Noroff
School of technology and digital media

Reflection Report

Exam Project 1

Daniel Vier

Word count

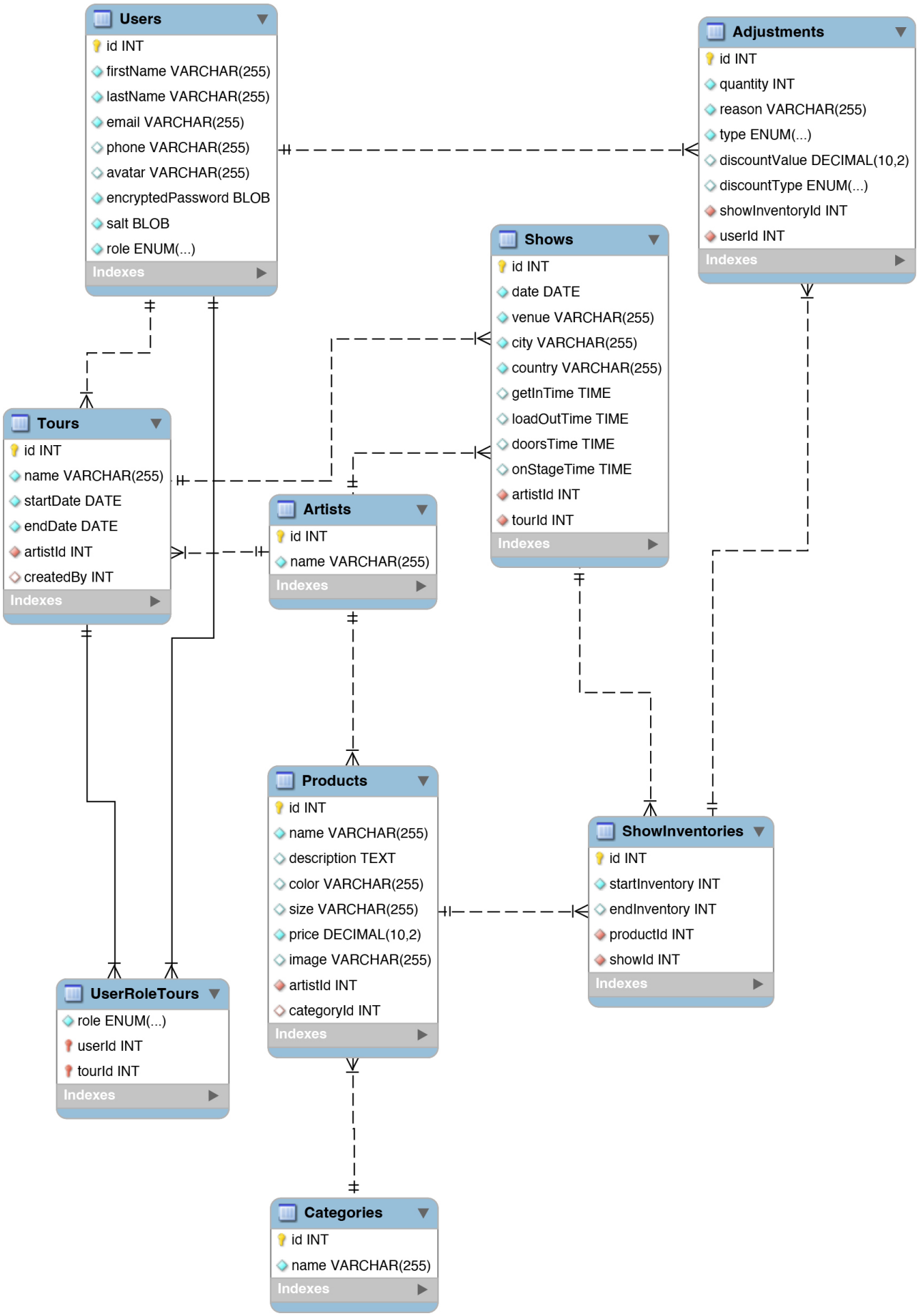
Main text: 1193

Table of Contents

1. Summary

The Merchandizer API is designed to manage band/artist merchandise inventory during tours, providing endpoints for CRUD operations on users, artists, shows, and merchandise. It ensures a single source of truth for inventory tracking across shows, making it easier to monitor stock levels and sales, moving away from the difficult excel sheets and formulas.

2. Database Design


Users
id INT
firstName VARCHAR(255)
lastName VARCHAR(255)
email VARCHAR(255)
phone VARCHAR(255)
avatar VARCHAR(255)
encryptedPassword BLOB
salt BLOB
role ENUM(...)
Indexes
Adjustments
id INT
quantity INT
reason VARCHAR(255)
type ENUM(...)
discountValue DECIMAL(10,2)
discountType ENUM(...)
showInventoryId INT
userId INT
Indexes
Shows
id INT
date DATE
venue VARCHAR(255)
city VARCHAR(255)
country VARCHAR(255)
getInTime TIME
loadOutTime TIME
doorsTime TIME
onStageTime TIME
artistId INT
tourId INT
Indexes
Tours
id INT
name VARCHAR(255)
startDate DATE
endDate DATE
artistId INT
createdBy INT
Indexes
Artists
id INT
name VARCHAR(255)
Indexes
Products
id INT
name VARCHAR(255)
description TEXT
color VARCHAR(255)
size VARCHAR(255)
price DECIMAL(10,2)
image VARCHAR(255)
artistId INT
categoryId INT
Indexes
ShowInventories
id INT
startInventory INT
endInventory INT
productId INT
showId INT
Indexes
UserRoleTours
role ENUM(...)
userId INT
tourId INT
Indexes
Categories
id INT
name VARCHAR(255)
Indexes

Overview of Relationships

- An Adjustment belongs to one ShowInventory; therefore, this is a many-to-one relationship.
- An Adjustment belongs to one Product; therefore, this is a many-to-one relationship.
- An Adjustment belongs to one User; therefore, this is a many-to-one relationship.
-
- An Artist has many Products; therefore, this is a one-to-many relationship.
- An Artist has many Tours; therefore, this is a one-to-many relationship.
- An Artist has many Shows; therefore, this is a one-to-many relationship.
-
- A Category has many Products; therefore, this is a one-to-many relationship.
-
- A Product belongs to one Artist; therefore, this is a many-to-one relationship.
- A Product belongs to one Category; therefore, this is a many-to-one relationship.
- Products are linked to many Shows through ShowInventories; therefore, this is a many-to-many relationship.
-
- A Show belongs to one Artist; therefore, this is a many-to-one relationship.
- A Show belongs to one Tour; therefore, this is a many-to-one relationship.

- Shows are linked to many Products through ShowInventories; therefore, this is a many-to-many relationship.
-
- A ShowInventory belongs to one Show; therefore, this is a many-to-one relationship.
- A ShowInventory belongs to one Product; therefore, this is a many-to-one relationship.
- A ShowInventory has many Adjustments; therefore, this is a one-to-many relationship.
-
- A Tour belongs to one Artist; therefore, this is a many-to-one relationship.
- A Tour has many Shows; therefore, this is a one-to-many relationship.
-
- A User belongs to one Role; therefore, this is a many-to-one relationship.
- A User is linked to many Tours through UserRoleTour; therefore, this is a many-to-many relationship.
-
- A UserRoleTour belongs to one User; therefore, this is a many-to-one relationship.
- A UserRoleTour belongs to one Tour; therefore, this is a many-to-one relationship.

2.2. Main section of report

I doubted myself very much, in a lot of different decisions, like 3NF database normalizations, where I chose to put tour roles and user roles as enums and not in their own database tables, as this will definitely not be added to.
Naming inventory (in singular, in comparison to the rest of the endpoints which are in plural) in routes and services but naming the model ShowInventory can make further additions like warehouse inventory and bus inventory more managable.
How to organize the routes was also challenging, but I decided to nest shows everything, except from artists, products and categories, which should be accessed by everyone using the app. If an artist changes a merchandise company, they should be able to easily find the products they need to set up a show inventory.
It made sense to nest inventory and adjustments under shows and shows under tours, which shows a clear relationship and «ownership». This also made authorization easier, using middleware focusing on the tour id.
Counting in before a show and counting out after a show made me doubt how to set up the inventory endpoints, and maybe it would have been a bit more user friendly to have a /inventory/start post endpoint for counting in and a /inventory/end put endpoint for counting out at the end of a show, but I think I made a decent choice that works for now.

I faced many challenges trying to finish this project, like working with decimals in sequelize, parameters in express that are strings when you need numbers, middleware to fix this and check that nested resources exist.

I have already created a «showscraper» which scrapes artists shows from songkick and returns them in json format, this will eventually be implemented in the frontend of this application. Speaking of frontend Im glad it was optional, because there would have been no chance that I would have been able to complete that in addition to the backend in 4 weeks, so it was nice to focus on creating a solid api system instead of using this time to create an easy user experience.

I used way to much time on testing, creating all resources for each test and trying to refine it to using global variables took a lot of time and reading of jest documentation, which was not very informative. But I managed to use as little requests for testing as I possibly could in the end.
I focused mostly on success responses and I could easily have tripled the tests if I wanted to test all edge cases, but I realized there was not time to focus on that.

I also regret not taking a TDD approach when first starting this project, as I have spent way too much time on manual testing in Postman. This together with sales calculations in the last sprint made way too little time for setting up swagger and writing this report. I had spent too much time creating nice Postman documentation, examples and a system Im very happy with and got obsessed trying to convert this data to swagger, which resulted in something that wasnt optimal, and in my honest opinion, using swagger-autogen makes way too much code in the routes, especially if you take into consideration all the edge cases and errors that can occur, which Im proud to say I have

made a solid system to handle all the edge cases I can think of.

I have also never worked with images in a database before and spent a lot of time creating endpoints with binary data, which slows down each request, so I finally set up a AWS S3 bucket instead and just added links to these in the database. This was very new and also took a lot of time.
Im also very happy with my role base permission system, which took a lot of trying and failing. Refactoring is also something I felt I could have done forever, creating more and more factory functions, but at some point I had to say stop.

2.3. Conclusion

While I appreciated getting my internship project approved, this was a very challenging task and at certain points, when being stuck, without much guidance or clear direction from the initiators of the project, I sometimes wish I had gone for the e-commerce project exam the rest of the class did.
But I really learned a lot from this and I am somewhat happy with the «end result». There is still a lot to do before this can become a proper tool, as it is meant to be.
The initiators still need there to be a frontend in react-native, which will be a steep learning curve. And there are also many functionalities I still want to implement like venue concession calculations, stand alone shows with authorization, soft deletes, more advanced product variations, exporting of reports in pdf format, connecting to webshops to add and remove inventories, connecting to

payment services to verify the actual sales numbers and how to resolve differences.

Lets hope I can finish the rest of the features the initiators require and that it actually lands me a proper job!

3. References

References listed in README.md like specified in Course Assignment Instructions.

This template cut the image of the Jira sprints, so it will be in its own image file in the Documentation folder.