**11.2.1**

Beurteilung des Architekturstils anhand der folgenden Kriterien:

**Performace, Leistung:**

-Komponenteninteraktionen können der dominante Faktor für die wahrgenommene Leitungsfähigkeit einer Netzwerk-basierten Applikation sein

-Network Performance:

-Wie beschaffen ist das Netzwerk und wie viel kann eine Applikation

Von diesem Netzwerk nutzen?

-User-Perceived Performance:

-Latenz

-completion- time

-Network Efficiency:

-je weniger das Netzwerk genutzt werden muss, desto besser

**Scalability:**

-Fähigkleit eine Große Anzahl an Nutzer zu unterhalten während einer aktiven Konfiguration

-vereinfachen

-delegieren, dezentralisieren

**Simplicity:**

-Prinzip der Teilung: Aufgaben auf einzelne Komponenten aufgeteilt.

-> einzelne Komponenten weniger komplex

**Modifiability:**

-Wie einfach lässt sich eine Änderung an der Architektur durchführen?

-Evolvability:

-Inwiefern kann die Implementierung auf Komponentenebene geändert

Werden, ohne negativen Einfluss auf andere Komopnenten zu nehmen?

-Extensibility:

-Fähigkeit weitere Funktionalitäten zum System hinzuzufügen

-Customizability:

-inwieweit lässt sich temporär das Verhalten eines Elements ändern, sodass

Es eine ungewöhnliche Aufgabe durchführen kann, ohne andere Clients

Negativ zu beeinflussen

-Configurability

-

-Reusability:

-Komponenten, Verbindungen und Elemente lassen sich wiederverwenden

**Visibility:**

-Fähigkeit eine Komonente die Interaktion zwischen zwei anderen Komponenten zu überwachen und zu vermitteln

**Portability:**

-data elements are constrained to a set of formats

**Reliability:**

-Wie wahrscheinlich ist es dass das System ausfällt

**11.2.2**

**Client-Server:**

-Seperation of concerns: storage and ui

-Serverkomponenten können vereinfacht werden und nur für bestimmte Aufgaben dediziert werden

->besser skalierbar (für best. Aufgaben können best. Komponenten erstellt werden)

-UI über verschiedene Plattformen (da Datenspeicherung und Darstellung beim Client getrennt)

-durch Trennung Client-Server können sich die Komponenten unabhängig voneinander entwicklen

**Stateless:**

-jeder Request eines Clients muss über alle Informationen verfügen um den Request zu verstehen

-keine Informationen über den Kontext zu einer best. Server.

-bessere Scalability: server muss nicht Daten ziwschen Requests speichern.

-server kann Ressourcen schneller freigeben

-einfachere Implementierung

-Reliability:

-

-Visibility:

**Cache:**

-cash-Beschränkungen -> client-cash-stateless-server-Style

-> daten sollen als cachable und nicht cacheable markiert werden.

-Latenz wird verringert, dadurch dass bestimmte Elemente nicht erneut geladen werden müssen

-allerdings kann die Information im Cache bei erneutem Abruf bereits ungültig sein.

**Uniform Interface**

-uniform interface between components

-unabhängige Weiterenwicklung, da Implementierung nicht Teil der Schnittstelle

-jedoch ineffizient, da allgemeine Schnittstelle nicht immer zu den spezifischen Bedürfnissen einer

Applikation passt.

**Layered System**

-Hinzufügen von Layern mit seinen eigenen Komponenten (Abstraktionsstufen)

+Komplexität kann so einfacher in den Griff bekommen werden

-Overhead

-höhere Latenz

**Code-On-Demand**

**-**Funktionalität wird erweitert durch Skripts und Applets