Discovery Vitality: Active Rewards Account Management System

CMPG323: Project 1

Mr. DE Roux

Project submitted for the degree *Bachelors* in Information Technology at the North-West University

Supervisor(s):	Dr. JT Janse van Rensburg
	Mr. Z Boonzaaier

Student number: 30196299

PREFACE

As project 1 of the CMPG 323 module of the North-West University, I was instructed to create a rather simple system for the backend of the Discovery Vitality Account system.

The project's aim was to teach me the entry-level technology stacks, or the like, that will most probably be used in my early years in the field of software engineering. The technologies involved was mostly java-related, but the experience is assured to overflow into any other stack.

The project itself was of great concern for most of my semester, as it offered great challenges and difficulties I am still not used to dealing with. Nonetheless, it was a wonderful learning experience, and sparked interest into learning more.

ABSTRACT

The system’s purpose is the management of a user account of the Discovery Vitality Rewards Program. The following is applicable:

- The user can see their available balances of their accounts.
- They can subtract their balances by using their currencies for rewards, coupons, gifts, and many more.
- The user can add currencies to their rewards balances by claiming rewards, lucky draws, and weekly rewards.

The system that was developed included many components. These components were built using a technology stack that was provided:

- Maven – as a build tool.
- Spring-boot.
- Swagger – for testing and documentation.
- Hibernate and JPA – for database connections, repositories and entities.
- MySql Database, and connector – for database server, creation and connections.
- MySQL Workbench – for database creation, visualization and connections.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 DESIGN

1.1 ERD diagram

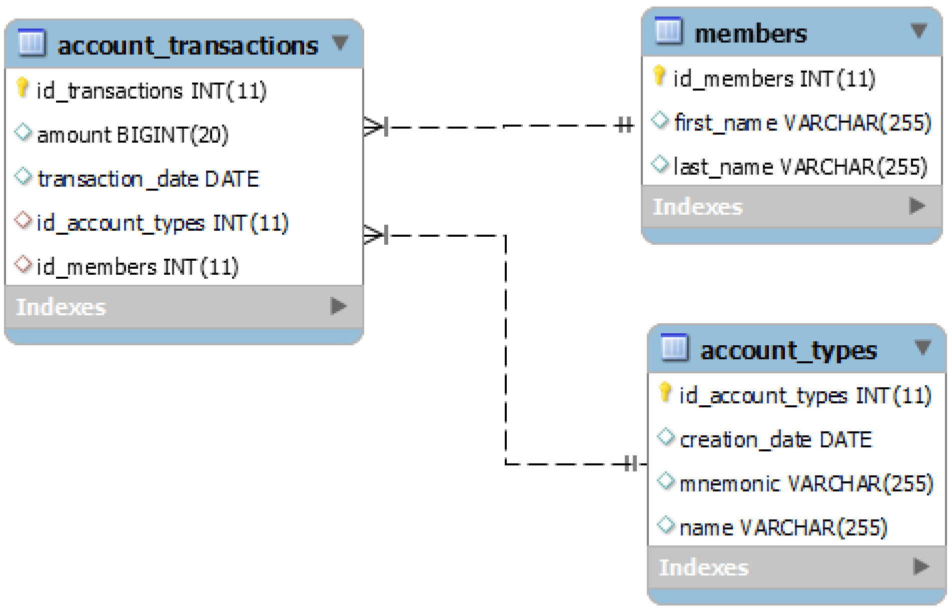


Figure 1:Figure showing the database ERD diagram

1.1.1 members

The members table represents the common users of the system, storing their first and last names, and auto generating a unique id for each row. This table will be the base of the database, as the other tables cannot exist without an existing entry in members.

1.1.2 account_types

The account_types table represents the different accounts that one member can have, storing a mnemonic value for different currencies, or even different rewards points and programs, as long as the basic structure is constant.

1.1.3 account_transactions

The account_transactions table represents a log of transactions made by users with their respective accounts. It acts as a balance calculation table in code as well.

1.2 Use case diagram

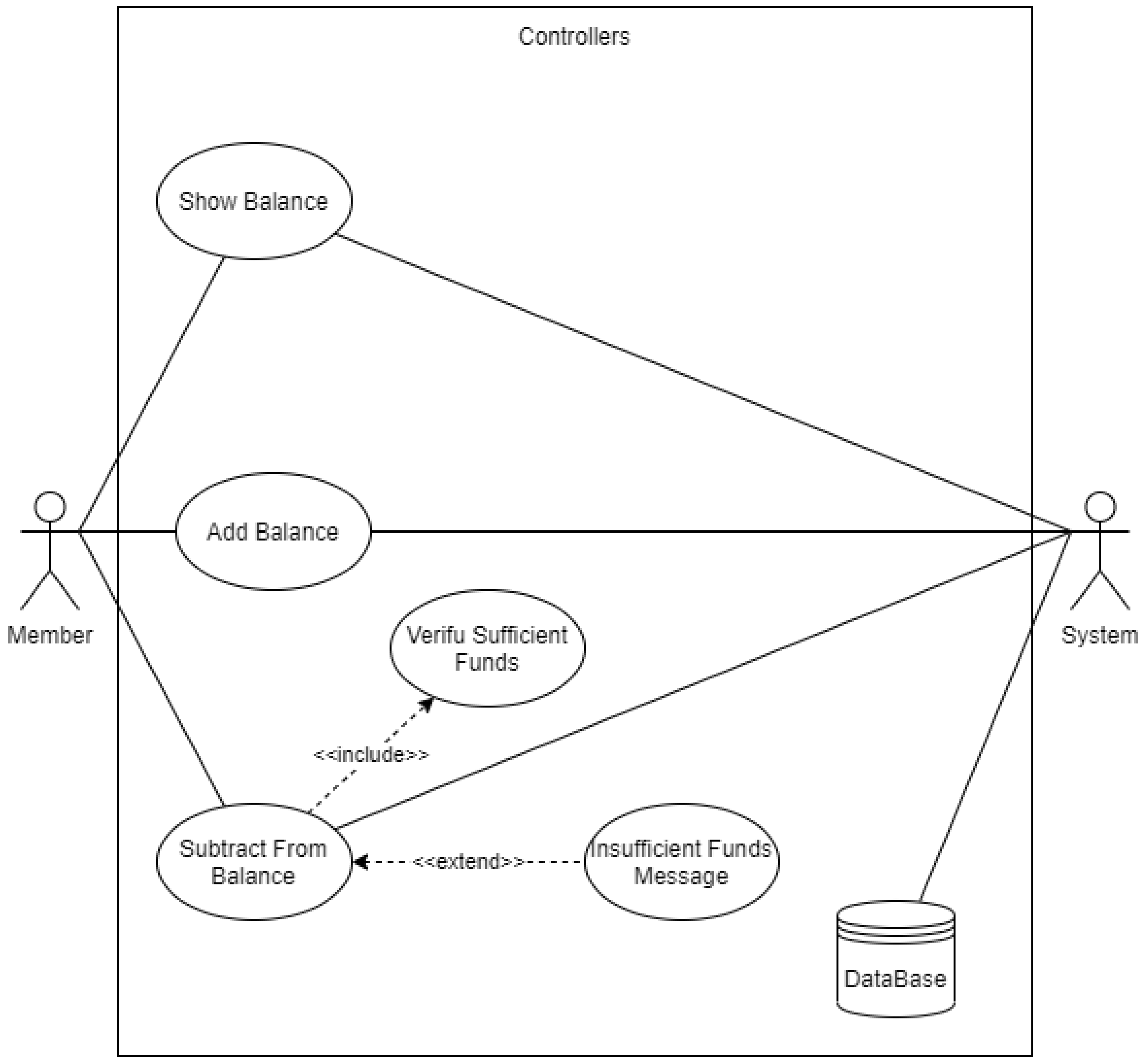
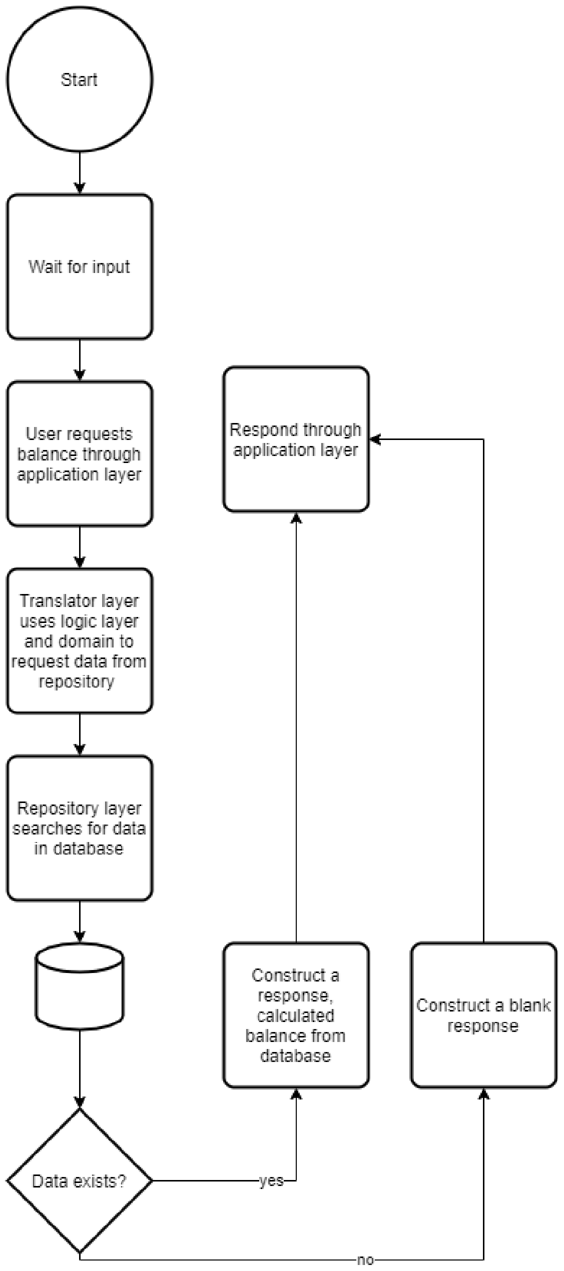


Figure 2: Figure showing the system's basic use case diagram

The system is used for very basic functionality, where a member can see their current balances, add points and subtract accordingly. Because of the application layering used in the code, this is a very basic representation of the product, but will be further fleshed out in the flow diagrams.

1.3 Flow diagrams

1.3.1 Show balance



Start
Wait for input
User requests balance through application layer
Translator layer uses logic layer and domain to request data from repository
Repository layer searches for data in database
Data exists?
yes
no
Construct a response, calculated balance from database
Construct a blank response
Respond through application layer

Figure 3:Figure showing the flow diagram of the balance showing service

1.3.2 Update (add/subract) currency

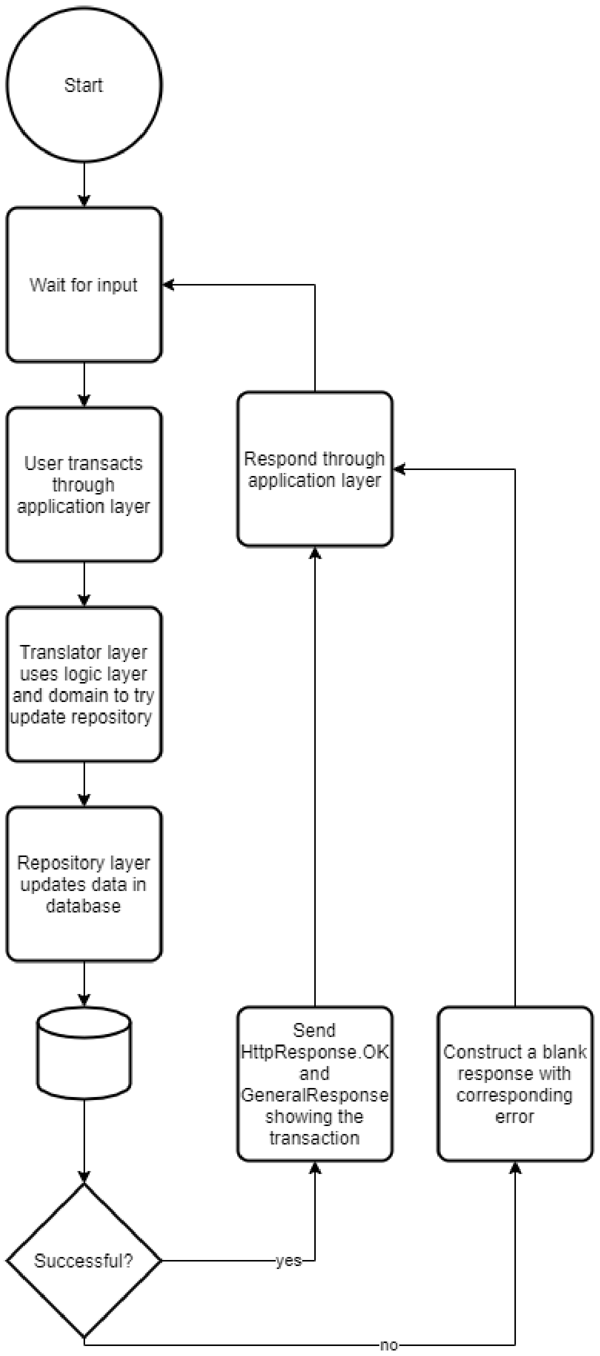

Start
Wait for input
User transacts through application layer
Translator layer uses logic layer and domain to try update repository
Repository layer updates data in database
Successful?
yes
no
Respond through application layer
Send HttpResponse.OK and GeneralResponse showing the transaction
Construct a blank response with corresponding error

Figure 4:Figure showing the update account / transaction service