Architekturbegründung

Die Anwendungsarchitektur basiert auf Datensätzen, die von Nutzern aus einer Datensammlung abgerufen und geschrieben werden können. Wichtig ist dabei, dass die Datensätze nicht verteilt über verschiedene Anwender gespeichert werden, sondern an einer zentralen Stelle gesammelt und gespeichert werden. Im Gegensatz zu anderweitig verteilten Architekturen wie Peer-to-Peer, bietet sich eine Client-Server-Architektur für dieses Projekt durch die Strukturierung des Anwendungslogik bestens an.

Daten werden über den HTTP Body in Form eines JSON Datenformats übertragen. Sowohl Requests als auch Responses werden als JSON übertragen. In Frage kommende Alternativen wurden abgewogen.

Unabhängig des Datenformats kann die strukturierte Kommunikation über REST stattfinden. Eine standardisierte Schnittstelle hätte ebenfalls den Vorteil, dass potentielle andere Client-Anwendungen die gleiche Server-Schnittstelle verwenden können. Nach Absprache mit den Dozenten wurde außerdem festgelegt, dass der Service auf Level 3 nach Richardsons Maturity Model (HATEOAS, Hypermedia as the engine of application state)[1] umgesetzt wird.

Die Datenstruktur soll sehr einheitlich und durchgängig sein. Theorethisch kann jeder Beitrag, egal ob Hauptbeitrag oder Kommentar, als ein Beitrag mit Eltern und Kindern gesehen werden. Jeder Beitrag hat damit 1 Elternteil und eine Liste aus Kindern. Diese in der Datenbank abgespeicherten Kinder und Eltern können geschachtelt als IDs abgespeichert werden, wobei jene IDs bei der Ausgabe mit den eigentlichen Objekten ersetzt (“*populated*” werden).

Diese Ausgabe muss noch als Proof Of Concept bestätigt werden.

Beispiel für eine *populated* Ausgabe:

```
[{
  id: 1
  parent: 0,
  title: “Medieninformatik”,
  children: [{
    id: 2,
    parent: 1,
    title: “Ein Beitrag”,
    children: [{
      id: 3,
      parent: 2,
      title: “Ein Kommentar”,
      children: [{ ... }]
```

[1] Webber, REST in Practice, Chapter 1, “Level Three Services”

```
    }, {
      id: 4,
      parent: 2,
      title: "Weiterer Kommentar",
      children: [{ ... }]
    }]
  }, {
    id: 5,
    parent: 1,
    title: "Weiterer Beitrag",
    children: [{ ... }]
  }]
}, {
  id: 6,
  parent: 0,
  title: "Wirtschaftsinformatik",
  ...
}]
```

Server

Der Server übernimmt dabei die Schnittstelle hauptsächlich zum Lesen und Schreiben von persistenten Daten. Dazu kommt noch Logik zum Versenden von Benachrichtigungen, Mails und Authentifizierung sowie Authorisierung der Nutzer. Jene Funktionalitäten benötigen eine zentrale Anlaufstelle und können vom Client nicht übernommen werden. Trotzdem soll der Server nicht mit vielen zusätzlichen Funktionalitäten ausgestattet werden, um ihn als simpel aufrufbare Schnittstelle mit wenig unterschiedlichen Rückgabewerten nutzen zu können. Dies spart außerdem Zeit bei der Entwicklung des Clients, da Aufrufe generalisierbar sind.

Client

Neben der zunächst einfach anmutenden Anzeige der Datensätze, übernimmt der Client doch einige Aspekte der Anwendungslogik. Als offensichtlichster Punkt zählt hierzu die Strukturierung der vom Server ausgelieferten Daten, da der Server (wie oben beschrieben) die Rolle einer sehr einfachen Schnittstelle einnimmt. Weiterhin soll die Verifizierung von Studenten nur über die Client-Applikation und nicht über ein weiteres Element wie einem Browser stattfinden, um nach außen hin die Wirkung zu erzielen, dass der Client das einzige Element der Applikation ist.

Internet

Zwischen Client und Server befindet sich das Internet als Kommunikationsschicht über das gängige HTTP Protokol. In dieser Schicht befinden sich systemfremde Komponenten wie Mailserver und Google Cloud Messaging, die während des Ablaufs in Anspruch genommen werden.
Zum Mailvsersand wird SMTP als Protokoll eingesetzt. Google Cloud Messaging besitzt eine API über HTTP.