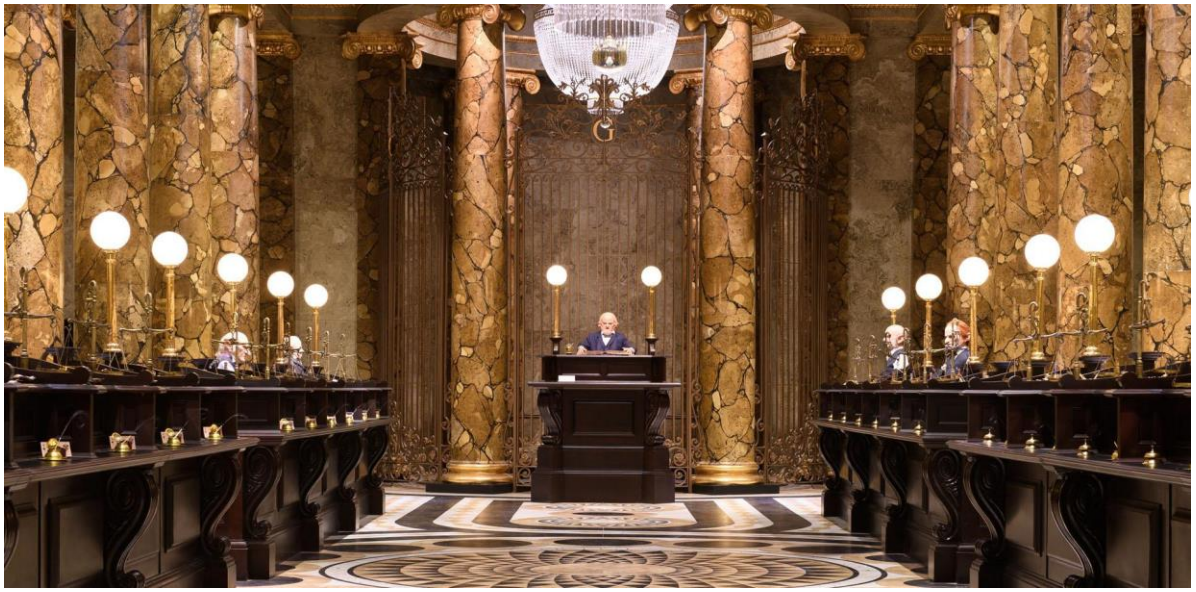
WIA1002 DATA STRUCTURE
SEMESTER 2, SESSION 2023/2024

GROUP PROJECT
TOPIC 1: E-GRINGOTTS



Introduction

Digital banks, such as GX Bank, MoneyLion, Quontic, et cetera, much like their traditional counterparts, offer a safe haven for our hard-earned fortunes. They have emerged as catalysts for change, reshaping the traditional landscape of finance and offering unparalleled convenience, accessibility, and security to users worldwide. With the ability to manage finances at their fingertips, patrons are liberated from the constraints of physical branches and limited banking hours, empowered to conduct transactions, monitor accounts, and access financial services anytime, anywhere. Moreover, digital banks have democratized banking, extending financial services to underserved populations and prioritizing the protection of user data through advanced encryption and fraud detection measures. As we delve into the enchanted realm of E-Gringotts, a digital bank inspired by the wizarding world of Abraxas, we explore the principles of digital banking intertwine with the magic of fiction, creating a truly transformative banking experience for witches, wizards, and muggles alike.

Problem Statement

Within the enchanted corridors of Diagon Alley and beyond, the absence of a digital banking solution tailored to the needs of wizards and witches has been keenly felt. Traditional methods of financial management, reliant on cumbersome parchment and quills, no longer suffice in an age where the magical community finds itself increasingly intertwined with the muggle world.

Gringotts, as the most prominent bank in the wizarding world, seeks to address this pressing need by providing a digital banking platform that seamlessly integrates with the wizarding world's unique currency system while offering the flexibility to transact in muggle currencies. Therefore, E-Gringotts is introduced, where patrons can embark

on their daily transactions, from purchasing potions in Diagon Alley to booking trips on the Abraxes Express, with the ease and convenience afforded by modern technology. What's more interesting is that the magical community could convert the dynamic magical currency to get exchange rates. The ruler of Diagon Alley has asked to create such an application as they are missing citizens who are familiar with computing technologies. You are tasked with developing E-Gringotts with your computer science and data structure skills for it to be efficient and faster.

Note: You are free to use any build style for developing E-Gringotts. There is no strict requirement for how many classes/methods there should be. You are free to develop it in your own style. However, the whole app must work as intended with the major requirements specified. You will be evaluated on examples that test your execution time efficiency, i.e., whether you have used the most efficient data structure to solve a particular problem, and by looking at the underlying code. Hence, this whole project is left open-ended for you to explore and push yourself towards more challenging ideas.

Basic Requirement (12%)

a. Account Creation, Transactions & User Types (3%):

Create a generic class for account creation (Example class: **Account**) of magical users holding their meta-data (*userId, username, password, DOB, address,* etc.). Be creative, explore more of what meta-data is required in account creation, and include it in your implementation. For example, you can create a class called **Card** for holding information about the credit/debit cards of a user (*cardNum, cVV, expiryDate, userId, etc.*). Furthermore, you can include the avatar of the user if you are going to make a UI (Example class: **UserAvatar** for storing *imagePath, userId,* etc.). The bottom line is that users must have an account to register and log in to the E-Gringotts app.

Furthermore, since the application is about having money transactions, you must create a class to hold transaction data for a particular user. For instance, the class can be called **Transaction** and instance variables can be *userId, amount, balance, dateOfTrans, category, etc.*

Additionally, develop a generic class to hold different types of users. Your class must take into account all the necessary variables that are sufficient for E-Gringotts to work. Specifically, there are 4 types of magical users defined below. You must make use of generics and the fundamentals of programming to build classes for each type of magical user, keeping in mind to extend your blueprint classes in case a new type of user emerges from the depths of Diagon Alley. The user types are as follows:

a) "**Platinum Patronus**" for top-tier users
b) "**Silver Snitch**" for mid-tier users
c) "**Golden Galleon**" for high-tier users

Meanwhile, we also should have an admin dashboard that has some extra features compared to normal users. For example, the total number of users, the number of transactions per day, some statistical data, etc. This feature should be hidden from the normal user view if you are logged in as a normal user. However, if you are logged in as an admin, then such features must be displayed. For reference, use the "**Goblin**" name for administrators.

b. Pensieve Past (Transaction History - 1%):

The magical users of E-Gringotts should be able to see their past transactions in the order in which the transactions were executed, i.e. the recent ones are on top. With the help of appropriate data structure, create a class called **Pensieve Past** (can be called whatever you like) that keeps track of transaction history for a particular user. Hence, when a user wants to see their transaction history, they can either select from the menu (in case of CLI-based output) or click on a button (in case of UI-based output).

c. Sorting Hat Select (Transfer filter – 1%):

E-Gringotts implements a robust filtration feature essential for magical users to efficiently refine their transaction searches. Users can apply filters based on specific criteria such as *date, amount threshold, or categorical type* (e.g., Food, Grocery, Entertainment). Importantly, the filtration process is optimized to run more efficiently than O(n^2) time complexity, ensuring swift retrieval of filtered results without excessive computational overhead. This functionality empowers magical users to effortlessly navigate and analyze their transaction history, enhancing their overall E-Gringotts experience.

d. Marauder's Map (Find friends for transfer based on phone or name – 1%):

As there are multiple people to whom the users can transfer or make transactions, it must be convenient for them to search for a particular fellow magician in their world quickly and send them some bucks. To make it efficient, you are required to implement a search functionality where magicians can search their friends based on name, contact number, or any other field that you desire. Apply an appropriate search strategy to make it quicker. Hint: Your search algorithm must run in better than O(n) time complexity.

After the completion of a transaction between magicians, the system automatically generates a receipt to document the specifics of the transaction. This receipt encompasses crucial details such as the amount transferred, the date and time of the transaction, and relevant participant information including their names and contact details. Additionally, a unique transaction ID may be included for tracking purposes. The receipt serves as a formal record of the transaction, providing transparency and accountability to both parties involved. It acts as evidence of the transaction and can be referred to for future reference or dispute resolution if needed, thereby ensuring a reliable and organized transaction process within the magician community.

```
🌟 E-GRINGOTTS RECEIPT 🌟

Transaction ID: **TRX123456**
Date: March 24, 2024

From: Harry Potter
To: Hermione Granger
Amount: **50.00 Knut**

Thank you for using E-Gringotts! Your magical transfer has been successfully completed. 🪄✨

For any inquiries or further assistance, owl us at support@egringotts.com.

May your galleons multiply like Fizzing Whizbees! 🪙🌠
```

e. Gringotts Exchange (Currency conversion – 3%):

Realm	Currency	1 Knut (K)	1 Sickle (S)	1 Galleon (G)
Wizarding World	Knut	-	29	493
	Sickle	0.03448	-	17
	Galleon	0.002028	0.05882	-

Gringotts exchange booth offers the exchange of many currencies. These currencies can change, i.e., they are dynamic in nature (new ones can be added). Implement functionality in the application where users can input a bunch of currencies and their conversion values. They will then input *from* and *to* currencies along with their values to get the desired exchange based on the list of currencies provided before.
Hint: Use a graph.

a. Currency input template = [[c1, c2, value, processingFee1], [c2, c3, value, processingFee2], […], …]
b. Example input: currencies = [[‘Knut’, ‘Sickle’, 29, 0.01], [‘Sickle’, ‘Galleon’, 17, 0.2]], from=’Knut’, to=‘Galleon’’, valueExchange=100
c. Example output: 100 Knuts = 49300 Galleons, processing fee to charge = 100*0.01 + 100*0.2 = 21 knut

Ensure to update the balance in the user's account after deducting the processing fee. Include all relevant details about the transaction in the receipt, such as the updated balance, the amount paid, and the amount received after conversion. This receipt serves as a formal record of the transaction and provides users with a clear overview of their financial exchange.

f. Divination Data (Analytics of expenditure through category – 2%):

Humans are visual creatures; it makes sense when things are ordered or categorised properly. Hence, implement functionality within the application to allow users to view their expenditures categorized by various categories such as Food, Entertainment, Grocery, etc. It's essential to organize transactions systematically for better comprehension. The implementation should ensure that transactions are **hashable** for efficient processing.

The final output should display the percentage of expenditure for each category alongside the total expenses incurred. Additionally, it's advisable to include visual aids such as pie charts for enhanced visualization. Furthermore, the application should support advanced filtration options to facilitate analytics based on categories, as well as daily, monthly, and payment methods (credit cards, debit cards, etc.).

g. Hogwarts Library (Usage of Database – 1%):

As you are developing an application, you will be required to store the information somewhere so that when the application loads, the information about its users stays persistent and consistent across any number of runs. Hence, you may implement the whole app on **MySQL storage**, **.txt files** or **.csv files**. It totally depends on you. The main purpose of this project is to learn how a software or application works while keeping in mind its efficiency (which you get from data structures).

Extra Features (4%)

a. Enchanted Interface (Graphical User Interface, GUI):

Creating an engaging user interface is crucial for any application. JavaFX is a great choice for building a GUI for E-Gringotts. It allows you to create a rich and interactive interface with various controls like buttons, text fields, and tables. You can design the interface to resemble the magical world of Abraxas, using thematic elements such as ancient scrolls for the background or wand-like cursors.

b. Gringotts Guard Key (Integrate Security Pin):

To enhance security and efficiency, integrating a security PIN system is a smart move. Unlike OTPs, a PIN is immediate and user-friendly. You can implement a PIN verification step right after the user logs in or before confirming transactions. Ensure that the PIN is encrypted and securely stored.

c. Fidelius Charm (Password Salting):

Storing passwords in plain text is a significant security risk. Salting passwords before hashing them adds an extra layer of security. A salt is a random string added to the password before it's hashed. This means that even if two users have the same password, their hashes will be different due to the unique salts.

d. Gringotts Glimpse (Email Notification):

Email notifications keep users informed about their account activities. You can send emails for various triggers, like successful transactions, logging in from a new device, or reaching a savings goal. Use an email service API to send automated emails from your application.

e. Owl Post Help (AI Chat Support):

Embark on a seamless banking journey with E-Gringotts' Owl Post Help, our innovative AI chat support feature. Imagine interacting with a knowledgeable owl, ready to assist you with any banking query you might have, day or night. This feature harnesses the power of Llama 2 (open source LLM model powered by Meta) with retrieval augmented generation (RAG), a state-of-the-art natural language processing technology, to provide you with precise and context-aware answers. The Owl Post Help is not just a service; it's your personal banking companion, always there to guide you through the magical world of E-Gringotts. For developers looking to integrate this feature, an API will be provided in the reference section, ensuring a smooth incorporation into your platform.

Tips & Comments

1. Use Git to do version control, GitHub (Git cheat sheet provided below)
2. Try to push your boundaries beyond JavaFX and enhance your real-world expertise by implementing the following: (optional)
    a. Front-end development
        i. Website
            - ReactJs (JavaScript-based framework)
            - VueJs (JavaScript-based framework)
            - NextJs (JavaScript-based framework)
        ii. Mobile Application
            - Android Studio (Java/Kotlin-based framework)
            - Flutter (Dart-SDK)
            - React Native (JavaScript framework)
    b. Back-end development
        i. Spring Boot (Java-based framework) !!! Only this framework allowed
    c. Database
        i. Firebase (NoSQL)
        ii. MongoDB (NoSQL)
        iii. MySql (SQL)
        iv. Postgres (SQL)
    d. UI/UX Design platform
        i. Figma (use 'explore community' templates to design your UI/UX faster)
3. All team members should use a consistent IDE to avoid versioning issues.
4. Explore a powerful AI tool, Github Copilot (Sign up using student email for free) for development.
5. Ask questions and clarify doubts.
6. Be creative! Don't limit your imagination.

References

1. Git Cheat Sheet
   https://education.github.com/git-cheat-sheet-education.pdf

2. Git Branching Practice (Important)
   https://learngitbranching.js.org

3. Full Stack Development with Java Spring Boot, React, and MongoDB tutorial
   https://youtu.be/5PdEmeopJVQ?si=Yq6oMU3EHBpaXjFn

4. Github Copilot
   https://education.github.com/pack

5. Personalised AI Chatbot Tutorial
   https://github.com/Poo-wei-chien/personalisedChatbot

Contact Information

Amaan Izhar
Email: 22086758@siswa.um.edu.my
Phone Number: 014-7154316

Poo Wei Chien (Rain)
Email: u2102771@siswa.um.edu.my
Phone Number: 013-9580338