Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Programowanie Komputerów 4

Aplikacja bankowa

autorzy	Tomasz Knura, Stanisław Czembor
prowadzący	mgr inż. Krzysztof Pasterak
rok akademicki	2019/2020
kierunek	informatyka
rodzaj studiów	SSI
semestr	4
termin laboratorium	czwartek, 10:15 – 11:45
sekcja	2
termin oddania sprawozdania	2020-06-16

1 Temat

Uproszczony model aplikacji bankowej. Symuluje działanie banku oraz część jego funkcjonalności.

2 Analiza zadania

Analiza zadania powstała przed rozpoczęciem pisania projektu. Zaczeliśmy od storzenia diagramu klas oraz szkiców interfejsu graficznego. Na tym etapie skupiliśmy się na głównych klasach i ich wzajemnej współpracy. Podzieliliśmy cały interfejs graficzny na ekrany dla użytkownika oraz administratora. Najlepiej naszym potrzebom odpowiadał dobrze współpracujący z C++ QML, część frameworku QT. Zdecydowaliśmy się na pracę z plikami JSON.

2.1 Struktury danych

W programie korzystamy z kontenerów biblioteki standardowej C++:

1. **unordered_map** - przechowujemy w niej obiekty klasy Account, kluczem jest numer konta. Zdecydowaliśmy się na mapę typu unordered ponieważ wartości naszych kluczy nie posiadaja żadnej kolejności,

2. **multimap** - przechowujemy w niej fundusze pod kluczem id użytkownika oraz karty pod kluczem numer konta. Zdecydowaliśmy się na ten typ mapy ponieważ pod jednym kluczem może znajdować się wiele obiektów(Do jednego konta może być podpięte wiele kart),

3. **list** - kontener używamy w wielu miejsach programu. Trzymamy w nim numery kont aktualnie zalogowanego użytkownika(klucze) oraz jego przyjaciół,

4. **tuple** - używamy go do połeczenia obiektu klasy loginData oraz saltu używanego do hashowania haseł. Umożliwia on stworzenie pary z obiektów różnych typów.

5. **vector** - używamy go do przechowywania lokalnych zmiennych w funkcjach.

2.2 Biblioteki i technologie

1. **QML** - język programowania umożliwiający tworzenie rozbudowanych interfejsów graficznych. Zdecydowaliśmy się na niego ze względu na dobrą interakcje z C++ oraz rozbudowaną dokumentację,

2. **nlohmann/json** - biblioteka umożliwiająca w łatwy sposób serializacje i deserializacją plików JSON. Ułatwia nam obsługę plików przechowujących dane z programu.

3. **framework QT** - wiele klas ułatwiających i umożliwiających rozwijanie programu na dwóch różnych platformach. Między innymi używamy klas pozwalających na określanie ścieżek do konkretnych folderów niezależnie od platformy i klasy odpowiedzialnej za hashowanie haseł.

2.3 Kluczowe klasy

Kluczowe klasy programu oraz ich zadania:

1. **User** - dziedziczy po klasie LogInData Klasa odpowiedzialna za przechowywanie unikalnych danych użytkownika takich jak lista kont (w postaci Stringow po których będzie przeszukiwana mapa w klasie Bank) czy lista przyjaciół (lista obiektów typu PaymentRetriever),

2. **Account** - klasa, bazowa dla wszystkich rodzajów kont. Obiekty tej klasy będą przechowywane w mapie w klasie Bank oraz w liście w klasie User.

3. **Bank** - klasa nie dziedziczy i nie jest dziedziczona. Zawiera pole User oraz przechowuje mapę wszystkich kont z pliku JSON,

4. **JsonManager** - klasa odpowiedzialna za obsługę plików JSON. Współpracuje z klasą Bank,

5. **Payment** - klasa obsługuje wszystkie przelewy w banku,

6. **Card** - klasa bazowa dla wszystkich rodzajów kart,

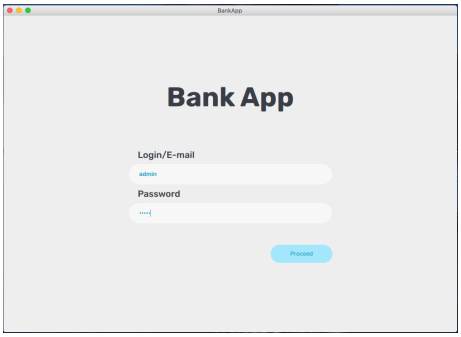
7. **Fund** - klasa bazowa dla wszystkich rodzajów fundusz,

8. **History** - klasa przechowująca listę wszystkich płatności. Jest kluczowa przy wyświetlaniu historii transakcji użytkownika. Klasa udostępnia publiczne metody Sort(), pozwalające posortować listę według różnych kryterium,

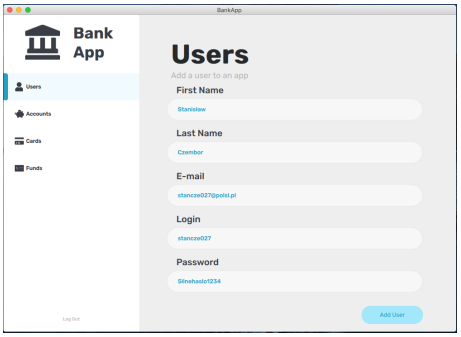
9. **Config** - klasa wczytuję nazwy plików oraz inne niezbędne dane z pliku config.txt.

3 Specyfikacja zewnętrzna

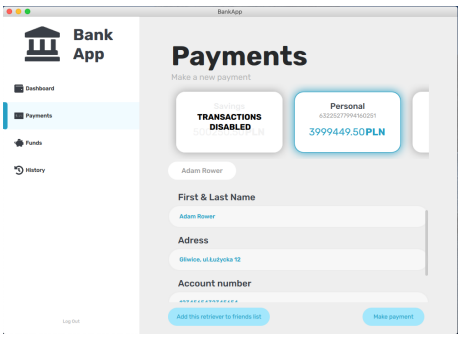
Po uruchomieniu programu wyświetla się ekran logowania Jeżeli zalogujemy się jako administrator mamy do dyspozycji ekrany umożliwiające dodawanie nowych użytkowników oraz dodawanie kont, kart i funduszy już istniejącym. W przypadku logowania jako użytkownik program udostępnia nam ekrany umożliwiające wykonywanie przelewów oraz podgląd historii.



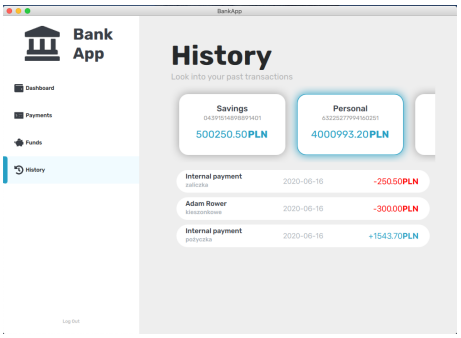
(a) logowanie



(b) dodawanie uzytkownika



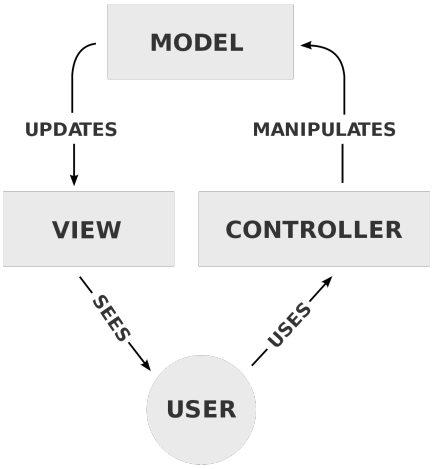
(a) przelewy



(b) historia

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem programowania obiektowego. Zarówno podczas projektowania jak i implementacji staraliśmy się realizować wzorzec projektowy MVC (model-view-controller).



4.1 Pliki

1. **LogInData.txt** - plik przechowuję zaszyfrowane dane logowania w formacie: login haslo id np. superman trudnehaslo22 235865,

2. **AccountData.json** - przechowuje dane dotyczące kont oraz kart. Głównymi obiektami w pliku są konta,

3. **FriendsData.json** - plik przechowuje dane dotyczące listy znajomych użytkownika. Pogrupowane według ID użytkownika,

4. **FundsData.json** - plik z danymi dotyczącymi funduszy. Pogrupowany funduszami,

5. **ConfigFile.txt** - plik z nazwami i ścieżkami do wszystkich plików.

4.2 Techniki obiektowe

1. Dziedziczenie

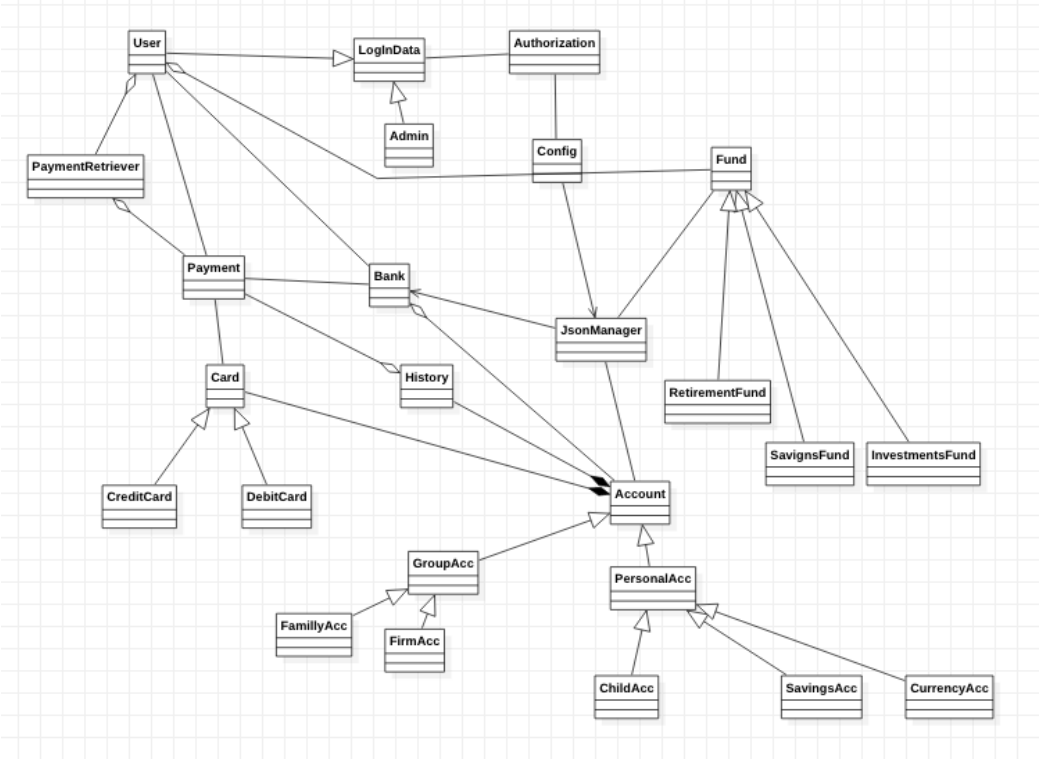
2. Przeciążanie operatorów

3. Interfejsy
4. Polimorfizm
5. Enkapsulacj

4.3 Wykorzystane zagadnienia z zajęć laboratoryjnych

1. Regex
2. Inteligentne wskaźniki
3. RTTI
4. Kontenery STL
5. Algorytmy i iteratory STL
6. Mechanizm wyjątków

4.4 Uprosczony diagram klas



5 Testowanie

W trakcie implementowania kolejnych klas i funkcjonalności moduły te były na bieżąco testowane w izolacji od reszty. Największa ilość błędów pojawiła się po rozpoczęciu testów całości, oraz przy podpinaniu UI do modelu.

1. **Serializacja obiektów do plików JSON** - biblioteka nholmann działała poprawnie pod systemem MacOS, lecz przy sieralizacji danych w Windowsie program kończył się przy próbie tworzenia obiektu JSON. Po wielu godzinach pracy nad problemem okazało się, że inicjalizacja obiektu JSON klamrami {} nie działa na windowsie poprawnie. Przypuszczamy, że problemem był kompilator nieobsługujący najnowszych wersji C++.

2. **Tworzenie modelów QML przy użyciu C++** - klasa modelu stworzona w opraciu o dokumentację QT nie była widziana w QML. Po wielu godzianch spędzonych na różnych forach okazało się, że dokumentacja jest nieaktualna.

3. **Inne ściezki do plików** - podczas pracy nad klasami obsługującymi pliki okazało się, że ścieżki do plików na Windowsie i MacOS są zupełnie inne. Problem rozwiązalismy przy użyciu klasy QPath z QT.

4. **Problem z wczytywaniem plików** - po spięciu modelu z z UI okazało się, że program przy próbie ładowania danych z plików crashuje się. Bardzo długo byliśmy przekonani, że problem powoduje zła obsługa plików. Okazało się, że powodem była metoda napisana na samym początku powtawania projektu, która za każdym razem podowała ustawianie id użytkownika na -1.

6 Wnioski

Tworzenie projektu w zespole i na różnych platformach uświadomiło nas jak trudne może być tworzenie oprogramowania przenoszonego na różne systemy. Bardzo ważna okazała się organizacja pracy, komunikacja oraz współpraca. Narzędziem niezbędnym był system kontroli wersji GIT. Przez objętość projektu trudne było usuwaniu błędów, które pojawiły się na końcowym etapie. Poznaliśmy język programowania QML oraz przećwiczylismy teamty z laboratorium.

7 Link do repozytorium

https://github.com/tknura/BankApp