Lernzettel

Middleware- und Web-Technologien: Messaging (JMS, AMQP), RPC-Frameworks (gRPC, RMI), Web-APIs (REST, GraphQL)

Universität:	Technische Universität Berlin
Kurs/Modul:	Architektur von Anwendungssystemen
Erstellungsdatum:	September 19, 2025

Lernzettel: Middleware- und Web-Technologien: Messaging (JMS, AMQP), RPC-Frameworks (gRPC, RMI), Web-APIs (REST, GraphQL)

(1) Messaging: JMS, AMQP.
Grundidee: asynchroner Nachrichtenaustausch, lose Kopplung von Komponenten, oft Broker-basierte Architektur.

JMS (Java Message Service).

- API-Standard zur asynchronen Nachrichtenverarbeitung in Java-Anwendungen.
- Kernkonzepte: Producer, Consumer, Destination (Queue/Topic), Message-Header, Payload.
- Persistenz, Transaktionen, QoS-Optionen (z.B. Delivery-Mode: persistent vs non-persistent).
- Typische Broker: ActiveMQ, IBM MQ, HornetQ (bzw. Artemis).
- Einsatzszenarien: Ereignisgesteuerte Architektur, Lastverteilung, Entkoppelung von Produzenten und Konsumenten.

AMQP (Advanced Message Queuing Protocol).

- Offenes, standardisiertes Wire-Protokoll für Messaging, Broker-Architektur.
- Wichtige Konzepte: Exchange-Typen (direct, fanout, topic, headers), Queues, Bindings, Routing-Keys.
- Delivery-Guarantees: at-most-once, at-least-once, exactly-once (je nach Implementierung).
- Channels, Frames, TLS-Sicherheit, Transaktionen/Confirmations.
- Typische Broker: RabbitMQ, Apache Qpid.

Vergleich JMS vs. AMQP.

- JMS: API-Sicht, plattformunabhängig innerhalb der Java-Welt; Broker-spezifisch implementiert.
- AMQP: Protokollebene, interoperabel über verschiedene Sprachen hinweg; beschreibt das Messaging-Modell und die Zuverlässigkeit.
- Kombination: JMS-Clients können auf AMQP-Brokern betrieben werden, oft über JMS-Provider.

Nicht-funktionale Aspekte (Messaging).

- Fehlertoleranz und Wiederholungen (Retry-Strategien, Dead-Letter-Queues).
- Ordering-Garantie je nach Topologie (Queues vs. Topics).
- Skalierbarkeit durch Partitionierung/Clusterbildung des Brokers.
- Observability: Metriken, Logging, Tracing von Nachrichtenpfaden.

Beispiele/Notizen.

- Einsatzbeispiel: Ereignis-Domäne, bei der Produzenten Event-Nachrichten an Consumer-Gruppen senden.
- Design-Hint: Verwende Messaging für asynchrone Kommunikation und Entkopplung, nicht als Ersatz für notwendige Synchronschnittstellen.

(2) RPC-Frameworks: gRPC, RMI.
RPC ermöglicht entfernte Funktionsaufrufe, Unterschiede in Stil, Sprachen und Protokollen.

gRPC.

- Modernes RPC-Framework basierend auf HTTP/2 und Protobuf/Protobuf-Textformat.
- Service-Definitionen in .proto-Dateien; Code-Generierung für viele Sprachen.
- Unterstützt RPC-Synchronous-, Server-Streaming-, Client-Streaming- und Bidirectional-Streaming-Modelle.
- Starke Typisierung, Efficiency durch kompaktes Binärformat.
- Sicherheitsoptionen: TLS, mTLS, Interceptors, authentication/authorization.
- Typische Einsatzszenarien: Microservices-Kommunikation, boundary-crossing APIs, Performanz-Anforderungen.

RMI (Remote Method Invocation).

- Java-spezifisches RPC-Paradigma zur ferngestützten Ausführung von Methoden auf entfernten Objekten.
- Syntax-getriebene Schnittstellen, Proxy-/Stub-Mechanismus, objektrelationale Serialisierung.
- Typische Grenzen: Sprach- und Plattformgebundenheit, Sicherheits- und Versionsprobleme über JVM-Grenzen hinweg.
- Einsatzszenarien: Alte oder konsistente Java-Monolithen, interne Service-Kommunikation innerhalb homogener Ökosysteme.

Vergleich gRPC vs. RMI.

- Sprachübergreifende Interoperabilität: gRPC (breite Sprachunterstützung) vs RMI (Java-spezifisch).
- Datenformat: gRPC nutzt Protobuf, kompakt und schnell; RMI nutzt Java-Serialisierung.
- Modi: gRPC unterstützt Streaming, RMI primär synchrone/Remote-Calls.

Designhinweise (RPC).

- RPC eignet sich gut für synchrone, performerintensive Interaktionen; Messaging eignet sich besser für asynchrone, entkoppelte Kommunikation.

- Berücksichtige Fehlertoleranz, Latenzanforderungen, Backwards-Compatibility der Schnittstellen.

(3) Web-APIs: REST, GraphQL.
Web-APIs ermöglichen den Zugriff auf Dienste über das Web-Protokoll HTTP.

REST (Representational State Transfer).

- Resource-basierte Architektur: Identifiziere Ressourcen über URIs, nutze HTTP-Methoden (GET, POST, PUT, PATCH, DELETE).
- Statelessness, Cachebarkeit, idempotente Operationen, sinnvolle Statuscodes.
- Designprinzipien: Ressourcenorientierung, HATEOAS optional, Versionierung oft via URI oder Header.
- Vorteile: Einfachheit, Skalierbarkeit, breite Tool-Unterstützung.

GraphQL.

- Abfrageorientierte API-Schnittstelle; Client requestet genau die benötigten Felder.
- Starke Typisierung über Schema, Resolver-Logik auf der Serverseite; eine einzige Endpoint-URL.
- Vorteile: Vermeidung von Over- oder Under-fetching, flexiblere Client-Entwicklung.
- Herausforderungen: Komplexität bei Caching, Sicherheit und Query-Performance-Management, Overhead bei kleinen APIs.

Vergleich REST vs GraphQL.

- REST: Klarer, gut standardisierter Weg, robuste Caching-Strategien, einfache Tooling-Einbindung.
- GraphQL: Größere Flexibilität, reduzierte Round-Trips, aber höhere Komplexität bei Server-Management, Caching und Sicherheit.

Designhinweise (Web-APIs).

- REST eignet sich gut für einfache, stabile Ressourcen-APIs mit klaren CRUD-Operationen.
- GraphQL lohnt sich bei klientenseitiger Flexibilität und komplexen Abfrageanforderungen; bedenke Caching-Strategien.

Architekturüberlegungen.

- Asynchron vs Synchronous: Messaging und GraphQL/REST mit asynchronen Mustern kombinieren (z. B. Event-sourcing).
- Service-Kompositionsmöglichkeiten: Sprechen Sie je nach Anforderung entweder RPC (direkter Aufruf) oder REST/GraphQL (Resource-/Query-basierter Zugriff).

- Sicherheit, Governance, Versionierung, Observability (Tracing, Logs, Metriken).

Zusammenfassung.

- JMS/AMQP adressieren asynchrone, entkoppelte Kommunikation und Messaging-Patterns.
- gRPC bietet leistungsfähiges, plattformübergreifendes RPC mit Streaming-Unterstützung.
- RMI dient primär Java-internaler Remote-Kommunikation (versions- und plattformabhängig).
- REST und GraphQL decken unterschiedliche API-Paradigmen ab: Ressourcenzugriffe vs. flexible Abfragen.