Contents

[Scope 1](#_Toc64241844)

[Tech Stack Used 1](#_Toc64241845)

[Other Info 1](#_Toc64241846)

[Database Design 2](#_Toc64241847)

[Import Design Consideration 3](#_Toc64241848)

[Future Improvement (not implemented in current scope) 3](#_Toc64241849)

# Scope

1. Admin role with features:
2. Sign in/out as admin.
3. Add bank employees.
4. Delete employees.
5. Employee role with feature:
6. Sign in/out as an employee.
7. Create a customer.
8. Create accounts like savings, salary, loan, current account etc.
9. Link customers with accounts.
10. Update KYC for a customer.
11. Get details of a customer.
12. Delete customer.
13. Get account balance for an account.
14. Transfer money from one account to another.
15. Print Account statement of an account for a time range in pdf.
16. Calculate interest for the money annually (at 3.5% p.a.) and update the account balance.

# Tech Stack Used

Tools/Frameworks: Spring, Spring Boot, Spring security, Akka, MySql Database

API/Libraries: com.auth0 (for authentication/authorization), librepdf (for pdf generation)

# Other Info

Github Repo:

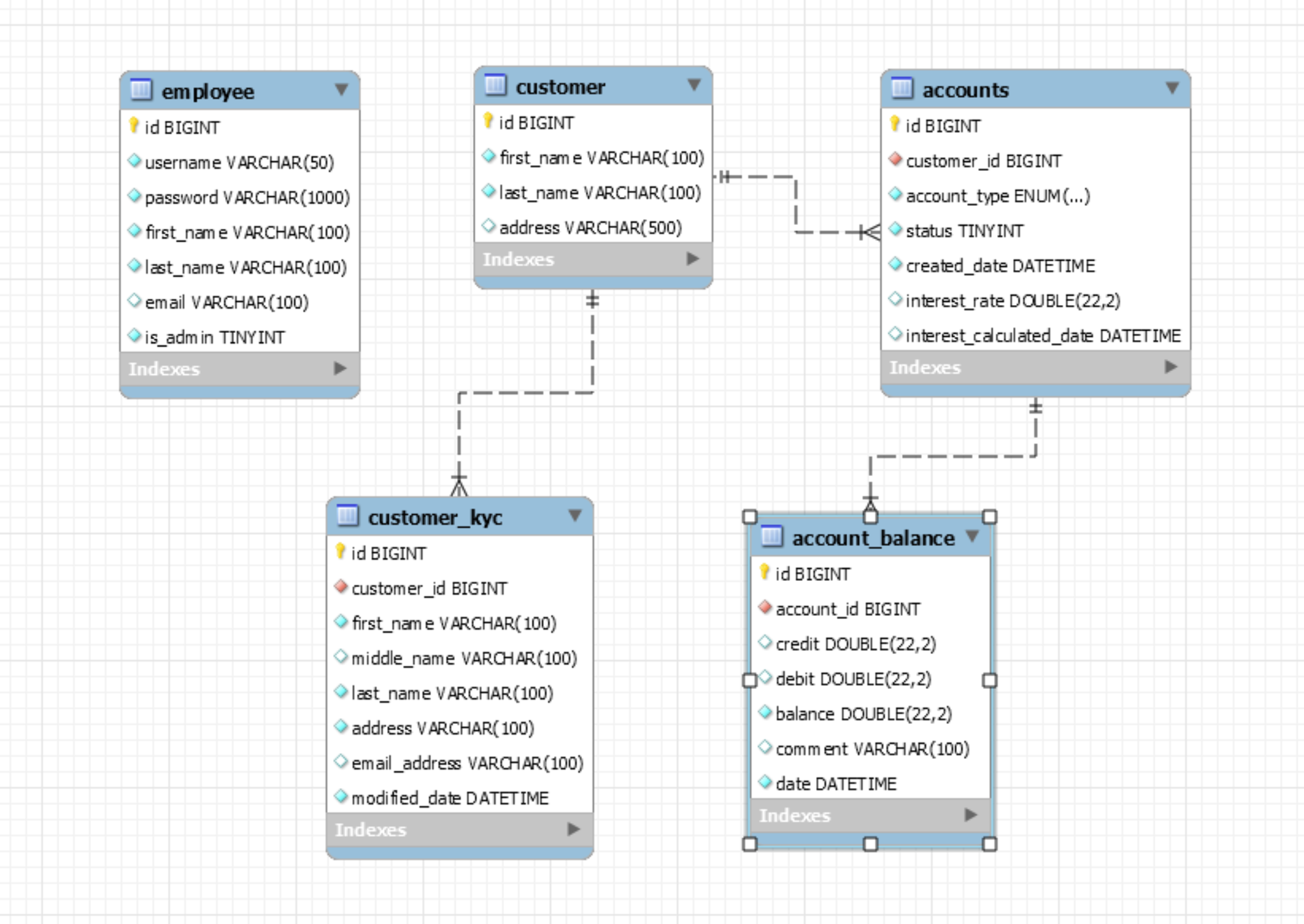
<https://github.com/ksoneji/BankApplication>

PostMan Collection: <https://github.com/ksoneji/BankApplication/blob/main/docs/BankApplication.postman_collection.json>

Database Dump:

https://github.com/ksoneji/BankApplication/blob/main/docs/database\_dump.sql

# Database Design



**Quick Overview**

* An employee can be an admin or non-admin. They are stored in the “employee” table.
* A customer can have multiple accounts of different types.
  + It is a one-many relationship between “customer” and “accounts” table. Account can be of type 'savings','salary','loan','current'.
  + Account can be active/inactive.
  + Interest related meta data (rate and last calculation date) are recorded in the account table.
  + Transactions for the account are recorded in the “account\_balance” table.
* There is 1:1 relationship between the “customer” and the “customer\_kyc” table.

**Future Data model Improvements (not in current scope)**

* Make account type more dynamic by having a different metadata table and referencing the account type id in the “accounts” table.
* Add a mechanism to archive/partition the records in the “account\_balance” table.

# Import Design Consideration

**Register Employee**

* User is non-admin by default. At present there is no API to change it to admin. It must be done via backend. Ideally, in real world application there would be a super admin who can setup an admin. An admin can thereafter setup other admins.

**Login**

* On successful authentication, it would return a JWT bearer token in the response header (“Authorization”). This header with the token must be included in other transactions to access the APIs.
* This token will have a default expiry period of 15 mins. The mechanism of refreshing the token beyond that interval was not included in this scope.

**Logout**

* Adds the token to a blacklist which is managed in a cache (ehcache). On authorization, the API will check if the incoming token is a part of a blacklist. If yes, then it would not allow the access. The token will evict from the cache in 15 mins since that’s the default expiry period of the token itself. Hence the cache size would not be large.

**Yearly Interest Calculation**

* Interest is calculated as simple interest. It is currently defaulted at 3.5% but can be overwritten while adding/updating an account using the REST API to add/update account.
* Used a background job that runs once a day Instead of using a REST API to calculate annual interest. The background job picks up all the ACTIVE accounts for which interest calculation is due and used Akka library to spawn separate threads to calculate the interest for each account.
* The JPA query used is a native query for better performance since there can be millions of records and filtering it on the app layer would have been non-performant. Its better to filter the records upfront and then bring it to the app layer for further processing.

# Future Improvement (not implemented in current scope)

**Tech Stack**

* Logger configuration can be taken out in a separate configuration file and have package wise settings, rotation settings, different log files for different packages etc.
* Few properties not related to Spring have been placed in the Application.properties. We should ideally put it into a separate configuration file.
* The Akka background job can be modified to spawn threads for a batch of accounts instead of one account for scalability purpose.
* Provide an additional refreshToken to continue the logged-in user interactions if it exceeds the token expiry period.

**Functionality**

* Transfer for account balance: We can put an additional check to verify if the accounts belong to the logged in user even if authentication/authorization is in place. We can also put additional checks that the accounts exist. For now, its assumed that the client will make the rest call only if both the accounts are valid. We also can query the last balance for the source and target account instead of expecting it from the payload.
* Currently, the user role is limited to Admin/Non-admin. We can change the authorization code and data model to handle multiple roles.