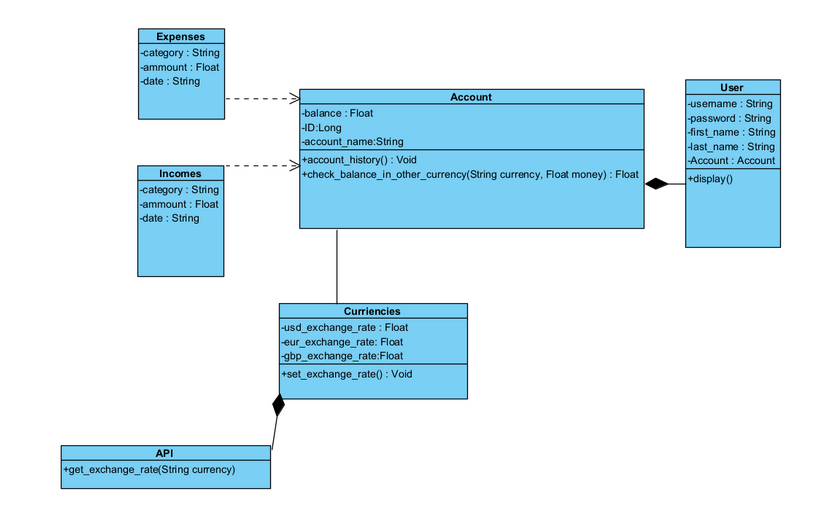
Diagram klas projektu



SOLID

1**.”Single responsibility”**

W naszym diagramie klas jest zachowana pierwsza zasada solid „Single responsibility”.

Każda z naszych klas jest odpowiedzialna za konkretną rzecz tj. klasy Expenses i Incomes są rozdzielone zamiast tworzyć jednej klasy dotyczącej wydatków i przychodów. Tym sposobem dane przechowywane w klasach są bardziej czytelne i zapewniany jest do nich łatwiejszy dostęp.

SOLID

2**.”Open/Closed”**

W naszych klasach brakuje zastosowania polimorfizmu który umożliwiałby nam modyfikowanie metod wirtualnych w czasie wykonywania programu. Bez używania polimorfizmu, musielibyśmy już na etapie pisania programu, wiedzieć jak będzie się on zachowywał. To za sprawą wczesnego wiązania, które musi być dostarczone kompilatorowi w momencie kompilacji i linkowania.  
W przypadku użycia polimorfizmu dostajemy nieograniczone możliwości projektowania aplikacji, gdzie zachowanie programu może się ciągle zmieniać. Wtedy klasy moglibyśmy modyfikować bez ich edycji. Po wprowadzeniu klas abstrakcyjnych powinniśmy zadbać o zastosowanie się do zasad **„Interface segregation”** jak **i „Liskov substitution”**,które to są do siebie zbliżone. Należy uważać by nie złamać ich np. stosując dziedziczenia bez użycia mechanizmu polimorfizmu. Nasze klasy bazowe powinny być jak najbardziej rozbite (bez zawierania w nich bardzo dużej ilości metod).

3.Zasada **DRY** (Dont’t repeat yourself)

W naszym programach powinniśmy usunąć fragmenty kodu które powtarzają się tak jak np. różne definicje tej samej klasy Account. Dzięki temu gdy konieczne będzie zastosowanie jakiejś zmiany w kodzie będziemy mogli zastosować ją w jednym miejscu. Również zwiększona zostanie czytelność naszego kodu.