

GigTool Dokumentation

Willkommen bei der Dokumentation unserer API-Applikation GigTool.

Ein Herzensprojekt von Robin Harris, Hendrik Lendeckel, Dario Daßler und Max Schelenz für die Inventarisierung und Organisation von Musikequipment.

Wir möchten Musikern dabei helfen ihr Equipment zu verwalten, um sich auf das eigentlich Wichtige - die Musik fokussieren zu können.

Hinweis: Unser Projekt wurde wie besprochen als Verbesserungsversuch für Java 1 angemeldet.

Zum Einstieg möchten wir euch erstmal zeigen, wie ihr unsere API-Applikation installieren könnt.

Inhaltsverzeichnis

- GigTool Dokumentation
 - Inhaltsverzeichnis
 - Installationsanleitung
 - Architektur
 - Komponenten
 - api
 - storage
 - db
 - docker
 - maven
 - Interfaces (wird noch überarbeitet)
 - Größere Architekturänderungen im Vergleich zu Java 1
 - Calc
 - WeightClass und WeightClassList
 - EquipmentList
 - Happening
 - Rental und Gig
 - Dimension
 - Anwenderdokumentation
 - Intelli
 - Postman
 - Auflistung Endpoints
 - Besonderheiten/Features des Projekts
 - Lessons Learned
 - Über den Projektumfang hinausgehende Funktionen

Installationsanleitung

[Docker Desktop](https://www.docker.com/products/docker-desktop/) (<https://www.docker.com/products/docker-desktop/>) benötigen wir für die Docker Engine.

Die [IntelliJ IDE](https://www.jetbrains.com/de-de/idea/download/?section=windows) (<https://www.jetbrains.com/de-de/idea/download/?section=windows>) nutzen wir im späteren Verlauf für die Tests der Services.

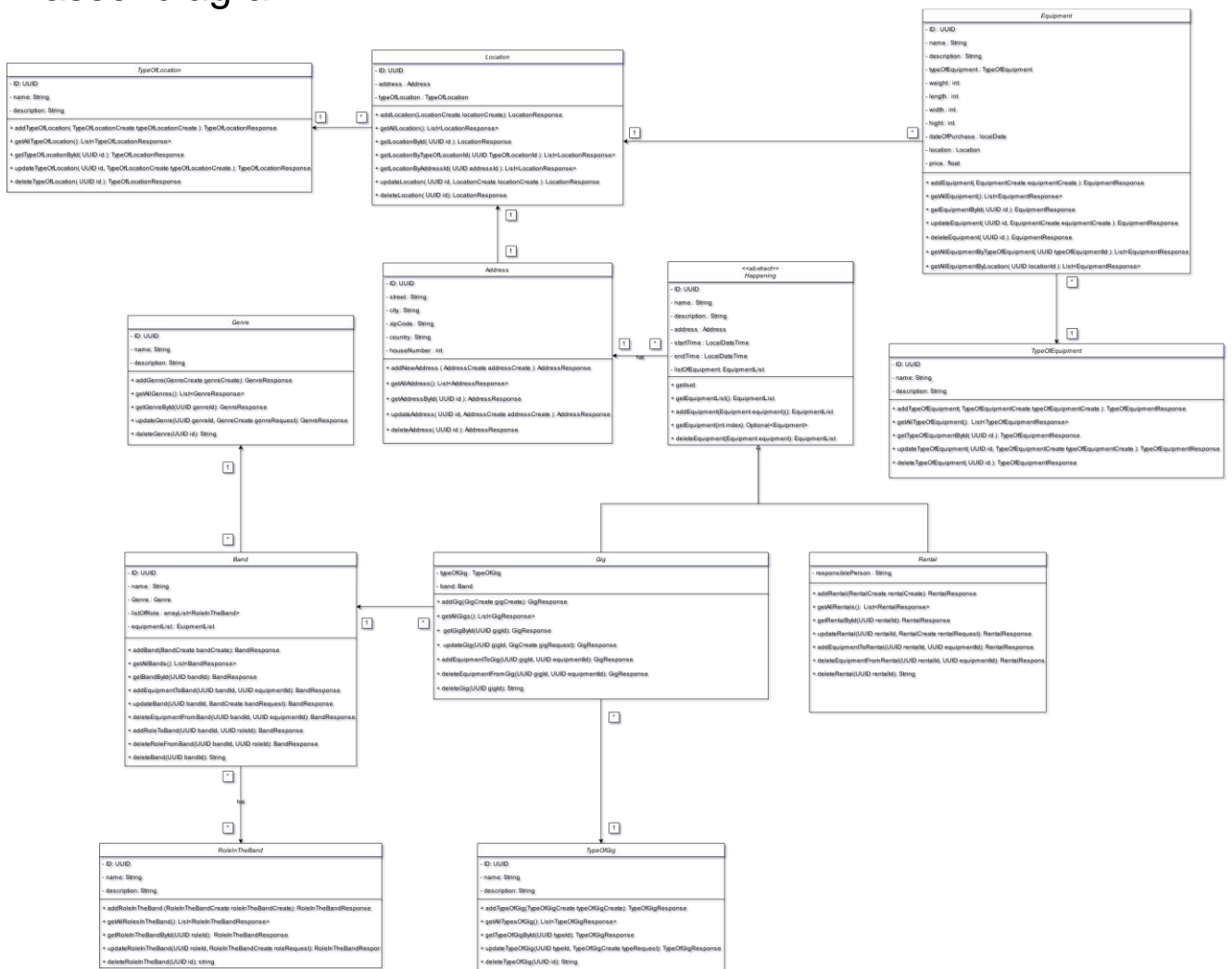
Die [Postman-App](https://www.postman.com/downloads/) (<https://www.postman.com/downloads/>) werden wir später in der Anwenderdokumentation nutzen, um unsere Routen und Controller zu testen.

```
git clone https://git.ai.fh-erfurt.de/prgj2-23/gigtool.git
docker compose up
```

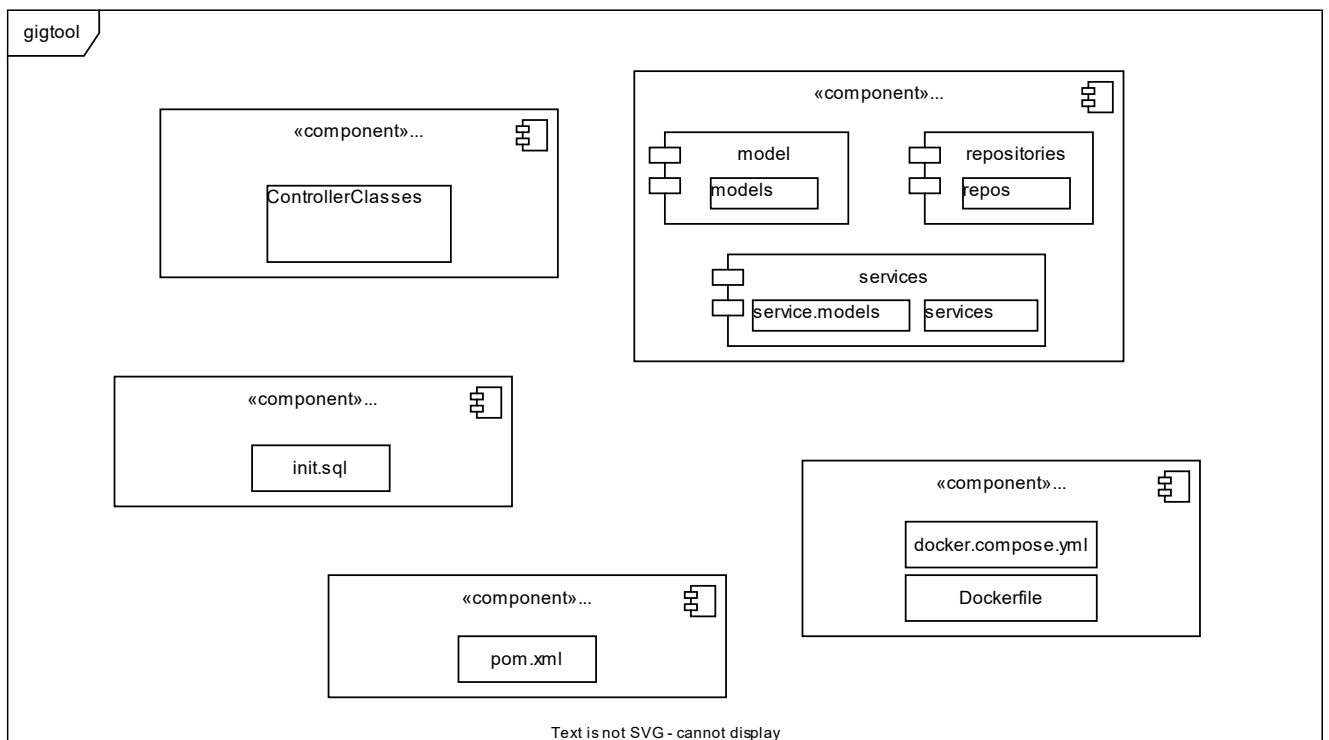
Architektur

Die GigTool Architektur umfasst mehrere strukturelle Bereiche, die wir euch in den kommenden Abschnitten verständlich näher bringen wollen.

Klassendiagramm



Komponenten



api

Wie im Komponentendiagramm zu erkennen, enthält die api alle wichtigen Controller-Klassen. Diese sind für das Routing der Anfragen zu den Endpoints zuständig.

storage

Die storage Komponente ist unterteilt in verschiedene Module.

Das "model" ist für die Definition der Datenbank-Objekte vorgesehen und beschreibt deren Aussehen.

Die repos im "repositories"-Modul, benötigen wir um überhaupt Datenbank-Operationen anwenden zu können. Sie sind bei Bedarf beliebig erweiterbar, falls die JPA-Funktionalitäten nicht ausreichen.

Das services-Modul ist nochmal unterteilt in service.models und services. Die service.models definieren die Struktur der create- und response-Objekte. Services sind für die Trennung von Geschäftslogik und Datenbankzugriff zuständig.

db

Die Komponente "db" enthält unser Initialisierungsskript mit dem Namen der Datenbank und dem root-User zur Verwaltung.

docker

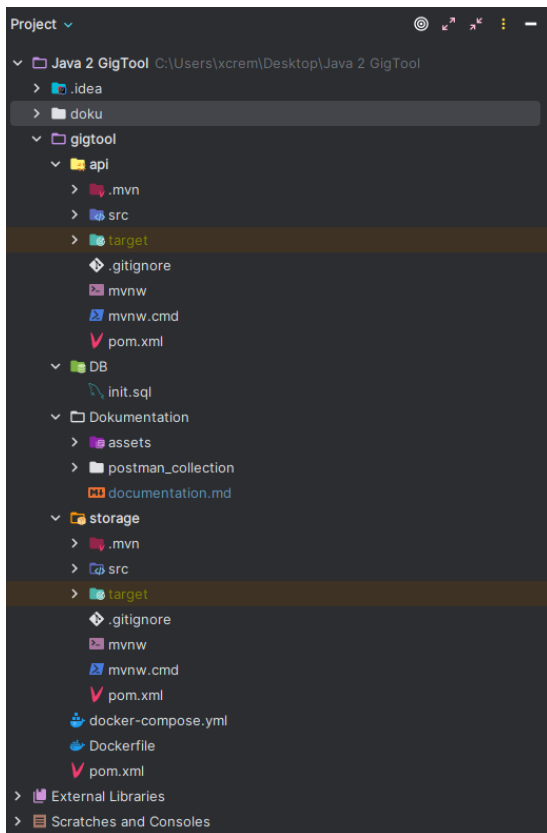
Die docker-compose.yml ist unsere docker-Komponente im Projekt. Sie sorgt für die Containerisierung unserer ApiApplication, PostgreSQL-DB und dem Verwaltungstool pgadmin.

maven

Die Maven POM enthält alle wichtigen Abhängigkeiten für unser Spring Boot-Projekt und stellt sozusagen den Build-Planer dar.

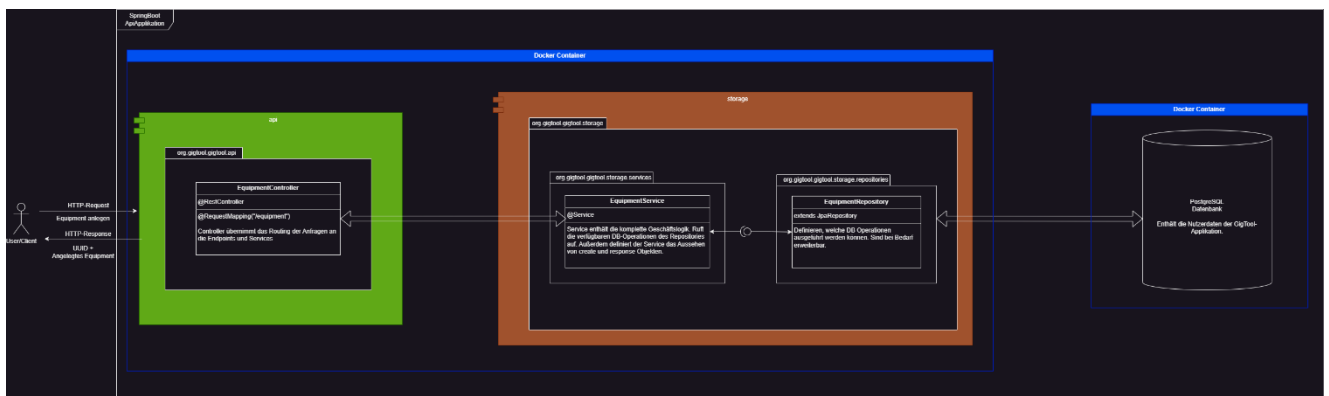
Ordnerstruktur

Unsere Struktur im Hauptordner gigtool ist ähnlich der Komponenten gegliedert: api, DB, Dokumentation, storage, docker und pom.xml.



Interfaces

Im folgenden Schaubild zeigen wir euch, wie unser API-Interface funktioniert und welche Abläufe im backend angestoßen werden, wenn der User beispielsweise ein Equipment anlegen möchte.



Größere Architekturänderungen im Vergleich zu Java 1

Unser Projekt haben wir von Java 1 weitergeführt. Der Grad an Komplexität ist jedoch deutlich gestiegen und wir haben mehrere Umbauten vornehmen müssen.

Calc

Diese Utility-Class wurde entfernt, da wir festgestellt haben, dass alle Funktionen auf entsprechende Services aufgeteilt werden konnten.

WeightClass und WeightClassList

Diese Klassen waren gedacht um das Equipment, anhand des Gewichts zu Kategorisieren. Diese Kategorien (WeightClasses) waren aufsteigend in der WeightClassList gespeichert. Das sollte dazu dienen, Aussagen über das Gewicht/Gesamtgewicht geben zu können. Um die Klassenstruktur zu vereinfachen wurden diese Klassen gestrichen. Aussagen über das Gewicht/Gesamtgewicht können trotzdem getroffen werden, da das Eigengewicht in der Entität Equipment gespeichert wird.

EquipmentList

Diese Hilfsklasse wurde entfernt und durch das direkte Verknüpfen von Equipments in Happenings und Band realisiert.

Rental und Gig























Diese Klassen erben nun nicht mehr von Happening. Sie verweisen jedoch auf ihr Happening.

Dimension

Diese Klasse wurde entfernt, weil der Sinn und Zweck erhalten bleibt, wenn wir Länge, Breite, Höhe direkt in der Entität Equipment speichern.

Anwenderdokumentation

Wenn alle in der Installationsanleitung beschriebenen Schritte erfolgreich ausgeführt wurden, sollte die ApiApplication, PostgreSQL-Datenbank und pgadmin auf der Docker Engine laufen

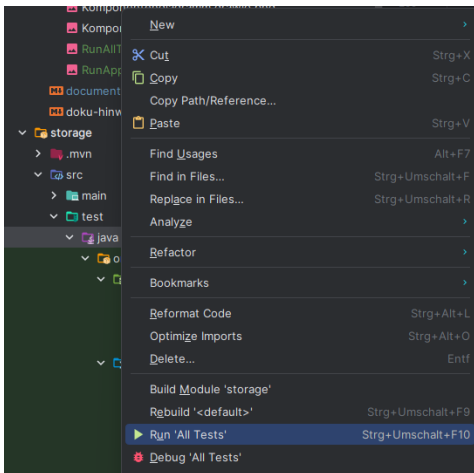
<input type="checkbox"/>		gigtool		Running (3/3)	23 minutes ago				
<input type="checkbox"/>		api 25dcd6df4019 	gigtool-gigtool-api	Running	8080:8080 	23 minutes ago			
<input type="checkbox"/>		postgres a75140302845 	postgres:12	Running	5432:5432 	23 minutes ago			
<input type="checkbox"/>		pgadmin 602c3670f916 	dpage/pgadmin4:latest	Running	5050:80 	23 minutes ago			

Nun sind wir endlich soweit die Funktionalität von GigTool zu testen.

IntelliJ

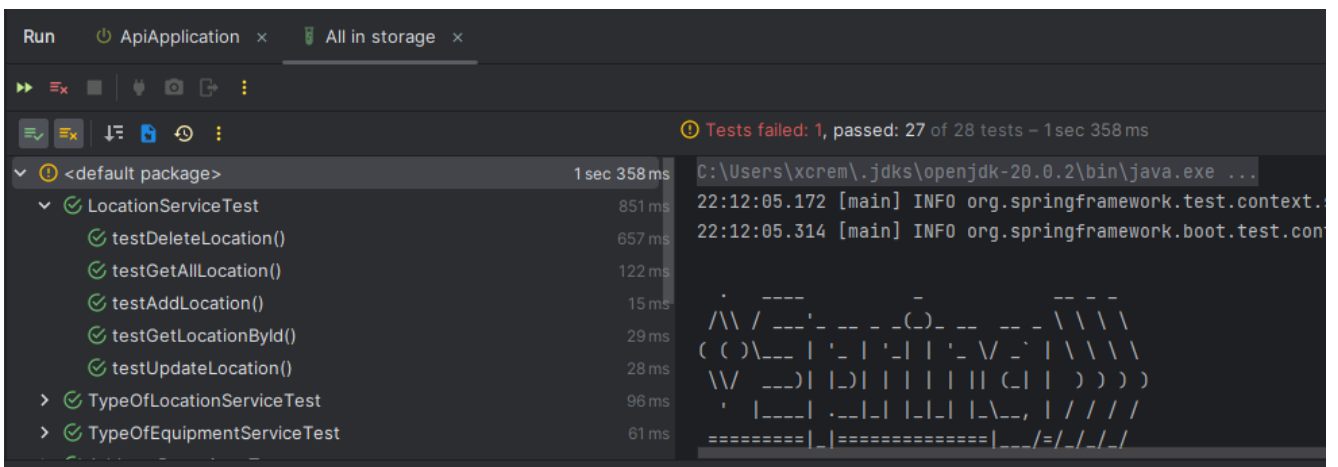
Für den Test der Services und Repositories können wir in der IntelliJ-IDE einen Testdurchlauf starten.

Mittels Rechtsklick auf den java-Testordner gelangen wir in das Optionsmenü, in dem wir " Run 'All Tests' " ausführen können.



Nun sollten alle vorhandenen Tests durchgeführt werden und im Konsolenfenster der IDE bekommen wir eine Auskunft über alle erfolgreichen oder auch fehlgeschlagenen Tests.

Zum Beispiel:



Postman

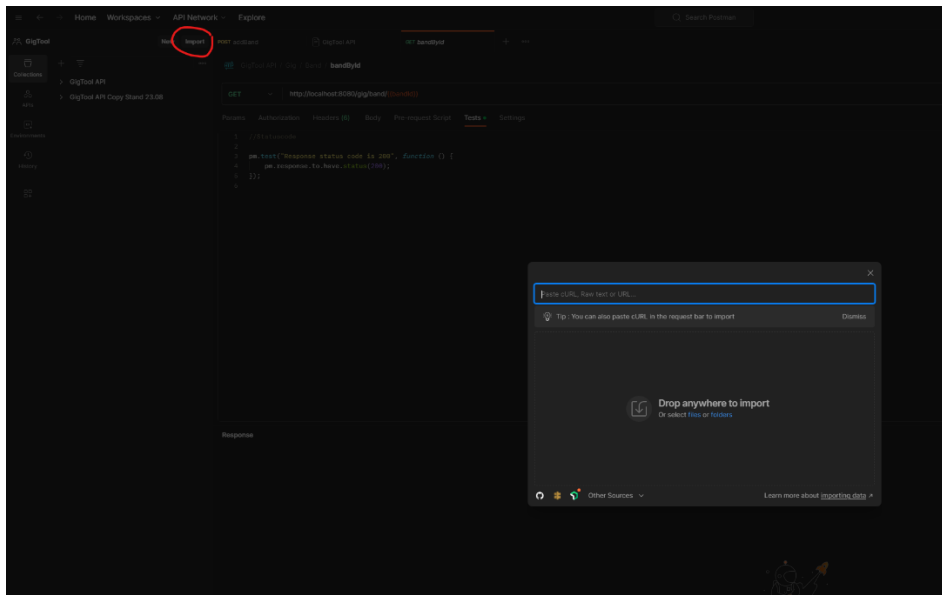
Postman ist eines der beliebtesten Werkzeuge zum Testen von APIs (Application Programming Interfaces).

Auflistung Endpoints

HTTP Methode	URL	Service	Controller
POST	/addresses	addAddress	AddressController
GET	/addresses	getAllAddresses	AddressController
GET	/addresses/{id}	getAddressById	AddressController
PUT	/addresses/{id}	updateAddress	AddressController
DELETE	/addresses/{id}	deleteAddress	AddressController
POST	/typeOfLocation	addTypeOfLocation	TypeOfLocationController
GET	/typeOfLocation	getAllTypeOfLocation	TypeOfLocationController
GET	/typeOfLocation/{id}	getTypeOfLocationById	TypeOfLocationController
PUT	/typeOfLocation/{id}	updateTypeOfLocation	TypeOfLocationController
DELETE	/typeOfLocation/{id}	deleteTypeOfLocation	TypeOfLocationController
POST	/typeOfEquipment	addTypeOfEquipment	TypeOfEquipmentController
GET	/typeOfEquipment	getAllTypeOfEquipment	TypeOfEquipmentController
GET	/typeOfEquipment/{id}	getTypeOfEquipmentById	TypeOfEquipmentController
DELETE	/typeOfEquipment	deleteTypeOfEquipment	TypeOfEquipmentController
PUT	/typeOfEquipment/{id}	updateTypeOfEquipment	TypeOfEquipmentController
POST	/location	addLocation	LocationController
GET	/location	getAllLocation	LocationController
GET	/location/{id}	getLocation	LocationController
DELETE	/location/{id}	deleteLocation	LocationController
PUT	/location/{id}	updateLocation	LocationController
POST	/equipment	addEquipment	EquipmentController
GET	/equipment	getAllEquipment	EquipmentController
GET	/equipment/byType/{typeOfEquipmentId}	getAllEquipmentByTypeOfEquipment	EquipmentController
GET	/equipment/byLocation/{locationId}	getAllEquipmentByLocationId	EquipmentController
GET	/equipment/{id}	getEquipmentById	EquipmentController
PUT	/equipment/{id}	updateEquipment	EquipmentController
DELETE	/equipment/{id}	deleteEquipment	EquipmentController
POST	/gig/typeOfGig	addTypeOfGig	TypeOfGigController
GET	/gig/typeOfGig	getAllTypeOfGig	TypeOfGigController
GET	/gig/typeOfGig/{id}	getTypeOfGigById	TypeOfGigController
UPDATE	/gig/typeOfGig/{id}	updateTypeOfGig	TypeOfGigController
DELETE	/gig/typeOfGig/{id}	deleteTypeOfGig	TypeOfGigController
POST	/gig/band	addBand	BandController
GET	/gig/band	getAllBand	BandController
GET	/gig/band/{id}	getBandById	BandController
UPDATE	/gig/band/{id}	updateBand	BandController
PUT	/gig/band/{id}/equipment/{id}	addEquipment	BandController
DELETE	/gig/band/{id}/equipment/{id}	deleteEquipment	BandController
PUT	/gig/band/{id}/roleintheband/{id}	addEquipment	BandController
DELETE	/gig/band/{id}/roleintheband/{id}	deleteEquipment	BandController
DELETE	/gig/band/{id}	deleteBand	BandController
POST	/gig	addGig	GigController
GET	/gig	getAllGig	GigController
GET	/gig/{id}	getGigById	GigController
PUT	/gig/{id}/equipment/{id}	addEquipmentToGig	GigController
DELETE	/gig/{id}/equipment/{id}	deleteEquipmentFromGig	GigController
UPDATE	/gig/{id}	updateGig	GigController
DELETE	/gig/{id}	deleteGig	GigController
POST	/rental	addRental	RentalController
GET	/rental	getAllRental	RentalController
GET	/rental/{id}	getRentalById	RentalController
PUT	/rental/{id}/equipment/{id}	addEquipmentToRental	RentalController
DELETE	/rental/{id}/equipment/{id}	deleteEquipmentFromRental	RentalController
UPDATE	/rental/{id}	updateRental	RentalController
DELETE	/rental/{id}	deleteRental	RentalController
POST	/gig/band/roleintheband	addRoleInTheBand	RoleInTheBandController
GET	/gig/band/roleintheband	getAllRoleInTheBand	RoleInTheBandController
GET	/gig/band/roleintheband/{id}	getRoleInTheBandById	RoleInTheBandController
PUT	/gig/band/roleintheband/{id}	updateRoleInTheBand	RoleInTheBandController
DELETE	/gig/band/roleintheband/{id}	deleteRoleTheBand	RoleInTheBandController
POST	/gig/band/genre	addGenre	GenreController
GET	/gig/band/genre	getAllGenre	GenreController
GET	/gig/band/genre/{id}	getGenreById	GenreController
PUT	/gig/band/genre/{id}	updateGenre	GenreController
DELETE	/gig/band/genre/{id}	deleteGenre	GenreController
GET	/timetable	getAllHappenings	TimetableController
GET	/timetable?name=Gig&des....	getFilterHappenings	TimetableController
GET	/timetable/{id}/calc	getTotalValuesOfHappening	TimetableController
GET	/timetable/{id}/whereismyequipment	getLocationsOfEquipmentFromHapp	TimetableController

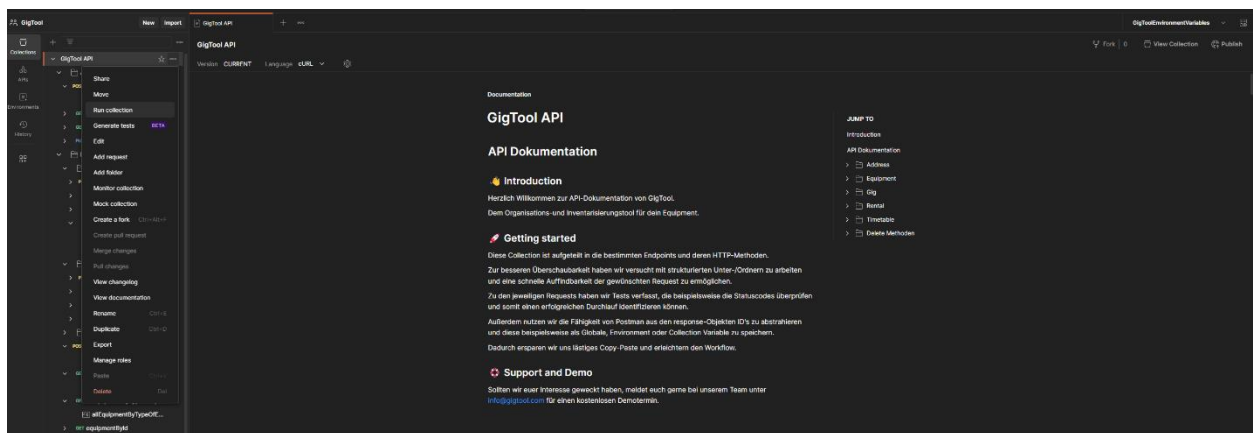
Import der Collection

Die GigTool Postman Collection findet ihr im /gigtool/dokumentation/postman_collection Ordner. Diese könnt ihr ganz simpel per Drag&Drop in Postman importieren.

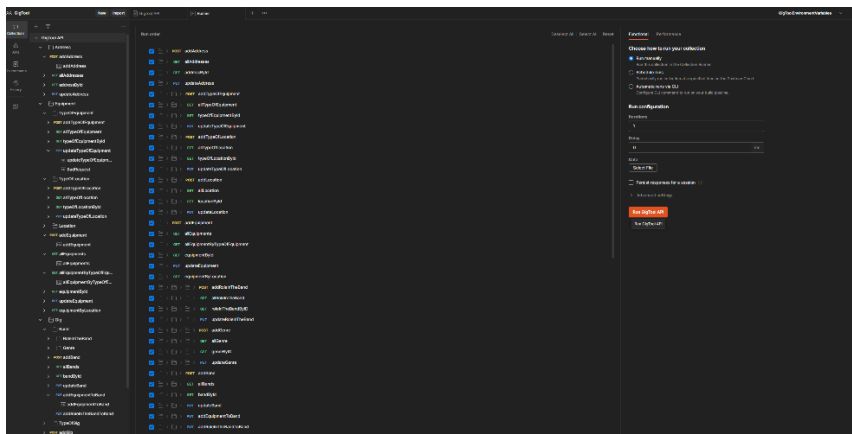


Ausführung der Collection

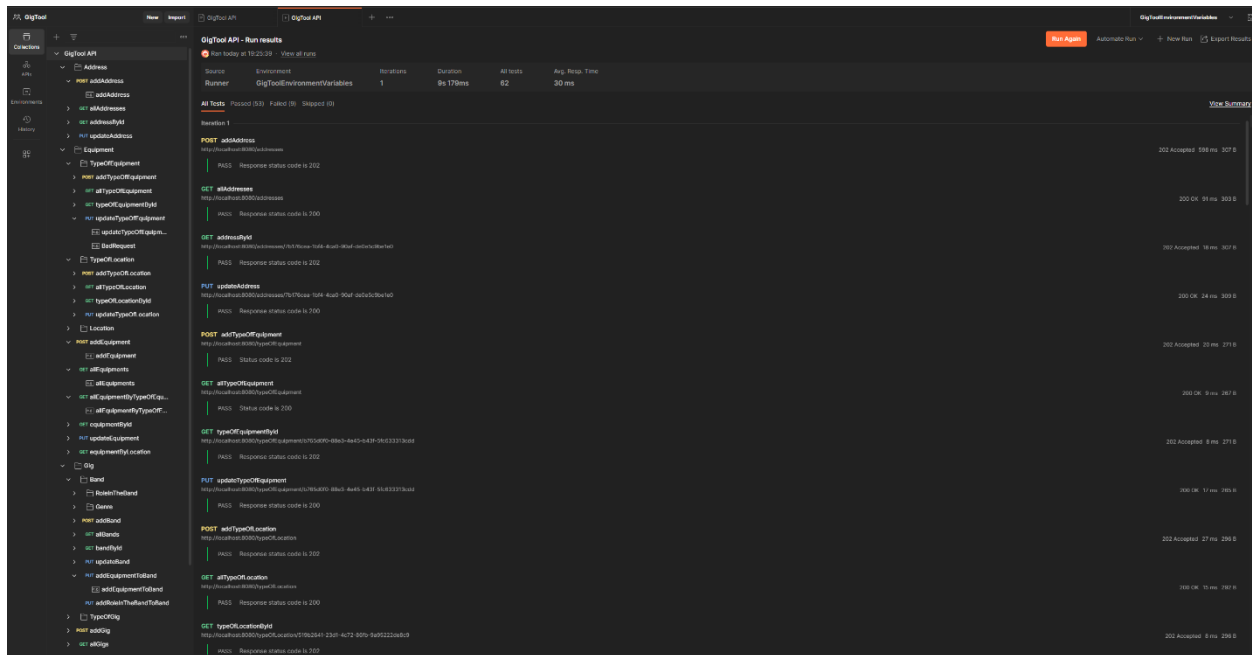
Mittels Rechtsklick auf den Collection_Ordner könnt ihr die komplette Collection ausführen.



Das GigTool API Run durchläuft dann alle gespeicherten Methoden...



...und überprüft deren Ergebnisse auf Erfolg oder Fehlschlag.



Im Postman Documentation Bereich haben wir für jeden Endpoint Beschreibungen und Beispiele hinzugefügt, um einen möglichst detaillierten Einblick zu gewährleisten.

Besonderheiten/Features des Projekts

Im Laufe des Projekts haben sich ein paar coole Features angesammelt, die es in unseren Augen verdienen hier nochmal erwähnt zu werden.

TestUtils

Als Random-Generator für unsere Servicetests nutzen wir unsere test_utils Klasse. Das erspart uns beim Erstellen der vielen Testszenarien des Öffern wertvolle Zeit.

Kollisionsprüfungen

GigTool verhindert durch verschiedene Überprüfungen Kollisionen bei der Verwaltung des Equipments. Folgende Kollisionsprüfungen werden von unserer Application abgedeckt:

- mehrere Gigs des Musikers können nicht gleichzeitig stattfinden
- Equipment ist nicht doppelt während eines Zeitslots verbuchbar (Verleihung während eines Gigs, welcher das Equipment benötigt, ist nicht möglich)
- Instrument, welches einer Band zugeordnet ist, darf automatisch nicht während des Auftritts dieser Band verliehen werden
- bei Änderungen des Zeitraums eines Happenings wird der neue Zeitpunkt erneut auf Kollisionen überprüft

Timetable

Diese Klasse stellt in unseren komplexen Datenstrukturen eine Art allmächtige Gottklasse dar. Ebenso führt sie klassenübergreifende Requests aus wie: WhereIsMyEquipment und Calculate. Sie enthält die Funktionalitäten, welche unseren Kalender abbilden sollen.

Dazu gehört:

- das Auflisten aller Happenings
- eine Filterfunktion für Happenings
- eine Suchfunktion für Equipments

Lessons Learned

Aufbau einer API-Applikation in Java

- *Spring Boot*
- *Maven POM*
- *JPA repos*
- *dockerize Application, PostgreSQL*
- *Postman Collection + Workspaces*
- *Location, Address und TypeOfLocation sind sehr komplex verschachtelt => hier hatten wir große Ambitionen sind dann aber an einen zu hohen Grad an Komplexität gelangt*
- *bei Happening ist uns der Schritt der Vereinfachung gelungen => hier konnten wir ein einheitliches Zeitformat (Zeit + Datum) für alle Happenings einführen und unsere Datenstrukturen optimieren*
- *schreiben von JavaTest-Klassen mithilfe von test_utils*

Über den Projektumfang hinausgehende Funktionen

Responses könnten mit einer response_utils noch weiter verfeinert werden. Beispielsweise zu individuellen Messages bei nicht 2xx-Statuscodes.

Bei dem Hinzufügen von mehreren Equipments zu einer Band könnte man anstelle von mehreren Aufrufen mit path-Parametern

auch eine Liste der hinzuzufügenden Equipments im Body übergeben. Dadurch könnten wir den Traffic noch mehr optimieren und die Usability verbessern.

Unsere Applikation könnte durch weitere Get-Requests oder Suchfunktionen nochmals funktionell erweitert werden, z.B. getAllBandsWithGenre oder searchByDescription.

Troubleshooting and Known Bug's

Die Unittests müssen manchmal mehrmals komplett ausgeführt werden. Ab und an schlagen Tests fehl. Wir vermuten, dass dieser Fehler im Zusammenspiel der TestUtils und der Transactional-Notation bei einigen Tests Fehlverhalten oder Inkonsistenz hervorruft. Diesen Bug konnten wir bislang noch nicht beheben. Deshalb bedarf es manchmal der mehrmaligen Ausführung der Tests, bis alle "grün" und erfolgreich durchlaufen sind.

Fazit

Das Projekt GigTool hat uns sehr viel Spaß gemacht, aber auch aufgrund der Komplexität und dem ersten Kontakt mit SpringBoot einiges abverlangt.

Trotz allem sind wir nun sehr zufrieden mit dem Endergebnis.

Danke hier nochmal speziell für die Mithilfe beim Aufbau des Projekts an Xander und den Verlängerungen der Projektabgaben.

BeOne

-Over and Out-

