**1. General Transition Scope**

* What is the scope of the transition?
* What are the timelines and milestones for the transition process?
* Who are the key stakeholders involved in this transition?
* Are there any documents or handbooks available for reference?
* What is the communication plan during the transition?

**2. Knowledge Transfer**

* What are the core processes, workflows, and dependencies I need to know about?
* What documentation is available for the current systems, applications, and infrastructure?
* Are there any architectural diagrams or flowcharts for the systems?
* What are the key challenges or pain points in the existing setup?
* Are there specific tools, frameworks, or technologies I need to get familiar with?

**3. Application/System Details**

* What are the key applications or systems in use?
* What is the application stack (e.g., languages, frameworks, and libraries)?
* Are there any third-party integrations or APIs?
* What are the deployment processes (e.g., CI/CD pipelines)?
* What monitoring tools are in place, and how are alerts configured?
* Are there any known bugs, limitations, or technical debt?

**4. Roles and Responsibilities**

* Who are the key team members and their roles?
* What are my primary responsibilities during and after the transition?
* Are there any backup resources or escalation points in case of issues?
* Who owns the different components (e.g., code, databases, servers)?

**5. Support & Operations**

* What is the incident management process (e.g., ticketing system)?
* What is the current SLA (Service Level Agreement) for support?
* Are there any recurring incidents or known issues?
* What is the escalation process for critical issues?
* Are there runbooks or playbooks for resolving common issues?
* What tools are used for logging, monitoring, and troubleshooting?

**6. Security & Compliance**

* What are the security protocols and policies in place?
* Are there any compliance standards or certifications to adhere to?
* What is the user authentication and authorization mechanism?
* How are backups managed, and what is the disaster recovery plan?

**7. Database/Infrastructure**

* What are the database technologies and versions in use?
* How are the databases managed and backed up?
* What is the server and infrastructure setup (on-premise, cloud, hybrid)?
* Are there scaling or performance bottlenecks in the current infrastructure?
* What is the maintenance schedule for infrastructure and databases?

**8. Project/Release Management**

* What is the release schedule or process?
* How are changes and updates tracked and managed (e.g., Git, version control)?
* Are there any pending tasks or incomplete projects that I need to take over?

**9. Client/Stakeholder Expectations**

* Who are the primary clients or end-users?
* What are the expectations from stakeholders during the transition?
* Are there regular review meetings or reporting requirements?

**10. Metrics & Performance**

* What are the current performance benchmarks or KPIs?
* How is performance tracked and reported?
* Are there any ongoing performance optimization initiatives?

**11. Historical Context**

* Are there any historical issues or lessons learned that I should know about?
* Have there been any major incidents or outages in the past? How were they resolved?
* What was the primary reason for initiating this transition?

**12. Future Plans**

* Are there any planned upgrades, migrations, or deprecations?
* What are the upcoming projects or initiatives I should prepare for?
* Are there any skillsets I need to develop for future requirements?

**13. Risks and Challenges**

* What are the major risks during this transition process?
* What challenges should I be prepared for post-transition?
* How is the risk mitigation plan defined?

**14. Training and Support**

* Will I receive any formal training or onboarding?
* Who should I approach for clarifications during the transition?
* Are there any hands-on sessions or shadowing opportunities?

**1. Application Overview**

* What is the purpose of the application? Who are the end-users?
* What are the key functionalities and features of the application?
* Are there any high-level architectural diagrams or flowcharts?
* What are the application dependencies (e.g., services, APIs, databases)?
* Are there any third-party tools, libraries, or integrations used?

**2. Codebase Details**

* Where is the source code stored (e.g., GitHub, Azure DevOps)?
* Are there version control and branching strategies documented?
* Are there any coding standards or guidelines followed?
* What is the structure of the codebase, and are there any modules or layers?
* Are there commented sections or parts of the code that need special attention?

**3. Infrastructure & Environment**

* What environments exist (e.g., dev, test, staging, production)?
* What is the hosting infrastructure (e.g., on-premises, cloud providers)?
* What are the server configurations (OS, hardware specs, VM details)?
* Are there any environment-specific configurations?
* How are backups managed, and what is the disaster recovery plan?

**4. Deployment Process**

* What is the deployment strategy (e.g., CI/CD pipelines, manual)?
* What tools are used for deployment (e.g., Jenkins, Azure DevOps)?
* Are there rollback mechanisms or scripts in case of deployment failures?
* Are there any post-deployment tasks or validations?

**5. Monitoring and Logging**

* What tools are used for monitoring (e.g., Dynatrace, App Insights)?
* What metrics are being tracked, and what thresholds are configured?
* Are there any dashboards or reports set up for performance or health checks?
* How are logs generated, stored, and accessed?
* Are there alerts configured for specific thresholds or incidents?

**6. Incident Management**

* What is the process for handling incidents and outages?
* Are there runbooks or playbooks for common issues?
* What is the ticketing tool (e.g., ServiceNow, JIRA)?
* Are there any recurring issues or incidents? How were they resolved?
* What is the escalation matrix for critical issues?

**7. Security**

* What are the authentication and authorization mechanisms in use?
* Are there any role-based access controls (RBAC) configured?
* How is data encrypted (in transit and at rest)?
* Are there specific security policies or standards to adhere to?
* Are there known security vulnerabilities or risks?

**8. Database Details**

* What databases are in use (e.g., SQL Server, MongoDB)?
* What is the database structure, and are there any key schemas to review?
* How are database backups and restores managed?
* Are there any data replication or synchronization mechanisms?
* Are there known performance bottlenecks or optimizations pending?

**9. SLA and Support Details**

* What are the Service Level Agreements (SLAs) for the application?
* What is the expected response and resolution time for incidents?
* Are there any maintenance windows scheduled for the application?
* Who are the key business users or stakeholders to coordinate with?

**10. Documentation**

* Is there a centralized repository for all documentation (e.g., Confluence)?
* What documentation is available (e.g., user manuals, API docs, troubleshooting guides)?
* Are there any gaps in documentation that need to be addressed?

**11. Known Issues and Limitations**

* Are there any ongoing issues that the new team should prioritize?
* What are the application’s limitations or areas that need improvement?
* Are there any pending bugs, tickets, or feature requests?

**12. Transition and Shadowing**

* Will there be a shadowing period for the new team?
* What is the plan for knowledge transfer sessions?
* Who can the new team contact for follow-ups or escalations?
* What are the timelines for the complete transition?

**13. Future Roadmap**

* Are there any planned upgrades, migrations, or deprecations?
* What are the upcoming releases or new features?
* Are there any upcoming projects the new team should be aware of?

**14. Key Contacts and Stakeholders**

* Who are the application owners and business stakeholders?
* Who are the technical points of contact for escalations?
* Are there any third-party vendor contacts involved?

**15. Best Practices and Lessons Learned**

* Are there any best practices for managing or maintaining the application?
* What are the key lessons learned from managing this application in the past?
* Are there any specific dos and don'ts the new team should follow?

**16. Testing and Quality Assurance**

* What is the testing strategy (unit, integration, functional, performance)?
* Are there automated test cases, and where are they stored?
* What tools are used for testing (e.g., Selenium, JUnit)?
* Are there any test data or environments available for testing?
* Are there known areas in the application with low test coverage?

**17. Application Performance**

* Are there any performance benchmarks for the application?
* What tools are used to monitor performance (e.g., load testing tools)?
* Are there known performance bottlenecks or optimization opportunities?
* How is scaling handled (horizontal/vertical scaling, auto-scaling)?
* Are there thresholds set for CPU, memory, or other resources?

**18. Configuration Management**

* Where are the configuration files stored (e.g., appsettings.json, environment files)?
* How are environment-specific configurations managed?
* Are there secrets or sensitive information stored, and how are they secured (e.g., Azure Key Vault)?
* What is the process for updating configurations across environments?

**19. Integration and Interfacing Systems**

* What are the upstream and downstream systems the application interacts with?
* Are there specific APIs, protocols, or data formats used (e.g., REST, SOAP)?
* How are external integrations authenticated (OAuth, API keys)?
* Are there SLAs for the external systems or APIs?
* What happens if an integration fails (e.g., retries, fallback mechanisms)?

**20. Job Scheduling and Background Processes**

* Are there any scheduled jobs or background tasks the application relies on?
* What tools or frameworks are used for scheduling (e.g., Hangfire, Quartz)?
* How are failed jobs handled, and where can logs be found?
* Are there dependencies between jobs or tasks that need attention?
* What is the frequency of critical jobs, and are they monitored?

**21. Change Management**

* What is the process for managing and deploying changes?
* Is there a change approval board (CAB) or other governance mechanisms?
* Are there audit logs for changes made in the application?
* What is the rollback strategy for failed changes or updates?
* Are there any pending change requests that need follow-up?

**22. Licensing and Compliance**

* Are there any licenses or subscriptions for the application or its dependencies?
* Are there compliance requirements (e.g., GDPR, HIPAA)?
* Are there periodic audits or reviews for compliance?
* Are there any pending actions from previous compliance assessments?

**23. End-User Support**

* How do end-users report issues or request enhancements?
* Is there a helpdesk or support portal for end-users?
* Are there user training materials, FAQs, or self-help resources?
* What is the feedback loop for user-reported issues?

**24. Build and Release Process**

* What tools are used for building the application (e.g., MSBuild, Maven)?
* Are there multiple pipelines for different environments?
* Are there any build scripts or custom steps in the build process?
* Are there common build failures or errors that need attention?
* How are artifacts generated, stored, and versioned?

**25. Business Continuity and Disaster Recovery**

* What are the recovery time objectives (RTO) and recovery point objectives (RPO)?
* How is failover handled in case of outages?
* Are there redundant systems or high-availability setups?
* What is the process for restoring services after a disaster?
* Are there periodic disaster recovery drills conducted?

**26. Automation**

* Are there any automation scripts or tools used (e.g., PowerShell, Python)?
* Are there repetitive tasks automated to reduce manual effort?
* Are there areas identified for future automation opportunities?

**27. Data Management**

* How is data archived, and what is the retention policy?
* Are there specific reports or dashboards that need to be generated?
* Are there ETL (Extract, Transform, Load) processes involved?
* Are there any compliance requirements for handling user data?

**28. Transition Risks**

* What are the key risks identified during the transition process?
* What are the mitigation plans for these risks?
* Are there critical handover tasks that need prioritization?

**29. Training for the New Team**

* Are there training sessions planned for the new team?
* Will there be hands-on sessions or practical exercises?
* Are there shadowing opportunities to observe the application in action?
* What is the schedule for follow-up sessions after the initial handover?

**30. Knowledge Gaps**

* Are there areas of the application that lack sufficient documentation?
* Are there parts of the application that only specific individuals understand?
* How will the transition address these knowledge gaps?

**31. Pending Actions and Follow-Ups**

* Are there any ongoing projects or initiatives related to the application?
* What are the high-priority tasks that need immediate attention post-handover?
* Are there any unresolved dependencies or issues?

**32. Communication Plan Post-Handover**

* Who will be the primary contact points for the application?
* What is the process for escalating issues after the transition?
* Will there be periodic review meetings or status updates?

**33. Lessons Learned**

* What went well during the current ownership period?
* What challenges were faced, and how were they resolved?
* Are there any best practices or recommendations for the new team?

**1. Application Overview (Expanded)**

* **Purpose and Business Impact**:  
  Provide a detailed explanation of the application's role within the organization. What business problems does it solve? Which business units or teams rely on it most heavily?  
  *Example*: "This is an order management system used by the sales and inventory teams to track orders and stock levels in real time."
* **Key Features**:  
  Break down major features or modules and their functionalities.  
  *Example*: "Module A handles user registrations, Module B manages order placement, and Module C generates financial reports."
* **Critical Dependencies**:  
  Outline any external or internal systems or data sources the application depends on.  
  *Example*: "The application consumes customer data from System X and sends sales data to API Y."

**2. Infrastructure & Environment (Expanded)**

* **Environment Details**:
  + List all environments (Development, QA, Staging, Production).
  + For each environment, include server IPs, access credentials, and any notable differences (e.g., database size, feature toggles).
  + Provide diagrams if there are multiple load-balanced servers or complex setups.
* **Hosting Details**:
  + Are you using a cloud service (Azure, AWS)? If so, specify the regions and resource groups.
  + Are there specific VM types, scaling rules, or storage configurations?
* **Disaster Recovery (DR)**:  
  Provide a clear overview of the DR strategy. Include:
  + Backup frequency (hourly, daily, etc.)
  + Locations of backups (on-prem, cloud storage)
  + Steps for DR activation, including responsible contacts.

**3. Deployment Process (Expanded)**

* **Tools and Pipelines**:  
  Detail CI/CD pipelines used for deploying the application, including:
  + Tools (Jenkins, Azure DevOps, GitHub Actions)
  + Pipeline stages (build, test, deploy, approvals)
  + Deployment success criteria (e.g., tests passing with 95% coverage).
* **Manual Deployment Instructions**:  
  If automation fails, provide step-by-step instructions for manual deployment. Include:
  + Required tools or scripts
  + Dependencies to check before deploying (e.g., database migrations)
* **Rollback Plan**:  
  Outline the steps to revert to a stable version if deployment fails:
  + Snapshot restoration
  + Database rollback scripts
  + Reverting configuration changes.

**4. Monitoring and Logging (Expanded)**

* **Tools**:  
  Provide a list of monitoring and logging tools (e.g., Dynatrace, Splunk, App Insights, Kibana).
* **Alerts**:
  + Which alerts are currently active?
  + What are the thresholds? (e.g., CPU > 80% for 5 minutes)
  + Who gets notified, and how (email, Slack, SMS)?
* **Log Details**:
  + Where are logs stored (file system, centralized system)?
  + What is the retention policy?
  + Provide examples of log format and key messages to look out for when troubleshooting.

**5. Incident Management (Expanded)**

* **Common Incidents**:
  + List the most frequent issues encountered (e.g., API downtime, database overload).
  + Provide root cause analyses of past incidents to help the new team learn from history.
* **Incident Resolution Playbooks**:
  + Attach playbooks for resolving recurring issues.
  + Include screenshots or commands for clarity (e.g., commands to restart services or query logs).
* **Escalation Process**:  
  Clearly define the escalation tiers:
  + Tier 1: Application Support Team
  + Tier 2: Infrastructure Support Team
  + Tier 3: Vendor Support (e.g., Microsoft or AWS).

**6. Testing and QA (Expanded)**

* **Automated Testing**:
  + Provide access to test automation scripts (e.g., Selenium, Cypress).
  + Explain how to execute automated tests and interpret results.
* **Regression Tests**:  
  Share a set of critical regression test cases that must be run before deployments.
* **Performance Testing**:  
  Include any load testing results, tools used (e.g., JMeter), and identified bottlenecks.

**7. Security (Expanded)**

* **Authentication and Authorization**:
  + Explain how users are authenticated (e.g., ASP.NET Core Identity, OAuth).
  + Are there role-based access controls (RBAC)? What are the roles?
* **Vulnerability Management**:
  + Highlight previous penetration tests or vulnerabilities identified.
  + Share remediation steps taken.
* **Sensitive Data**:
  + List sensitive data handled by the application (e.g., PII, financial data).
  + Describe how it is encrypted (e.g., AES-256).

**8. Database Management (Expanded)**

* **Database Relationships**:  
  Include ER diagrams to show key relationships.  
  Example:  
  "The Orders table is linked to the Users table through the UserID foreign key."
* **Maintenance Tasks**:
  + Scheduled maintenance jobs (e.g., index rebuilding, purging old data).
  + Queries/scripts frequently used by the support team.
* **Known Bottlenecks**:  
  Example:  
  "During peak hours, the Orders table query slows due to lack of proper indexing. Planned optimization is pending."

**9. Future Roadmap (Expanded)**

* **Pending Features**:  
  Detail any features that are partially implemented or planned for the future.  
  Example:  
  "A new reporting module is 70% complete and will be deployed in the next release."
* **Upcoming Changes**:  
  Share information about platform migrations, database upgrades, or deprecations.  
  Example:  
  "The system will move from .NET Core 3.1 to .NET 6 by Q2 2025."

**10. Documentation and Training (Expanded)**

* **Documentation Repository**:  
  Provide a single source for all documentation (e.g., Confluence, SharePoint). Ensure it's organized into:
  + High-Level Overviews
  + Technical Details
  + User Manuals
  + Troubleshooting Guides
* **Hands-On Training**:
  + Provide step-by-step sessions (e.g., how to debug a service, how to perform deployments).
  + Assign mock issues for the new team to solve during shadowing.

**11. Knowledge Gaps & Risks (Expanded)**

* **Uncovered Areas**:  
  Highlight areas where documentation or knowledge is sparse.  
  Example:  
  "The legacy module lacks documentation because the original developer is no longer available."
* **High-Risk Components**:
  + Identify components that are fragile or difficult to maintain.
  + Example:  
    "The payment gateway module has a single point of failure and no backup system."

**12. Tools and Utilities**

* **Development Tools**:
  + Which IDEs and frameworks are used (e.g., Visual Studio, Rider, .NET Core, Node.js)?
  + Are there any custom plugins or extensions required?
* **Build Tools**:
  + How is the build configured (e.g., MSBuild, npm, Maven)?
  + Are there any scripts (e.g., batch, PowerShell) that streamline builds or deployments?
* **Utility Scripts**:
  + Are there any pre-built scripts for common tasks (e.g., log cleanup, data exports)?
  + Are these scripts stored in a shared location or repository?

**13. Backup and Restore Procedures**

* **Database Backup**:
  + How frequently are backups taken (hourly, daily, etc.)?
  + What backup tools are used (e.g., SQL Server Agent, Azure Backup)?
  + Where are backups stored (e.g., cloud, on-premises servers)?
* **Application Configuration Backup**:
  + Are application configuration files backed up separately?
  + What’s the process for restoring configurations?
* **Restore Process**:
  + What are the step-by-step instructions for restoring backups?
  + Are there any specific dependencies or considerations (e.g., restoring backups in a specific order)?

**14. User and Role Management**

* **User Base**:
  + How many users does the application typically serve?
  + Are there user segmentation details (e.g., admin, power users, read-only)?
* **User Onboarding**:
  + What is the process for adding or removing users?
  + Are there tools or admin panels for user management?
* **Role-Based Access**:
  + What roles are available in the system, and what permissions do they have?
  + Are there known issues with role management or permission changes?

**15. Scheduled Tasks and CRON Jobs**

* **Task Overview**:
  + What are the key scheduled tasks or CRON jobs?
  + Are these tasks mission-critical (e.g., nightly data sync, reporting)?
* **Execution Details**:
  + Where are these tasks configured (e.g., Task Scheduler, CRON, Hangfire)?
  + How do you check the status of these tasks or debug failures?
* **Failure Handling**:
  + Are there alert mechanisms for task failures?
  + What’s the retry mechanism for failed tasks?

**16. Reporting**

* **Report Types**:
  + What types of reports are generated (e.g., operational, financial, analytical)?
  + Who consumes these reports, and how are they delivered (email, dashboards)?
* **Tools and Integration**:
  + What reporting tools are used (e.g., SSRS, Power BI)?
  + How are data sources configured for reporting?
* **Custom Reports**:
  + How can the team create new or modify existing reports?
  + Are there templates or guidelines for custom reports?

**17. Licensing, Subscriptions, and Renewals**

* **Application Licensing**:
  + Is the application itself licensed? If yes, what is the renewal process?
  + Are there any key dates or dependencies for license renewals?
* **Third-Party Dependencies**:
  + What third-party libraries or APIs require licensing or subscriptions?
  + Are there usage limits or rate limits associated with these services?

**18. Integration Details**

* **APIs and Webhooks**:
  + What are the key APIs used by the application (internal and external)?
  + Are API credentials securely stored, and where (e.g., Key Vault)?
  + Are there webhooks or callbacks configured? What do they trigger?
* **Data Flow**:
  + How does data flow between the application and other systems?
  + Are there any common integration failures? How are they resolved?
* **Legacy Systems**:
  + Are there integrations with older or deprecated systems?
  + What are the risks of these integrations, and is migration planned?

**19. Alerts and Notifications**

* **Critical Alerts**:
  + What are the high-priority alerts configured (e.g., application down, database unresponsive)?
  + Are these alerts actionable, or do some require investigation?
* **Notifications**:
  + What user or admin notifications are generated by the application?
  + Are notifications configured for email, SMS, or other platforms?
* **Monitoring Gaps**:
  + Are there areas where alerts or notifications are missing but needed?

**20. Troubleshooting Scenarios**

* **Recurring Issues**:
  + List recurring problems and their root causes.
  + Provide resolutions for issues like high CPU usage, database timeouts, or failed deployments.
* **Critical Scenarios**:
  + What to do if the application goes down?
  + Steps to resolve urgent database issues (e.g., stuck transactions).
* **Troubleshooting Tools**:
  + Are there specific tools or utilities for debugging (e.g., SQL Profiler, Postman)?
  + How can logs and monitoring dashboards assist with troubleshooting?

**21. Escalation and Support Matrix**

* **Primary Contacts**:
  + List all key contacts for technical, business, and infrastructure-related issues.
* **Support Tiers**:
  + Define who handles what types of issues at Tier 1, Tier 2, and Tier 3.
* **Vendor Escalation**:
  + Are there third-party vendors who provide support for the application?
  + Provide SLA details and escalation paths for vendors.

**22. Pending Work or Known Backlogs**

* **Outstanding Issues**:
  + Provide a list of unresolved issues (bugs, performance enhancements).
  + Note any tickets in the backlog that require immediate attention.
* **In-Progress Features**:
  + Highlight partially developed features.
  + Share timelines or roadblocks for completion.
* **Deferred Work**:
  + What tasks have been intentionally deferred, and why?

**23. Governance and Compliance**

* **Audit Trails**:
  + Is the application configured to log all sensitive activities (e.g., admin actions)?
  + How long are audit logs retained?
* **Regulatory Compliance**:
  + Does the application comply with regulations like GDPR, HIPAA, or PCI DSS?
  + What are the specific compliance checkpoints?

**24. Training Plan for New Team**

* **Initial Training**:
  + What topics must the new team be trained on first?
  + Who will provide training sessions, and in what format (live, recorded)?
* **Shadowing**:
  + How long will the new team shadow the outgoing team?
  + What tasks should they observe to understand workflows?
* **Knowledge Retention**:
  + Are there quizzes, hands-on tasks, or reviews planned to ensure knowledge retention?

**25. Follow-Up and Transition Completion**

* **Post-Handover Support**:
  + Will the outgoing team provide post-handover support? For how long?
  + What is the process for clarifications during the post-handover phase?
* **Final Checklist**:
  + Ensure that all KT sessions are documented and signed off.
  + Confirm that the receiving team has access to all tools, credentials, and resources.
* **Feedback Mechanism**:
  + How will the receiving team provide feedback on the KT process?
  + Are there follow-up sessions planned to address any gaps?

**26. Configuration Management**

* **Centralized Configuration**:
  + Is there a centralized configuration management system (e.g., Azure App Configuration, HashiCorp Vault)?
  + How are secrets and sensitive configurations stored securely?
  + Are there guidelines for adding or modifying configurations?
* **Environment-Specific Configuration**:
  + What are the key differences between environments (Dev, QA, Prod)?
  + Are there separate config files or parameter overrides for each environment?
* **Documentation**:
  + Are there guides for interpreting and modifying configuration settings?
  + What are the dependencies between configurations and runtime behaviors?

**27. Operational Insights**

* **Operational Metrics**:
  + What are the critical KPIs (e.g., latency, uptime, response time)?
  + Where can the team find operational dashboards or logs?
* **Incident History**:
  + What are the most significant outages the application has faced?
  + What actions were taken to resolve these incidents, and were root causes addressed?
* **Recurring Maintenance Activities**:
  + List regular maintenance tasks (e.g., database index rebuilding, cache clearing).
  + Are there automated tools/scripts for performing maintenance?

**28. Onboarding Process**

* **Access Requests**:
  + How can new team members request access to servers, tools, and repositories?
  + Are there specific onboarding documents for new joiners?
* **Walkthroughs**:
  + Are there system walkthroughs available (e.g., architecture overview, common workflows)?
  + Provide a timeline or checklist for onboarding new team members.
* **Credential Transfers**:
  + Ensure a secure process for transferring credentials (e.g., password managers like LastPass, 1Password).
  + Are there guidelines for rotating credentials after onboarding?

**29. Legacy System Dependencies**

* **Older Components**:
  + Are there any legacy modules or systems that the application relies on?
  + Is there a plan to retire or upgrade these components?
* **Support for Legacy Systems**:
  + What level of support or maintenance is required for legacy systems?
  + Are there known challenges or risks associated with these systems?
* **Compatibility Issues**:
  + Are there compatibility constraints with newer systems, tools, or operating environments?

**30. Application Scalability**

* **Scaling Mechanisms**:
  + Is the application horizontally or vertically scalable?
  + Are there auto-scaling rules in place (e.g., based on CPU usage, memory)?
* **Performance Tuning**:
  + Have there been any performance tuning efforts in the past?
  + Are there areas identified for further optimization?
* **Bottlenecks**:
  + What are the known scalability bottlenecks (e.g., database locks, API rate limits)?
  + Are there mitigation strategies in place?

**31. End-User Training**

* **Documentation for Users**:
  + Are there user-facing manuals or quick-start guides?
  + Are training videos or walkthroughs available for key features?
* **Support for User Issues**:
  + What is the process for handling user complaints or feedback?
  + Are there FAQs or self-service tools to help users troubleshoot common issues?

**32. Compliance and Auditing**

* **Log Audits**:
  + Are logs reviewed periodically for anomalies or security breaches?
  + What tools or scripts are used for auditing purposes?
* **Data Retention Policies**:
  + How long is data retained in the system, and what regulations govern retention?
* **Audit Trail Requirements**:
  + Does the application track user activities for auditing purposes?
  + Are audit trails centralized or stored per module?

**33. API Management**

* **Documentation**:
  + Are the APIs documented (e.g., Swagger, Postman collections)?
  + Are there versioning guidelines for maintaining backward compatibility?
* **Access Control**:
  + How is API access managed (e.g., OAuth, API keys)?
  + Are there rate limits or throttling mechanisms in place?
* **Monitoring and Debugging**:
  + How is API usage monitored (e.g., usage statistics, error rates)?
  + Are there tools for debugging API calls (e.g., Fiddler, Postman)?

**34. Middleware and Libraries**

* **Custom Middleware**:
  + Are there any custom middleware components in the application?
  + What are their functions and points of integration?
* **Library Updates**:
  + What libraries or dependencies require frequent updates?
  + Are there any known deprecated libraries that need replacement?
* **Dependency Vulnerabilities**:
  + Have vulnerability scans been conducted for third-party libraries?
  + How are vulnerabilities patched or mitigated?

**35. Debugging and Diagnostics**

* **Debugging Tools**:
  + What tools are used for debugging issues (e.g., Visual Studio Debugger, Chrome DevTools)?
  + Are there specific settings or configurations required for debugging?
* **Diagnostics Steps**:
  + Provide a step-by-step guide for diagnosing common application issues.
  + Include commands, tools, or logs to check during diagnostics.
* **Sandbox Testing**:
  + Are there sandbox environments for safe debugging and experimentation?

**36. External Dependencies**

* **Service Providers**:
  + List all external service providers (e.g., payment gateways, SMS services).
  + Provide contact details and escalation paths for these providers.
* **Contracts and SLAs**:
  + What are the service-level agreements with external vendors?
  + Are there penalties or clauses for service downtime?
* **Dependency Failures**:
  + What happens when external dependencies fail (e.g., retries, fallback mechanisms)?
  + Are there alternative providers configured for critical services?

**37. Disaster Recovery (Detailed Plan)**

* **Failover Systems**:
  + Are there secondary/backup systems in place for failover?
  + How is the switch between primary and backup systems managed?
* **Disaster Scenarios**:
  + Provide steps to follow during specific disaster scenarios (e.g., complete server failure, data corruption).
* **Testing and Validation**:
  + Is the disaster recovery process tested periodically?
  + What metrics are used to measure DR readiness?

**38. DevOps Integration**

* **Pipeline Details**:
  + Provide details of CI/CD pipelines (tools, stages, common failures).
  + How are build artifacts managed and stored?
* **Infrastructure as Code (IaC)**:
  + Are tools like Terraform, ARM templates, or Ansible used to define infrastructure?
  + Where are these configurations stored?
* **Containerization**:
  + If using Docker, provide Dockerfile templates and orchestration details (e.g., Kubernetes, Docker Compose).

**39. Service Observability**

* **Tracing and Metrics**:
  + Is distributed tracing configured (e.g., OpenTelemetry)?
  + What key metrics (latency, throughput) are tracked for observability?
* **Log Correlation**:
  + Can logs be correlated across services (e.g., transaction IDs)?
  + What tools are used for log aggregation (e.g., ELK Stack, Splunk)?

**40. Custom Business Logic**

* **Critical Business Processes**:
  + Are there workflows or processes that are highly customized?
  + What are the risks associated with these customizations?
* **Audit for Custom Logic**:
  + Are there periodic reviews of custom business logic to ensure it meets requirements?

**41. Additional Shadowing Scenarios**

* **Live Support Observation**:
  + Allow the new team to shadow live support scenarios to gain real-world experience.
* **Mock Incidents**:
  + Simulate mock incidents and have the new team resolve them as practice.
  + Evaluate their responses and provide feedback.

**42. Critical Workflows**

* **Workflow Diagrams**:
  + Are there visual diagrams (e.g., flowcharts) that outline how the application processes critical tasks?
  + Can these workflows be broken down step by step for debugging and optimization?
* **Edge Cases**:
  + What are the edge cases for these workflows, and how are they handled?
  + Example: "What happens when an API times out during an order submission workflow?"
* **Workflow Monitoring**:
  + How are these workflows monitored (e.g., tracking order states, process checkpoints)?
  + Are there alerts if a workflow stalls or fails?

**43. Data Migration**

* **Previous Migrations**:
  + Have there been past data migrations? If yes, what tools or strategies were used?
  + Were there challenges during these migrations, and how were they resolved?
* **Future Migrations**:
  + Are there any pending or planned migrations (e.g., database upgrades, schema changes)?
  + Provide a step-by-step migration plan and scripts if applicable.
* **Data Consistency**:
  + How is data consistency maintained during migrations?
  + Are there validation checks in place post-migration?

**44. Configuration Audit**

* **Validation**:
  + Are there automated checks to validate configurations during deployments?
  + Are configuration files versioned for tracking changes?
* **Dynamic Configurations**:
  + Are there configurations that change dynamically during runtime (e.g., feature flags)?
  + How are such changes managed and tested?
* **Common Misconfigurations**:
  + What are the most common configuration issues, and how are they resolved?

**45. End-of-Life (EOL) Plans**

* **Application Deprecation**:
  + Is there an EOL plan for this application or its components?
  + What are the timelines and dependencies for retiring the application?
* **System Replacement**:
  + If this application is being replaced, what is the replacement system?
  + How will the transition from the current system to the new system be managed?
* **Data Archiving**:
  + What is the strategy for archiving historical data when the application is decommissioned?

**46. Dependency Matrix**

* **Direct Dependencies**:
  + What external services, APIs, or libraries does the application rely on?
  + Provide a detailed matrix showing how each dependency integrates with the application.
* **Transitive Dependencies**:
  + Are there any transitive dependencies that could pose risks if they change or fail?
* **Dependency Updates**:
  + How are dependencies updated (e.g., library version upgrades)?
  + Are there known compatibility issues with specific dependency versions?

**47. Resource Utilization**

* **System Resources**:
  + What is the typical resource usage (CPU, memory, disk) during normal and peak loads?
  + Are there resource usage benchmarks?
* **Optimization Efforts**:
  + Has the application undergone resource optimization in the past?
  + Example: "Switched to in-memory caching to reduce database calls."
* **Capacity Planning**:
  + How does the team handle resource scaling as usage grows?
  + Are there tools for forecasting future resource needs?

**48. Application Health Checks**

* **Built-in Health Checks**:
  + Does the application have health check endpoints (e.g., /health)?
  + What do these endpoints monitor (e.g., database connectivity, service availability)?
* **Custom Monitoring Scripts**:
  + Are there custom scripts for checking application health?
  + How often are these scripts run, and where are the results logged?

**49. Multi-Tenancy Details**

* **Tenant Management**:
  + If the application supports multiple tenants, how are tenants managed?
  + Example: "Each tenant has its own database schema but shares the same application instance."
* **Tenant-Specific Configurations**:
  + Are there tenant-specific settings (e.g., branding, localization)?
  + How are these settings configured and stored?
* **Cross-Tenant Issues**:
  + Are there risks of data leaks or misconfigurations between tenants? How are they mitigated?

**50. License Compliance**

* **Software Licenses**:
  + Are all third-party libraries and tools used in the application properly licensed?
  + Are there open-source licenses that require attribution or compliance?
* **Renewal Management**:
  + Who is responsible for tracking license renewals?
  + Are there alerts or reminders for approaching expiration dates?

**51. API Documentation and Contracts**

* **API Versioning**:
  + How are API versions managed?
  + What happens when breaking changes are introduced?
* **Contract Testing**:
  + Are there automated tests to validate API contracts with external systems?
* **API Deprecation**:
  + Are there APIs planned for deprecation? If yes, what is the transition strategy?

**52. Governance and Reporting**

* **Governance Policies**:
  + What governance policies are in place (e.g., approval workflows for changes)?
  + Are there mandatory reviews for specific changes?
* **Reports and Audits**:
  + Are there recurring reports for stakeholders (e.g., application performance, issue trends)?
  + How is compliance reported for governance audits?

**53. Cost Management**

* **Operational Costs**:
  + What are the estimated monthly or annual costs for running the application (e.g., cloud resources, licenses)?
  + Are there areas identified for cost optimization?
* **Budget Oversight**:
  + Who is responsible for managing and approving expenses?
* **Billing Alerts**:
  + Are there alerts for unusual billing spikes (e.g., unexpected cloud usage)?

**54. Localization and Internationalization**

* **Supported Languages**:
  + Which languages are supported by the application?
  + Are there tools used for translation (e.g., Google Translate API)?
* **Localization Files**:
  + Where are localization files stored, and how are they updated?
  + Example: "All localization keys are in a JSON file per language."
* **Date and Currency Formatting**:
  + Does the application handle regional differences in date formats or currency?

**55. Retrospectives and Lessons Learned**

* **Previous Projects**:
  + What lessons were learned from earlier application versions or projects?
  + Example: "Frequent deployments led to downtime due to inadequate testing."
* **Best Practices**:
  + What best practices have been adopted in managing the application?
* **Improvement Areas**:
  + What areas need improvement or further development?

**56. Contingency Planning**

* **Staffing Contingencies**:
  + Are there backup resources in case key team members are unavailable?
  + Is there cross-training between team members to reduce knowledge silos?
* **Unplanned Downtime**:
  + What are the steps to handle unplanned outages or disasters?
  + Are there predefined communication plans during critical downtime?

**57. Performance Metrics**

* **Historical Performance Data**:
  + Is there historical data for key performance metrics?
  + Example: "The average response time during peak hours has improved by 20% after optimization."
* **Key Bottlenecks**:
  + What performance bottlenecks have been identified but not yet resolved?

**58. Transition Closure**

* **Final Review**:
  + Conduct a final review meeting to ensure all aspects of the KT are covered.
* **Checklist Sign-Off**:
  + Use a formal checklist to track completed KT items.
  + Ensure both the outgoing and incoming teams sign off on the transition.
* **Feedback**:
  + Collect feedback from the new team on gaps in the KT process.
  + Use feedback to improve future transitions.

**59. Team Structure and Collaboration**

* **Team Roles**:
  + Who are the key team members responsible for the application (current and future)?
  + Are there dedicated roles for DevOps, QA, business analysis, and user support?
* **Collaboration Tools**:
  + What tools are used for team collaboration (e.g., Slack, Microsoft Teams)?
  + Are there specific communication channels or shared spaces for discussions?
* **Handoff Point of Contact**:
  + Who will remain as a fallback contact post-handover in case clarifications are needed?

**60. Application Dependencies**

* **Internal Services**:
  + What other internal services does the application rely on?
  + Are there service-level agreements (SLAs) or contracts for these dependencies?
* **External APIs**:
  + Are there any APIs provided by third-party vendors?
  + Example: Payment gateways, shipping services, or cloud-based authentication.
* **Dependency Failure Mitigation**:
  + What happens when these dependencies fail? (e.g., retries, cached responses, alerts).
* **Vendor Relationships**:
  + Are there critical vendor relationships tied to this application?
  + Provide contacts and escalation paths for vendor-related issues.

**61. Scalability and Load Management**

* **Load Management**:
  + What is the typical peak load the application handles?
  + Example: "During Black Friday, traffic spikes by 300%."
* **Stress Testing**:
  + When was the last stress test performed?
  + Are there logs or results showing how the application performed under stress?
* **Scaling Architecture**:
  + How is the application scaled during high-traffic periods (manual, auto-scaling)?
  + Are there predefined rules in place for horizontal or vertical scaling?

**62. Infrastructure as Code (IaC)**

* **Configuration Details**:
  + Are there scripts for creating or modifying infrastructure using tools like Terraform, ARM Templates, or AWS CloudFormation?
* **Repository Locations**:
  + Where is the IaC code stored (e.g., GitHub, Bitbucket)?
  + Are there guidelines for modifying and deploying IaC?
* **Automation**:
  + Are there automated pipelines that integrate with the IaC configurations?
  + How often is infrastructure validation done?

**63. Environment-Specific Details**

* **Environment Differences**:
  + Highlight differences in configurations, features, or performance settings across environments (Development, QA, Production).
* **Data Masking**:
  + Is there a data masking policy for sensitive information in non-production environments?
* **Environment Access**:
  + Who has access to each environment, and how is access managed?

**64. CI/CD Pipelines (Continuous Integration/Continuous Deployment)**

* **Pipeline Architecture**:
  + Describe the CI/CD pipeline setup (e.g., stages for build, test, deploy, rollback).
  + What tools are used for CI/CD (e.g., Jenkins, Azure DevOps, GitHub Actions)?
* **Pipeline Failures**:
  + What are the common causes of CI/CD pipeline failures?
  + How can these failures be debugged or mitigated?
* **Deployment Frequencies**:
  + How often are deployments done? Are there release schedules or sprints tied to them?

**65. Logging Frameworks**

* **Log Storage**:
  + Where are logs stored (e.g., Splunk, ELK stack, Azure Monitor)?
  + How long are logs retained?
* **Search Queries**:
  + Provide example queries for searching logs during debugging.
* **Structured Logging**:
  + Are logs structured with consistent formats (e.g., JSON)?
  + Do logs include correlation IDs for tracing distributed workflows?

**66. Audit and Compliance**

* **Regulatory Requirements**:
  + Are there specific regulations the application must comply with (e.g., GDPR, HIPAA)?
  + What tools or processes ensure compliance?
* **Audit Trails**:
  + Is user activity logged for auditing purposes?
  + What is the retention period for these logs?
* **Compliance Documentation**:
  + Provide evidence or reports of past compliance audits.
  + Are there outstanding compliance requirements?

**67. Security and Vulnerability Management**

* **Security Scans**:
  + When was the last vulnerability scan performed?
  + Were any high-priority vulnerabilities identified? If yes, were they resolved?
* **Authentication**:
  + What mechanisms are used for authentication (e.g., OAuth, OpenID Connect)?
  + Are there MFA (Multi-Factor Authentication) policies in place?
* **Penetration Testing**:
  + Has the application undergone penetration testing? If yes, provide reports.

**68. End-User Support**

* **User Feedback Loop**:
  + How do users provide feedback (e.g., surveys, helpdesk tickets)?
  + Are there patterns in recurring user complaints?
* **Support Team Handoff**:
  + Who are the key contacts for user support (if external to the application team)?
  + Are there tools for managing user tickets (e.g., ServiceNow)?

**69. Testing Processes**

* **Manual Testing**:
  + Are there test cases documented for manual testing?
  + What are the priority test scenarios?
* **Test Data Management**:
  + How is test data created, managed, and secured in non-production environments?
* **Regression Testing**:
  + How often are regression tests run, and what tools are used?

**70. Pending Feature Requests**

* **Backlog Features**:
  + List features requested by stakeholders but not yet implemented.
  + Are there defined priority levels for these requests?
* **Approval Process**:
  + What is the process for approving new features?
  + Example: "All features must pass a feasibility analysis by the product owner."

**71. Budget and Cost Optimization**

* **Current Costs**:
  + What are the primary cost drivers (e.g., cloud hosting, licensing)?
  + Are there recent cost breakdown reports?
* **Cost Optimization Efforts**:
  + Have there been efforts to reduce costs, such as server downgrades or unused resource cleanups?
* **Budget Forecasting**:
  + Is there a budget plan for the next 6–12 months?

**72. Training and Onboarding Resources**

* **Training Materials**:
  + Are there pre-recorded sessions, tutorials, or guides for the new team?
  + Provide a schedule for live KT sessions or shadowing opportunities.
* **Access and Tools**:
  + How can the new team gain access to all tools, credentials, and environments?
  + Is there a centralized onboarding checklist?

**73. Transition Risks**

* **Identified Risks**:
  + Are there risks associated with the transition?
  + Example: "Knowledge gaps in legacy components could delay incident resolution."
* **Risk Mitigation**:
  + What measures are in place to reduce transition risks (e.g., extended shadowing period)?

**74. Post-Handover Metrics**

* **Transition Success Metrics**:
  + How will the success of the handover be measured?
  + Examples:
    - Reduction in ticket resolution time.
    - New team meeting performance SLAs.
    - Fewer escalations post-transition.
* **Continuous Support**:
  + Will the outgoing team provide periodic check-ins after the handover?
  + How long will post-handover support last?

**75. Wrap-Up and Handoff Completion**

* **Final Deliverables**:
  + Ensure all deliverables are handed off (e.g., documentation, access credentials, unresolved tickets).
  + Conduct a final walkthrough with the receiving team.
* **Feedback and Closure**:
  + Solicit feedback from the receiving team on gaps in the KT process.
  + Document lessons learned for improving future handovers.

\*\*Comprehensive Application Handover Checklist\*\*

---

## \*\*1. Application Overview\*\*

- \*\*Purpose:\*\* Briefly describe the purpose of the application and its business impact.

- \*\*Key Features:\*\* Highlight the main features or modules of the application.

- \*\*Architecture Diagrams:\*\* Attach or link to diagrams showing the high-level architecture.

- \*\*Dependencies:\*\* List all internal and external dependencies (services, APIs, databases).

---

## \*\*2. Team Structure and Contacts\*\*

- \*\*Current Team Members:\*\* Provide names, roles, and responsibilities of the outgoing team.

- \*\*Post-Handover Contacts:\*\* List fallback contacts for post-handover support.

- \*\*Collaboration Tools:\*\* Share links to communication platforms (e.g., Slack, Teams).

---

## \*\*3. Infrastructure Details\*\*

- \*\*Environment Setup:\*\* Provide details about Development, QA, Staging, and Production environments.

- \*\*Hosting Information:\*\* Cloud provider (e.g., AWS, Azure), regions, VM configurations.

- \*\*Environment Access:\*\* Credentials or instructions for accessing each environment.

- \*\*Scalability:\*\* Auto-scaling configurations or manual scaling steps.

---

## \*\*4. CI/CD Pipelines\*\*

- \*\*Tools Used:\*\* Jenkins, GitHub Actions, Azure DevOps, or others.

- \*\*Pipeline Stages:\*\* Detail stages such as build, test, deploy, and rollback.

- \*\*Common Issues:\*\* Document troubleshooting steps for common pipeline failures.

- \*\*Artifact Storage:\*\* Location of build artifacts (e.g., Nexus, Artifactory).

---

## \*\*5. Application Dependencies\*\*

- \*\*Internal Dependencies:\*\* List other systems or services this application relies on.

- \*\*External Dependencies:\*\* Include APIs, third-party services, and licensing details.

- \*\*Dependency Management:\*\* Steps to update or replace dependencies safely.

---

## \*\*6. Database Details\*\*

- \*\*Database Type:\*\* SQL Server, MongoDB, etc.

- \*\*Schema Details:\*\* Attach schema diagrams or key table descriptions.

- \*\*Backup and Restore:\*\* Instructions for backups and restoration processes.

- \*\*Performance Tuning:\*\* Index optimizations or common query issues.

---

## \*\*7. Security and Compliance\*\*

- \*\*Authentication Mechanisms:\*\* OAuth, SSO, or other methods in use.

- \*\*Vulnerability Management:\*\* Details of recent scans or penetration tests.

- \*\*Compliance Requirements:\*\* GDPR, HIPAA, or other regulatory needs.

- \*\*Audit Trails:\*\* Location of logs or tools used for auditing.

---

## \*\*8. Logging and Monitoring\*\*

- \*\*Log Storage:\*\* Splunk, ELK Stack, Azure Monitor, or other systems used.

- \*\*Critical Alerts:\*\* Describe configured alerts and thresholds.

- \*\*Monitoring Tools:\*\* Dynatrace, App Insights, or others.

- \*\*Sample Queries:\*\* Include common log queries for troubleshooting.

---

## \*\*9. Testing and QA\*\*

- \*\*Testing Tools:\*\* Selenium, JMeter, or others.

- \*\*Regression Test Cases:\*\* Highlight critical test scenarios.

- \*\*Test Data Management:\*\* How test data is created and secured.

- \*\*Automated Testing:\*\* Describe the scope and coverage of automated tests.

---

## \*\*10. Scheduled Tasks and CRON Jobs\*\*

- \*\*List of Jobs:\*\* Name, purpose, and frequency of each scheduled task.

- \*\*Job Monitoring:\*\* Tools or methods used to monitor job statuses.

- \*\*Failure Handling:\*\* Steps to retry or debug failed jobs.

---

## \*\*11. API Documentation\*\*

- \*\*Endpoints:\*\* List all APIs with descriptions and expected parameters.

- \*\*Versioning:\*\* Details of API versioning and deprecation plans.

- \*\*Error Handling:\*\* Common API errors and their resolutions.

- \*\*Authentication:\*\* OAuth tokens, API keys, or other mechanisms.

---

## \*\*12. Pending Work\*\*

- \*\*Outstanding Features:\*\* List features currently under development or in the backlog.

- \*\*Known Issues:\*\* Highlight bugs or performance bottlenecks.

- \*\*Migration Plans:\*\* Ongoing or planned migrations (e.g., database upgrades).

---

## \*\*13. Training and Resources\*\*

- \*\*Onboarding Materials:\*\* User manuals, video tutorials, or walkthroughs.

- \*\*Shadowing Plan:\*\* Schedule for live shadowing of outgoing team members.

- \*\*Access Checklist:\*\* Ensure access to tools, credentials, and environments.

- \*\*Training Sessions:\*\* Schedule and topics for live training.

---

## \*\*14. Documentation Links\*\*

- \*\*Central Repository:\*\* Link to Confluence, SharePoint, or other repositories.

- \*\*Runbooks:\*\* Attach detailed runbooks for handling incidents or common tasks.

- \*\*Troubleshooting Guides:\*\* Include guides for recurring issues.

- \*\*Architecture Documentation:\*\* Provide access to design and technical architecture documents.

---

## \*\*15. Risk and Mitigation Plan\*\*

- \*\*Identified Risks:\*\* List potential risks during and after the transition.

- \*\*Mitigation Steps:\*\* Actions to reduce these risks (e.g., extended support).

- \*\*Fallback Plan:\*\* What to do if critical handover items are missed.

---

## \*\*16. Post-Handover Plan\*\*

- \*\*Support Period:\*\* Duration of post-handover support by the outgoing team.

- \*\*Review Meetings:\*\* Schedule follow-up meetings to address gaps or concerns.

- \*\*Feedback Mechanism:\*\* How the new team can provide feedback on the KT process.

- \*\*Success Metrics:\*\* Define metrics to evaluate the success of the handover.

---

## \*\*17. Closure Checklist\*\*

- [ ] All documentation delivered and verified.

- [ ] Credentials transferred and access confirmed.

- [ ] Training sessions completed.

- [ ] Pending issues acknowledged by the new team.

- [ ] Handover signed off by both outgoing and receiving teams.

---

## \*\*18. Performance Metrics and Benchmarks\*\*

- \*\*Historical Metrics:\*\* Provide historical data on performance metrics (e.g., response time, uptime).

- \*\*Benchmarks:\*\* Document agreed performance benchmarks for normal and peak loads.

- \*\*Monitoring Dashboards:\*\* Share links to dashboards showing real-time performance metrics.

---

## \*\*19. Disaster Recovery (DR) Plan\*\*

- \*\*Backup Strategy:\*\* Detail the frequency and location of backups.

- \*\*Failover Mechanisms:\*\* Describe failover strategies and redundant systems.

- \*\*Disaster Scenarios:\*\* Provide step-by-step guides for handling disasters such as server crashes or data corruption.

- \*\*Testing DR:\*\* Include logs or results from the last disaster recovery test.

---

## \*\*20. Cost Management and Optimization\*\*

- \*\*Current Costs:\*\* Provide a breakdown of operational costs (e.g., cloud resources, licenses).

- \*\*Optimization Opportunities:\*\* Highlight potential areas for cost reduction.

- \*\*Billing Alerts:\*\* Describe any billing alert mechanisms in place to detect unusual spikes.

- \*\*Forecasting:\*\* Share cost projections for the next 6-12 months.

---

## \*\*21. End-User Support and Feedback\*\*

- \*\*Support Channels:\*\* List channels for user support (e.g., helpdesk, email, chatbots).

- \*\*Feedback Mechanisms:\*\* Describe how user feedback is collected and addressed.

- \*\*Common Complaints:\*\* Highlight frequently reported issues and their resolutions.

- \*\*Self-Service Tools:\*\* Document available self-service resources for users.

---

## \*\*22. Governance and Compliance\*\*

- \*\*Governance Policies:\*\* List internal policies governing application changes.

- \*\*Audit Logs:\*\* Provide access to audit logs for recent changes or incidents.

- \*\*Compliance Checks:\*\* Describe periodic compliance review processes.

- \*\*Future Compliance Needs:\*\* Highlight any upcoming regulatory requirements or audits.

---

## \*\*23. Lessons Learned and Recommendations\*\*

- \*\*Challenges Faced:\*\* Document challenges encountered during the application's lifecycle.

- \*\*Best Practices:\*\* Share practices adopted to ensure smooth operations.

- \*\*Future Recommendations:\*\* Provide suggestions for further optimization or improvements.

---