|  |  |
| --- | --- |
| **NAME OF SESSION** | Modern Software Development |
| **STUDENT NAME** | Funkeyi Jessica Omoro |

|  |  |
| --- | --- |
| **ASSESSMENT FOCUS** | Modernizing Legacy Applications |

|  |
| --- |
| **Scenario:**  You are a software developer working for a medium-sized company with a legacy application that is critical to its operations. The application is becoming increasingly difficult to maintain and update due to its monolithic architecture, outdated technology stack, and lack of documentation. The company is considering modernizing the application to improve its agility, scalability, and maintainability.  **Task:**  Based on the information provided in the "Modern Software Development" lesson and your understanding of the principles discussed, identify and justify **two** key modern methods of Software Development that you would recommend for this project.  For **each** strategy, provide:   * A brief description of the method. * A justification for why this method is suitable for modernizing the legacy application, drawing on the principles discussed in the training document. * An explanation of the potential benefits and challenges associated with implementing this method.   **Deliverables:**   * A short written response (approximately 500 words) outlining the two modernization strategies and your justifications. |
| YOUR ANSWER:  **Legacy Application Modernization Strategies**  **Strategy 1: Microservices Architecture**  Description  Microservices architecture involves decomposing monolithic application into smaller, independent services that communicate through well-defined APIs. Each microservice is responsible for a specific business capability and can be developed, deployed, and scaled independently.  Justification  This approach directly addresses the core issues with your legacy monolith. The lack of documentation and difficulty in maintenance stem from tight coupling and complex interdependencies within the monolithic structure. By breaking the application into discrete, focused services, you create clear boundaries and responsibilities that are easier to understand and modify.  The outdated technology stack becomes less problematic when you can modernize services incrementally. Rather than requiring a complete system overhaul, microservices enable a gradual strangler fig pattern where new services are built with modern technologies while legacy components continue operating until they can be replaced.  Benefits  Enhanced Maintainability: Smaller codebases are easier to understand, test, and modify  Technology Flexibility: Each service can use the most appropriate technology stack  Independent Deployment: Teams can release updates without coordinating across the entire application  Improved Scalability: Individual services can be scaled based on specific demand patterns  Fault Isolation: Issues in one service don't necessarily impact the entire system  Challenges  Increased Complexity: Distributed systems introduce network communication, service discovery, and data consistency challenges  Operational Overhead: Requires robust monitoring, logging, and deployment pipelines  Team Coordination: Requires clear service boundaries and API contracts between teams  Initial Performance Impact: Network calls between services can introduce latency  **Strategy 2: DevOps and Continuous Integration/Continuous Deployment (CI/CD)**  **Description**  DevOps practices emphasize collaboration between development and operations teams, supported by automated CI/CD pipelines that enable frequent, reliable deployments. This includes automated testing, infrastructure as code, and continuous monitoring.  Justification  The difficulty in updating your legacy application likely stems from manual, error-prone deployment processes and fear of breaking existing functionality. CI/CD addresses these concerns by automating testing and deployment, making updates safer and more predictable.  Given the lack of documentation, implementing comprehensive automated testing becomes crucial for understanding system behaviors and preventing regressions. The CI/CD pipeline serves as living documentation of the deployment process while ensuring that changes are thoroughly validated before reaching production.  Benefits  Reduced Risk: Automated testing catches issues early in the development cycle  Faster Time-to-Market: Automated deployments enable more frequent releases  Improved Quality: Consistent testing and deployment processes reduce human error  Better Collaboration: Shared responsibility between development and operations teams  Rollback Capabilities: Automated deployment pipelines enable quick recovery from issues  Documentation Through Code: Infrastructure as code serves as executable documentation  **Challenges**  Cultural Shift: Requires organizational change and buy-in from both development and operations teams  Initial Investment: Setting up comprehensive testing suites and automation infrastructure requires significant upfront effort  Skill Development: Team members need training in new tools and practices  Legacy Integration: Existing systems may not easily integrate with modern CI/CD tools  Implementation Approach  These two strategies work synergistically. Start by implementing CI/CD practices around your existing monolith to establish reliable deployment processes and comprehensive testing. This foundation makes the transition to microservices safer and more manageable, as each extracted service can leverage the established pipeline infrastructure.The combination addresses all major pain points: microservices solve architectural issues while DevOps practices ensure reliable, maintainable operations throughout the modernization journey. |