1. **Access to Source Code**
   * **Insights into the source code are generally possible, but the full source code is not provided.**
   * **Which specific components are accessible?**
   * **How is access granted (e.g., remote, sandbox)?**
2. **Application Landscape Analysis**
   * **Is a tool-based analysis of the application landscape possible, including access to source code repositories?**
3. **Scope of Analysis**
   * **The analysis is intended as a compact, pragmatic first step without transitioning into a full conceptual or implementation phase.**
   * **How deep can the analysis go to meet requirements without exceeding the intended scope?**
4. **Stakeholder Interviews**
   * **How many stakeholders are available, and how much time can they realistically contribute?**
5. **Cloud-Ready or Hybrid Operating Models**
   * **Are there existing preferences or restrictions regarding cloud providers or operating models?**
6. **Initial Cost Estimation**
   * **Is a rough estimate expected, or a detailed breakdown by effort and modules?**
7. **Technical Debt Assessment**
   * **Since full source code access is not available, a comprehensive evaluation of technical debt is limited.**
   * **Is this acceptable for the DFB?**
8. **Data Model Documentation**
   * **Is there a documented data model overview or entity-relationship diagrams available for analysis?**
9. **Synchronization & Error Handling**
   * **What mechanisms currently exist for synchronization and error handling?**
   * **Is there monitoring for replication errors?**
10. **API Documentation & Versioning**

* **Is there a central API documentation or an API gateway?**
* **How is versioning and backward compatibility managed?**

1. **CI/CD Pipeline**

* **Is there a unified CI/CD pipeline for legacy and microservices, or are they separate processes?**
* **How is deployment monitoring handled?**

1. **Kubernetes Hosting & Cloud Migration**

* **Kubernetes is hosted on-premises.**
* **Are there plans or restrictions regarding possible migration to the cloud (e.g., Azure, AWS)?**
* **What cloud strategy is being pursued?**

**Business & Functional Understanding**

What are the common pain points in the current system?

Is there any primary users, and what roles do they have?

Can you share the existing architecture diagram to understand module dependencies?

What are the expected improvements in terms of user experience and responsiveness?

What third-party integrations exist, and how critical are they?

How is authentication managed? (RBAC, OAuth, LDAP, etc.)

What existing security protocols and compliance standards must be followed?

Are there any existing security vulnerabilities that must be addressed?

How does data persist—what databases are currently in use?

What API layer exists, and will services need to be re-exposed?

How are external systems integrated? (REST, SOAP, file exchange)

Any cloud provider is preferred? (AWS, Azure, GCP)

DevOps CI/CD and automation be structured?

What observability and monitoring tools will be used for performance tracking? (in existing system do we have any 3rd party monitoring tools),

1. What are the common pain points in the current system?
2. Who are the primary users, and what roles do they have?
3. How does data flow across different modules? Can you share existing architecture diagram ?
4. Are there any third-party integrations in the current system?
5. What is the dependency between various modules?
6. What should be improved in terms of user experience and responsiveness?
7. How is authentication handled (RBAC, OAuth, LDAP, etc.)?
8. How does data persistence work—what databases are currently used?
9. How are external systems integrated (REST, SOAP, file exchange)?
10. Is there an API layer, or will services need to be exposed?
11. How are external systems integrated (REST, SOAP, file exchange)?
12. Are there any dependencies on specific Java libraries or features?
13. What security protocols and compliance standards must be adhered to?
14. Are there existing security vulnerabilities that need to be addressed?
15. What are the availability and uptime requirements?
16. What disaster recovery and backup strategies need to be implemented?
17. What cloud provider should be used (AWS, Azure, GCP)?
18. Should we adopt a serverless approach for specific workloads?
19. What kind of autoscaling mechanisms are preferred?
20. Are containerized deployments needed (Docker, Kubernetes)?
21. How will CI/CD be structured?
22. What should be automated for deployment?
23. How will logging and monitoring be managed?
24. What cloud regions should be targeted?
25. Should we implement multi-cloud or hybrid-cloud architecture?
26. How should cost optimization be considered during cloud migration?
27. Are we already using regression testing ?
28. What observability mechanisms should be integrated?
29. What monitoring tools will be used?

**Top 50 Questions to Ask the Client (Including Functional and NFR Requirements)**

**Business & Functional Requirements**

1. What are the primary business objectives behind modernization?
2. What key features should be retained from the existing application?
3. Are there any new functionalities to be added?
4. How is the existing application being used by end users?
5. What are the common pain points in the current system?
6. Who are the primary users, and what roles do they have?
7. How does data flow across different modules?
8. Are there any third-party integrations in the current system?
9. What is the dependency between various modules?
10. What should be improved in terms of user experience and responsiveness?

**Technical Questions**

1. Is the existing Java Swing application a monolith?
2. How is authentication handled (RBAC, OAuth, LDAP, etc.)?
3. What programming language versions and frameworks are in use?
4. How does data persistence work—what databases are currently used?
5. How is data structured—relational, NoSQL, flat files?
6. How are business rules currently implemented?
7. Is there an API layer, or will services need to be exposed?
8. How are external systems integrated (REST, SOAP, file exchange)?
9. How will the new system interact with legacy applications, if needed?
10. Are there any dependencies on specific Java libraries or features?

**Non-Functional Requirements (Performance, Security, Scalability)**

1. What are the expected performance benchmarks for the new system?
2. What peak load does the current system experience?
3. How does user concurrency affect performance?
4. Are there specific latency requirements for API responses?
5. What security protocols and compliance standards must be adhered to?
6. Are there existing security vulnerabilities that need to be addressed?
7. How should data encryption and storage be handled?
8. How should user authentication and access control be managed?
9. What are the availability and uptime requirements?
10. What disaster recovery and backup strategies need to be implemented?

**Cloud & Deployment Strategy**

1. What cloud provider should be used (AWS, Azure, GCP)?
2. Should we adopt a serverless approach for specific workloads?
3. What kind of autoscaling mechanisms are preferred?
4. Are containerized deployments needed (Docker, Kubernetes)?
5. How will CI/CD be structured?
6. What should be automated for deployment?
7. How will logging and monitoring be managed?
8. What cloud regions should be targeted?
9. Should we implement multi-cloud or hybrid-cloud architecture?
10. How should cost optimization be considered during cloud migration?

**Testing & Maintenance**

1. What test automation strategies should be in place?
2. What regression testing is required during modernization?
3. How will performance testing be conducted?
4. What observability mechanisms should be integrated?
5. How will post-deployment maintenance be handled?
6. How should rollback strategies be designed?
7. What end-user training is required post-modernization?
8. What monitoring tools will be used?
9. How frequently should patches and upgrades be applied?
10. How will we handle feature rollout and updates?

**Top 20 Planning Points for Migration**

1. **Web & Mobile Support** –new architecture is cross-platform.
2. **Microservices Approach** – Decompose the monolith into independent services.
3. **API Gateway Implementation** – Introduce API management and security.
4. **Role-Based Access Control (RBAC)** – Implement secure access.
5. **Authentication Upgrade** – Integrate OAuth2.0 and SSO if needed.
6. **Event-Driven Architecture** – Use Kafka or RabbitMQ for real-time events.
7. **Containerization Strategy** – Deploy services in Docker/Kubernetes.
8. **CI/CD Pipeline** – Automate testing and deployments.
9. **Data Migration Plan** – Strategically migrate databases.
10. **Cloud Strategy** – Define deployment on AWS/Azure/GCP.
11. **Autoscaling & Load Balancing** – Implement elastic scaling.
12. **Security Enhancements** – Ensure data encryption, WAF, and threat monitoring.
13. **Performance Optimization** – Use caching, async processing, etc.
14. **API Versioning & Governance** – Ensure future-proof APIs.
15. **Infrastructure as Code (IaC)** – Automate infra using Terraform or Bicep.
16. **Monitoring & Observability** – Integrate App Insights, Dynatrace, etc.
17. **Database Optimization** – Optimize indexing, partitioning, caching.
18. **Hybrid-Cloud Considerations** – Ensure smooth integration.
19. **DevOps Collaboration** – Establish structured collaboration workflows.
20. **Regulatory Compliance** – Adhere to necessary compliance standards.