Dokumentation

Daniel Hildebrandt, Tim Prott, Tobias Kavsek

Table of Contents

Dokumentation Dienstanbieter

Ressourcen, HTTP Verben und deren Semantik

Ressource	Methodik	Semantik	content-type (req)	content-type (res)
/user	GET	Ruft eine Nutzerliste auf	text/plain	application/json
/user/:id	GET	Ruft eine Liste von Cocktails auf, die von diesem Nutzer erstellt wurden	text/plain	application/json
/user	PUT	Speichert einen neuen Usereintrag	application/json	application/json
/user	POST	aktualisiert einen Usereintrag	application/json	application/json
/user/:id	DELETE	Löscht einen Usereintrag	text/plain	text/plain
/cocktails	GET	Ruft eine Liste bestehender Cocktails auf	text/plain	application/json
/cocktails/:name	GET	Ruft ein spezifisches Cocktailsrezept auf, mit dazu eine Liste der benötigten Zutaten.	text/plain	application/json
/cocktails	PUT	Speichert einen neuen Cocktail in das System, dazu die passende Liste mit Zutaten	application/json	application/json
/cocktails	POST	Aktualisieren eines Cocktails	application/json	application/json
cocktails/:name	DELETE	Löscht einen Cocktail aus dem System	text/plain	text/plain
/cocktails/:name/ingredients	GET	Ruft eine Zutatenliste zu einem einzelnen Cocktail ab	text/plain	application/json
/cocktails/:name/ingredients	POST	Aktualisiert einen Zutateneintrag eines einzelnen Cocktails	application/json	application/json
/ingredients	GET	Ruft einer Liste aller jemals verwendeten Zutaten auf	text/plain	application/json
/ingredient/:name	GET	Ruft eine Beschreibung einer einzelnen Zutat auf	text/plain	application/json

Ressource	Methodik	Semantik	content-type (req)	content-type (res)
/ingredients	PUT	Speichert eine neue Beschreibung einer Zutat in das System.	application/json	application/json
/ingredients	POST	Updaten eines Zutateneintrags	application/json	application/json
/ingredients/:id	DELETE	Löscht eine Zutat aus dem System	application/json	text/plain
/cocktails/:name/comments	GET	Ruft alle Kommentare auf, die zu einem Cocktail verfasst wurden	text/plain	application/json
/cocktails/:name/comments	POST	Verfasst einen Kommentar zu einem Cocktail	application/json	application/json

Statuscodes

Code	Status	Bedeutung
200	OK	Anfrage erfolgreich bearbeitet
201	Created	Anfrage Bearbeitet, Ressource wurde erstellt
204	No Content	Die Anfrage wurde erfolgreich durchgeführt, die Antwort enthält jedoch bewusst keine Daten.
205	Reset Content	Die Anfrage wurde erfolgreich durchgeführt; der Client soll das Dokument neu aufbauen und Formulareingaben zurücksetzen.
301	Moved Permanently	Die angeforderte Ressource steht ab sofort unter der im „Location“-Header-Feld angegebenen Adresse bereit (auch Redirect genannt). Die alte Adresse ist nicht länger gültig.
303	See Other	Die Antwort auf die durchgeführte Anfrage lässt sich unter der im „Location“-Header-Feld angegebenen Adresse beziehen. Der Browser soll mit einem GET folgen, auch wenn der ursprüngliche Request ein POST war.
400	Bad Request	Die Anfrage-Nachricht war fehlerhaft aufgebaut.
401	Unauthorized	Die Anfrage kann nicht ohne gültige Authentifizierung durchgeführt werden. Wie die Authentifizierung durchgeführt werden soll, wird im „WWW-Authenticate“-Header-Feld der Antwort übermittelt.
403	Forbidden	Die Anfrage wurde mangels Berechtigung des Clients nicht durchgeführt, bspw. weil der authentifizierte Benutzer nicht berechtigt ist, oder eine als HTTPS konfigurierte URL nur mit HTTP aufgerufen wurde.

Code	Status	Bedeutung
404	Not Found	Die angeforderte Ressource wurde nicht gefunden. Dieser Statuscode kann ebenfalls verwendet werden, um eine Anfrage ohne näheren Grund abzuweisen. Links, welche auf solche Fehlerseiten verweisen, werden auch als Tote Links bezeichnet.
405	Method Not Allowed	Die Anfrage darf nur mit anderen HTTP-Methoden (zum Beispiel GET statt POST) gestellt werden. Gültige Methoden für die betreffende Ressource werden im „Allow“-Header-Feld der Antwort übermittelt.
408	Request Time-out	Innerhalb der vom Server erlaubten Zeitspanne wurde keine vollständige Anfrage des Clients empfangen.
423	Locked	Die angeforderte Ressource ist zurzeit gesperrt.
500	Internal Server Error	Dies ist ein „Sammel-Statuscode“ für unerwartete Serverfehler.
501	Not Implemented	Die Funktionalität, um die Anfrage zu bearbeiten, wird von diesem Server nicht bereitgestellt. Ursache ist zum Beispiel eine unbekannte oder nicht unterstützte HTTP-Methode.
507	Insufficient Storage	Die Anfrage konnte nicht bearbeitet werden, weil der Speicherplatz des Servers dazu zurzeit nicht mehr ausreicht.

Definition der Ressourcen mit Alternativen

Zunächst wurden Cocktails als Ressourcen definiert. Diese bestehen aus Name, Zubereitungsanleitung und einer Liste von Zutaten und ihren Mengen. Zutaten sind eigene Ressourcen mit Name und Beschreibung. Nutzer sind eigene Ressourcen. Diese enthalten Name, Email-Adresse und Passwort.

cocktail

```
{
    "name": "name_of_cocktail",
    "desc": "detailed_recipe",
    "mail": "email of user",
}
```

comment

```
{
    "auth": "author of the comment",
    "comm": "written commentary"
}
```

ingredients

```
[
    {
        "ingr": "author of the comment",
        "meng": "count of ingredients"
    }, ...
]
```

Beschreibung der Funktionalität und Datenstrukturen

Der Dienstanbieter definiert dabei die Datenbank-Logik. Für die Datenhaltung selbst boten sich mehrere Lösungen an. In Frage kamen SQL-Devariate, NoSQL-Devariate, sowie einfaches speichern in eine Datei. Um Relationen gut darstellen zu können, ohne ein komplexes Schema zu definieren, entschieden wir uns für das NoSQL-Devariat Redis. Der Dienstanbieter soll hierbei allerdings nach Außen hin unabhängig des implementierten Systemes arbeiten.

Aus Zeitmangel nicht umgesetzte Funktionalität

- Im Dienstanbieter konnten alle Features funktional implementiert werden.
- Probleme gab es nur seitens der update-Methoden, welche noch nicht funktional lauffähig sind.
- Leerzeichen im Cocktailnamen müssen ersetzt werden, da diese sonst einen "unescaped sequence"-Fehler werfen.

Dokumentation des Dienstnutzers:

Funktionalität und Datenstrukturen

- Neue Cocktails sollen angelegt werden
- Neue Nutzer sollen angelegt werden
- Nutzer sollen Kommentare für einzelne Cocktails verfassen
- Zutaten werden individuell mit ihrer Menge eingespeichert
- Später wurde für das zu den Zutaten gehörende Eingabefeld eine kleine Markup-Language erstellt
- Cocktails sollen gelöscht werden.

Use Cases

Anlegen eines neuen Cocktails

- POST /cocktails
- POST /cocktails/:name/ingredients
- POST über Twitter API

Table 1. Anlegen eines Cocktails

Bedingung	**Schritte**
Name	Anlegen eines neuen Cocktails
Beschreibung	Ein neuer Cocktail wird mit allen wichtigen Daten erstellt
Auslösendes Ereignis	Aktivierung seitens des Nutzers
Vorbedigung	Keine
Hauptszenario	1. Nutzer trägt Name ein 2. Nutzer trägt Email ein 3. Nutzer trägt eine Zubereitungsanleitung ein 4. Nutzer trägt eine liste von Zutaten ein
Alternativszenario	1. Cocktail existiert bereits a. In dem Fall kann der Neue Cocktail nicht erstellt werden 2. Zutaten wurden falsch eingeben a. Cocktail muss neu erstellt werden
Nachbedingung	1. Cocktail wird gespeichert 2. Kommentare können gemacht werden

Abrufen einer Liste aller Cocktails

- GET /cocktails
    1. Abrufen einer Liste aus Cocktails

Bedingung	**Schritte**
Name	Abrufen einer Liste aus Cocktails
Beschreibung	Eine Liste aller erstellten Cocktails
Auslösendes Ereignis	NV/Erstellen eines neuen Cocktails
Vorbedingung	Mindestens ein Cocktail muss existieren
Hauptszenario	1. Nutzer ruft die Liste auf.

Bedingung	Schritte
Alternativszenario	1. Keine
Nachbedingung	1. Liste bestehender Cocktails existiert.

Abrufen eines Spezifischen Cocktails mit Kommentaren

- GET /cocktails/:name
- GET /cocktails/:name/ingredients
- GET /cocktails/:name/comments
    1. Abrufen eines spezifischen Cocktails

Bedingung	Schritte
Name	Abrufen eines spezifischen Cocktails
Beschreibung	Ein Cocktail mit Zutatenliste, Rezept und Kommentaren wird angezeigt
Auslösendes Ereignis	Aktivierung durch Nutzer
Vorbedingung	Der abzurufende Cocktail muss existieren
Hauptszenario	1. Cocktail wird angezeigt 2. Kommentare werden angezeigt
Alternativszenario	1. Keines
Nachbedingung	1. View für ein Cocktail ist aufgebaut 2. Kommentare können geschrieben werden

Verfassen eines neuen Kommentars

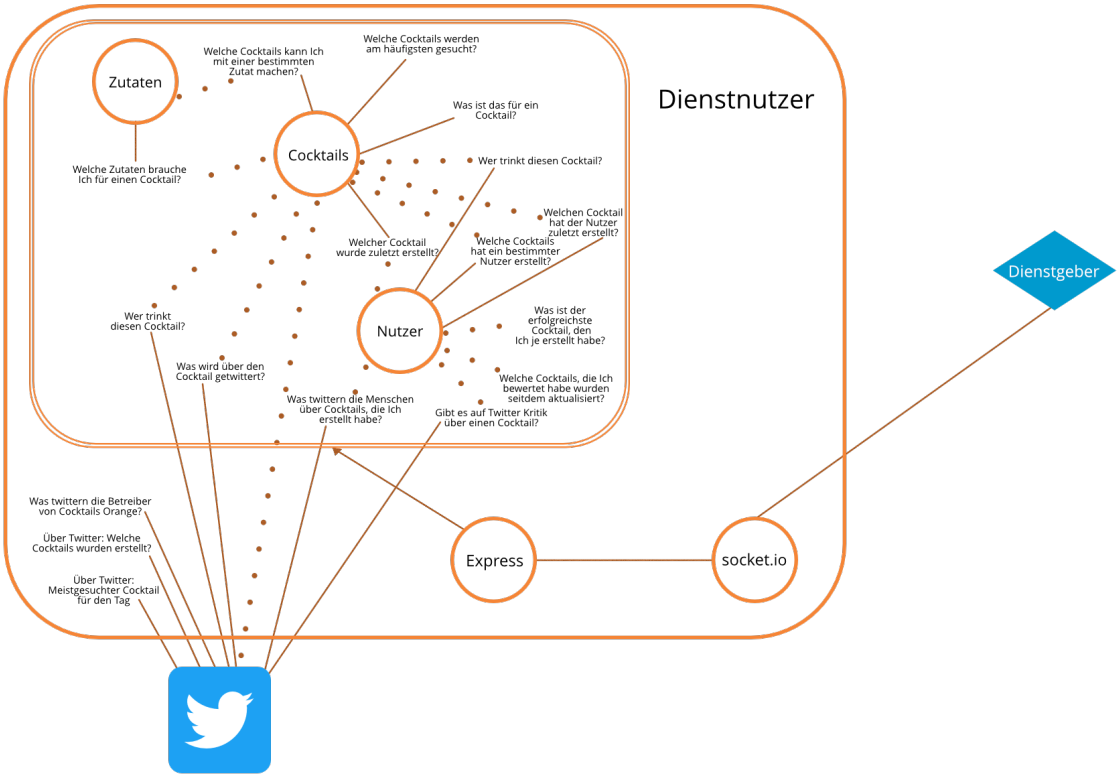
- POST /cocktails/:name/comments

Table 2. Anlegen eines Cocktails

Bedingung	Schritte
Name	Verfassen eines neuen Kommentars
Beschreibung	Ein Kommentar zu einem spezifischen Cocktail wird angelegt
Auslösendes Ereignis	Aktivierung durch den Nutzer
Vorbedingung	Der abzurufende Cocktail muss existieren; Der Nutzer muss auf der View des Cocktails sein

Bedingung	Schritte
Hauptszenario	1. Nutzer gibt seinem Namen ein 2. Nutzer gibt seinen Kommentar ein 3. Nutzer sendet den Kommentar
Alternativszenario	1. Kommentar enthält ungültige Symbole a. In dem Fall kann der Kommentar nicht abgeschickt werden
Nachbedingung	1. Kommentar Existiert in der Liste mit Kommentaren

Externer Webservice



Zunächst wurde Amazon als API in Betracht gezogen, jedoch aus Kostengründen verworfen. Folgend wurde mit 'twit' die Twitter-API implementiert. Über einen eigenen Twitter-Account werden neu erstellte Cocktails automatisch über Twitter beworben. Der aktuellste Tweet ist im Dienstnutzer einsehbar.

Asynchron implementierte Teile

Über socket.io wurde asynchroner Datenaustausch implementiert. Die implementierten Features funktionieren über das publish-subscribe-Konzept. Der aktuellste Tweet, sowie die Liste aller Nutzer werden so in kurzen Intervallen automatisch und asynchron vom Client über den Dienstnutzer bezogen.

Aus Zeitmangel nicht umgesetzte Funktionalität

Aus Zeitmangel wurde das Löschen und Aktualisieren der einzelnen Komponenten seitens des Dienstanbieters noch nicht implementiert. Auch die Liste mit Cocktails, die eine bestimmte Zutat enthalten, wurde nicht implementiert. Nicht alle vorgesehenen Statuscodes konnten bisher implementiert werden.

Dokumentation des Prozesses:

Beschreibung der Vorgehensweise, auch der Irrwege

1. Entwurf eines groben Konzeptes
    a. Cocktails sollen erstellt werden können
    b. Zutaten sollen erstellt werden können
    c. Nutzer sollen erstellt werden können
    d. Cocktails sollen kommentiert und bewertet werden können
    e. Bewertungen wurden abgeschafft
2. Produktergebnisse sollen über die Amazon-API für einzelne Cocktails abrufbar sein
3. Aus kostentechnischen Gründen wurde die Amazon-API verworfen
4. Festlegung auf API, die sich zum werben eignet
5. Die Twitter-API soll dazu dienen, neue Cocktails zu promoten
6. Ressourcen wurden definiert
    a. Cocktails
    b. Zutaten
    c. Nutzer
7. Dienstgeber wurde programmiert
    a. Rohimplementierung der einzelnen Anforderungen
    b. Debugging fehlerhafter Implementierungen
    c. Testing
    d. Error-Handling
    e. Implementieren der HTML-Statuscodes
8. Dienstnutzer wurde programmiert
    a. Rohimplementierung der einzelnen Anforderungen
    b. Debugging fehlerhafter Implementierungen
    c. Testing
    d. Error-Handling
    e. Implementieren der HTML-Statuscodes
    f. Implementierung Twitter-API
    g. Vergleich socket.io, faye
        i. resultat: socket.io schneller zu implementieren
    h. Implementierung socket.io
        i. Tweets des eigenen Accounts [pub]
        ii. Aktuelle Nutzerliste [pub]

i. Testing

Fazit

Der Umfang und Arbeitsaufwand des Projektes lies die Implementierung aller geplanten Funktionen, aufgrund von Zeitmangel nicht zu. Somit wurde das persönliche Ziel, alle Ideen bis zum ersten Abgabetermin umzusetzen nicht erfüllt. Daher wird die zusätzliche Zeit in den Semesterferien dazu genutzt, die fehlenden Funktionen zu implementieren und nicht wie geplant, nur zum Feinschliff. Dennoch sind, basierend auf der uns zur Verfügung gestellten Checkliste, alle Anforderungen erfüllt.

Arbeitsmatrix

Aktivität	Daniel Hildebrandt	Tobias Kavsek	Tim Prott
Projektkonzept	10%	20%	70%
Dienstanbieter Programmierung	35%	40%	25%
Dienstanbieter Implementierung	33%	33%	33%
Dienstanbieter Testen/Debuggen	40%	20%	40%
Dienstnutzer Programmierung	35%	55%	10%
Dienstnutzer Implementierung	33%	33%	33%
Dienstnutzer Testen/Debuggen	45%	35%	20%
Dokumentation	18%	65%	17%