool resonates across the software development ecosystem, driving efficiency, scalability, and agility in the pursuit of digital innovation.