Friday, 18 January 2019 12:08

11.1

HTTP-Methode	CRUD-Operation	Beschreibung
GET	Read	Fordert die angegebene Ressource vom Server an.
PUT	Update	Die durch die URI spezifizierte Ressource wird mit den Daten aus dem Rumpf aktualisiert oder es wird eine neue Ressource mit den Daten aus dem Rumpf angelegt, falls die angegeben URI nicht existiert.
POST	Create	Fügt eine neue Ressource unterhalb einer angegebenen Ressource ein. Da die neue Methode noch keine URI besitzt, wird die URI der Elternressource angegeben. Als Ergebnis der Operation wird die URI der neu angelegten Ressource zurück gegeben.
DELETE	Delete	Löscht die angegebene Ressource.
OPTIONS		Prüft, welche Methoden auf einer Ressource zur Verfügung stehen.
HEAD		Fordert die Metadaten (HTTP Header-Attribute) einer Ressource an.

Eigenschaften: - safe : read-only -> get, head , options

- indempotent : resultat bei mehrmaligem ausführen = resultat bei einmaligem ausführen
-> put, delete und safe methoden

- Cacheable: Antworten können für zukünftige Wiederverwertung gespeichert werden
-> "this specification defines GET, HEAD, and POST as cacheable, although the overwhelming majority of cache implementations only support GET and HEAD."

11.2

a) Der Architekturstil wird nach folgenden Eigenschaften bewertet:
- Performance
(Network Performance [wie gut bzw schnell das Netzwerk Infos bewegt],
User-perceived Performance[hauptsächlich Latenzzeiten und completion time],
Network Efficiency [durch möglichst wenige Netzwerkinteraktionen , zb. Durch caching])
- Scalability: Fähigkeit große Zahlen an Komponenten und deren Interaktion zu supporten
"Scalability can be improved by simplifying components, by distributing services across many components (decentralizing the interactions), and by controlling interactions and configurations as a 33 result of monitoring. "
- Simplicity : durch unterteilung in einfache Komponenten
- Modifiability (Evolvability : inwieweit man einzelne Komponente verändern kann ohne andre zu beeinflussen)
(Extensibility : wie einfach man Funktionalitäten hinzufügen kann)
(Customizability : inwieweit man Fähigkeit der Komponente verändern kann)
(Configurability: post-deployment modifikation an Komponenten)
(Reusability: Wiederverwertung zb in andren Applikationen)
- Visibility: Fähigkeit einer Komponente andere Komponenten zu überwachen/ steuern
- Portability : ob Software auf in andren Umgebungen läuft
- Reliabilty : wie zuverlässig

b) Constraints:
- Client Server: als architekturstil , Separation von UserInterface und DataStorage -> besssere Portability und Scalability
- Stateless Client Server Interaction, jeder Request von Client an Server muss alle Infos beeinhalten-> besssere

visibility, reliability, and scalability

- Cache: ggf wird response data für zukünfitg, gleiche requests gecached -> vorteile : "eliminate some interactions, improving efficiency, scalability, and userperceived performance by reducing the average latency of a series of interactions."
- Uniform Interface: zwischen Komponenten , erleichtert generelle system architektur und die Visibility der Interaktionen ist verbessert. Implementationen getrennt von Services die sie providen-> bessere evolvability allerdings schlechtere efficiency
- Layered System: hierarchische Schichten , jede Schicht sieht nur benachbarte Schicht, mit der sie interagiert Verbessert overall complexity und scalability
- Code an demand : erlaubt Client Funktionalität in form von applets herunterzuladen und auszuführen. das simplified clients, weil nicht schon alle features implementiert sein müssen und verbesser extensibility verringert allerding visibility -> nur optionaler constraint