Contract Framework:

Diagram

Description automatically generated

**Loan Application Process:**​

* Students and investors can call the "**createInvestor**" & "**createStudent**" functions to create their accounts with 0 balance;​
* Students can call "**createApplication**" function from StudentLoan contract to apply for their loans, with credit amount, interest rate and risk rating as inputs for a student with no ongoing loan and ongoing application, which will return the application indexed by "numApplication" (which is the application id);​

**Investment Process:**​

* Investors can call the "**grantLoan**" function, if investor account balance is higher than the amount invested and with no ongoing investment, record the counter "numLoans", application id(appID), timestamp, etc to this loan and complete the amount transfer; ​
* Students can call the "**countLoans**" function from StudentLoan contract to get the number of loans they  applied;​
* Students and Investors can call the "**findLoans**" & "**invFindLoans**" function to check the loan\_id in which they involved;​

**Loan Usage:**​

* InvestorToken contract imports the studentLoan contract, so Investors can use "**viewBalanceInLoan**" function from InvestorToken contract to use the viewBalance function in StudentContract to check their balances, and call "**depositToLoan**" function to increase the investments;​
* Student can use "**withdraw**" function from StudentLoan contract to use the credit amount inside the loan secured by "noReentrancy" modifier;​
* "**getApplicationData**" function returns the index of the application, credit amount, interest rate, risk rating and student information for checking;​
* "**getLoanData**" function returns loan id, interest rate, principal amount, original amount, amount paid, start time, monthly checkpoint, loan application id, and associated student   and investor information for checking;​
* Student can call the "**repayLoan**" function from StudentLoan contract if the their account balance is sufficient to repay  the some amount of the loan. If finally, the principal amount is reduced to 0, the loan will be closed;​
* Investor can call "**withdraw**" function secured by "noReentrancy" on the basis of "**withdrawFromLoan**" function modifier from InvestorToken contract to withdraw amounts from the loan and from account balances because it requires that the withdraw amount is smaller than the investor account balance;​

**Loan Trading:**​

* Investor can call "**sell**" function with ask\_price as input in InvestorToken contract, this function utilise "**withdrawFromLoan**" function to reduce the amount equalling with ask price from the loan; Investor can call "**buy**" function once there's an ask on the market in  InvestorToken contract; ​

Web3 and Contracts interactions:

* Generate Student Loan as "**StuLoan**"and Investor Token as "**InvToken**" contracts by "web3.eth.contract()" function with student loan address(StuLoanAddress), investor token address(InvTokenAddress) and corresponding ABI from remix in  local machine;​
* Generate the first three accounts: Student; Investor and Buyer by web3.eth.accounts[]; ​
* Uploading the students' names; credit amounts, interest rates and risk ratings generated via Machine Learning Algorithm as a dataframe named as "data" for next operations;​
* Define "**apply()**" function with data and the student name as inputs, call "createStudent()" and "createApplication()" functions via StuLoan to finish the loan application process;​
* Define "**invest()**" function to complete the grant loan process: for all the displayed applications interacted with the "getApplicationData()" function in InvToken, the investor enter the id of the student(appId), investor's name and the amount he or she wants to invest, then the "invest" function calls "createInvestor()", "depositToLoan()" and "grantLoan()" in InvToken to finish the process;​
* Define "**stu\_withdraw()**" function, first utilise "viewBalance()" in StuLoan and then input the amount the student wants to withdraw and call the "withdraw()" in StuLoan;​
* Define "**get\_loan\_data()**" function, if the operator is the student then call "getLoanData()" in StuLoan to get the loan index, pricipal amounts, interest rates, etc. Otherview call "getLoanData()" in InvToken, return the loan information as a dataframe named "df" for future operations;​
* Define "**repay()**" function to complete the student repayment process: call "findLoans()" in StuLoan to display all the active loans under this student's account, the student can choose the id of the loan he or she wants to repay, and inputs the repaying amount, the function completes the repayment process and update the loan information;​
* Define "**view\_loan\_history()**" function to view the information of active loans under the student's account by calling "findLoans()" in StuLoan;​
* Define "**inv\_withdraw()**" to complete the investor withdrawl process: with "viewBalance()" and "viewBalanceInLoan()" in InvToken, the investor can know how much in total he or she can withdraw both from own account balance and from the loan, then "withdraw()" in InvToken finished the procedure;​
* Define "**sell()**" function: first display  loan data with specific id by calling "get\_loan\_data()", then ask the investor to manually input the ask price and call "sell()"  in InvToken to put the ownership for sale: ​
* Define "**buy()**" function, if this investor is satisfied with the ask price listed on the market, then by typing "y" the function calls "buy()" in InvToken and switch the current investor account to the buyer's account: web3.eth.Investor = web3.eth.Buyer​

​

​