Leo Knittel, Valentin Spiroski und Konstantin Gerlach

Arbeitsanteile:

Leo Knittel:

- app.js entsprechend angepasst
- borrows.json
- borrowsController.js

Valentin Sprioski:

- equipment.json
- equipmentController.js

Konstantin Gerlach:

- alle routes
- usersController.js
- users.json

Anforderungen und deren Umsetzung:

Users:

Per Post soll ein neuer User hinzugefügt werden und per get sollen alle oder bei Angabe einer ID als URL Parameter ein einziger User ausgegeben werden. Über put lässt sich ein User editieren und mit delete löschen. Wird ein neuer Nutzer via Post erstellt, wird zunächst mit der crypto-Bibliothek eine zufällige ID generiert. Diese kann per put auch nicht mehr veröndert werden. Danach wird geprüft, ob auch alle erforderlichen Daten im post body übertragen wurden, ansonsten wird eine Fehlermeldung ausgegeben. Es wird allerdings nicht geprüft, ob die Email tatsächlich valider Email-Syntax entspricht. Das eingebene Passwort wird nicht im Klartext gespeichert, sondern mit der crypto Bibliothek gehasht. Das registryDate ist der Zeitpunkt der Regisitrierung und wird per Date() erzeugt, allerdings nicht weiter umformatiert. Die Ausgabe, die Löschung und das Aktualisieren eines bestimmten Users durchlaufen zunächst alle User und wenn ein User mit der gesuchten ID existiert, erfolgt die entsprechende Aktion. Die User-Daten werden in einem Array verwaltet und in einer JSON Datei gespeichert.

Equipment:

Die Verwaltung der Ausleihartikel erfolgt über verschiedene Endpunkte der REST-API. Die Routen ermöglichen es, alle Artikel abzurufen (GET equipment), einen neuen Artikel hinzuzufügen (POST /equipment), einen spezifischen

Artikel anhand seiner ID abzurufen (GET /equipment/:id), einen Artikel zu aktualisieren (PUT /equipment/:id) oder zu löschen (DELETE /equipment/:id). Zusätzlich wurden Filterfunktionen für das Abrufen von Artikeln nach Jahr und Monat implementiert. Beim Hinzufügen eines neuen Artikels wird eine eindeutige ID generiert, und es wird sichergestellt, dass die EAN-Nummer 13-stellig und numerisch ist. Das Kaufdatum wird automatisch auf das aktuelle Datum gesetzt. Die Artikeldaten werden in der equipment.json Datei gespeichert. Die Fehlermeldungen umfassen das Nichtvorhandensein eines Artikels bei Abruf, Aktualisierung oder Löschung sowie eine unzureichende Validierung der Eingabedaten, insbesondere der EAN-Nummer.

Borrows:

Auf eine POST-Anfrage wird ein neuer Ausleihvorgang hinzugefügt. Über GET können alle Ausleihvorgänge abgerufen werden, oder bei Angabe einer ID als URL-Parameter wird ein bestimmter Vorgang ausgegeben. PUT ermöglicht es, einen bestehenden Ausleihvorgang zu aktualisieren, während DELETE die Löschung eines Ausleihvorgangs ermöglicht. Sobald ein neuer Ausleihvorgang per POST hinzugefügt wird, entsteht automatisch eine zufällige ID, die durch kryptische Verschlüsselung äußerst einzigartig ist. Es wird überprüft, ob alle benötigten Daten im Anfragekörper vorhanden sind, andernfalls wird eine Fehlermeldung ausgegeben. Die Ausgabe, Löschung und Aktualisierung eines bestimmten Ausleihvorgangs durchlaufen alle Ausleihvorgänge. Gespeichert werden alle Daten in der borrows.json Datei.

Was konnte nicht umgesetzt werden?

Borrows

Die Verbindung von Benutzer-ID und Ausleihartikel-IDs ist nicht vollständig implementiert worden. Bisher werden die CRUD-Operationen für Ausleihvorgänge ohne detaillierte Verknüpfung und Referenzierung der beteiligten Benutzer und Ausleihartikel durchgeführt.

Test-Daten:

Users:

siehe users.json

Borrows:

Hier wurden zufällige Testdaten verwendet, die über ThunderClient in Visual Studio Code eingespeist wurden. Dabei wurden verschiedene Anfragen an die REST-API gesendet, um sicherzustellen, dass die implementierten Endpunkte

wie erwartet funktionieren. Die Testdaten umfassten typische Szenarien, die während des Betriebs der Ausleihplattform auftreten können. Diese werden dann in der json Datei gespeichert.

Equipment:

Die Testdaten in equipment.json umfassen verschiedene Ausleihartikel wie Laptops und Beamer. Alle CRUD-Operationen wurden mit diesen Beispieldaten getestet, um sicherzustellen, dass die API wie erwartet funktioniert.

Abgedeckte Fehlerfälle:

- Zugriffsversuch auf User mit ID, die nicht existiert (GET, PUT und DELETE)
- Erstellung eines Users mit unvollständigen Daten
- Zugriffsversuche auf nicht existierende Artikel oder fehlerhafte Eingaben werden entsprechend behandelt.
- Fehlende oder fehlerhafte EAN-Nummern werden abgelehnt.

Borrows:

Unvollständige Daten: - Im Controller addBorrow wird überprüft, ob die erforderlichen Felder (userID, borrowDate, returnDate,equipmentID) im req.body vorhanden sind. - Falls eines dieser Felder fehlt, wird ein 400er-Statuscode mit der Nachricht “Incomplete data” zurückgegeben.