

Possible examples include:

Purchase ordering system\*

Banking application\*

Collectable card game tracker\*

Employee management system

# **Project 0**

****Description**** Leveraging Java 8, create a CLI application with a multi-tier user system. You may choose the actual use case for your application as long as it meets all the requirements defined below.

Possible examples include:

* Banking application
* Employee management system
* Collectable card game tracker
* Purchase ordering system

****Requirements****

Technologies:

* + Java 8
    - Scanner (CLI inputs)
    - JDBC (Database Connectivity)
    - JUnit (Unit Testing)
    - Logback (Logging)
  + PostgreSQL
  + Maven (Dependency Management)

Functionalities:

Multi-tier Users - There must be at least three different types of user accounts each different functionality access.

Basic mathematic manipulation of the fields of an entity object.

User login with passwords and logout.

* + Passwords may not be stored in their raw form in the database. They must be encrypted somehow.
  + Stretch Goal: User registration.

Validation for invalid inputs.

Reasonable Logging of user interactions with the application.

Design:

* + Data persistence in a SQL database normalized to 3rd normal form.
    - Must contain at least three entity tables. (Join tables do not count.)
    - Must contain at least two relationships between tables.
      * Stretch Goal: Create a Many to Many relationship between tables.
    - Must contain at least one stored procedure.
  + Three layered architecture of the Java application.
    - Reasonable test coverage of the service layer.
      * Stretch Goal: Implement Test Driven Development.
  + DAO design pattern.

Other:

* + A SQL Script file that can be used to create a clone of your database structures.
  + Stretch Goal: A slide show presentation introducing your application before demoing it.

****Example Bank Application Requirements****

* Customers of the bank should be able to register with a username and password, and apply to open an account.
  + Stretch Goal: Customers should be able to apply for joint accounts
* Once the account is open, customers should be able to withdraw, deposit, and transfer funds between accounts
  + All basic validation should be done, such as trying to input negative amounts, overdrawing from accounts etc.
* Employees of the bank should be able to view all of their customers information. This includes:
  + Account information
  + Account balances
  + Personal information
* Employees should be able to approve/deny open applications for accounts
* Bank admins should be able to view and edit all accounts. This includes:
  + Approving/denying accounts
  + withdrawing, depositing, transferring from all accounts
  + canceling accounts

## **Evaluation**

The project will be evaluated out of 100 points split between three main catagories: 70 points for the functionality and design of your project, 25 points for the presentation of your project during the project showcase, and 5 points for the use of proper Scrumban practices. The evaluation will be further subdivided as follows:

****Project Score****(70 pts):

* 15 pts: Ability to persist meaningful data in the database and then retrieve it and display it to the console.
* 10 pts: Ability for the application to logically address the use-case for which it was designed.
* 10 pts: Ability to mathematically manipulate values stored in an entity object.
* 10 pts: Proper database schema achieving 3rd normal form. (E.g. Accounts have a proper relationship to their owning user.)
* 10 pts: Login, register, update and logout functionality for users.
* 5 pts: Proper use of DAO design pattern.
* 5 pts: Different user roles with different levels of access implemented correctly.
* 5 pts: Reasonable test coverage of the service layer and proper logging.

****Presentation Score****(25 pts):

* 15 pts: Clear, concise, logical and professional communication during the project presentation.
* 10 pts: Ability to communicate clear answers to fully address questions asked about the project.

****Agile Score****(5 pts):

* 5 pts: Kanban board and daily standup notes utilized and completed.

Banking App:

Driver:

getUser()

* First of all: new user or existing user?
  + If new user: register account, role, password, confirm pwd, balance, then proceed to login page
  + Existing: login()
* Login():
  + Input userID
    - Check if user exists,
      * If not, asks user to enter another one.
      * Existing user: Input password
        + Check pwd
        + If matches. Return user.
        + Else: exit program. / asks user to enter id again.

UserUI(user)

Shows UI for user role determined.

User can then: deposit, withdraw, getAccountInfo(), transaction log.

Clerk can check transaction log, ???

Admin: do everything the others can do, plus more. Like assign clerk role to customers. Or make clerks customers again.

FULL backlog:

User, admin, bank clerk.

User: userId, balance, deposit, withdraw, check account info, transaction log

deposit(): user money--, account money++, transaction log: add new log

transactionLog: obj contains userId, clerk id, date&time, amount of money in or out, comment.

withdraw(): user money++, account money--, transactionlog add new withdraw log.

checkInfo(userId, account number): get balance, list transactionlog.

Clerk: clerkId, getAccountInfo(userId, account number), createsAccounts and CancelAccounts

admin: have all the functions, plus

Purchase system backlog:

list all available products with product id with prices

purchase with id and $account, seller get money from customer

product: product id, name, price, stock.

buyer: id, money, role: buyer

buyer can spend money to change stock, and seller's money

purchase(itemId, quantity of purchase):

buyer money - item price \* quantity of purchase; seller money + item price \* quantity of purchase; stock - quantity of purchase;

print receipt: seller money amount, quantity of purchase, current stock, money spent, tracking number, transactionID

if buyer not enough money: throw exception/catch and fail purchase.

if out of stock: throw out of stock message.

seller: id, money, role: seller

seller can change item price, stock, name, id

can add new item, remove existing item.

admin: can change everything.

given transaction ID can find a purchase, which contains

transaction: contains list of itemids purchased, their quantity, money spent and tracking number, as well as buyer id seller id.