Banking System - Citadel Financial Inc.

Hashim Amin Georgia State Univ
Chul Chong Georgia State Univ
Kenny Montilus Georgia State Univ
Sarah Mottram Georgia State Univ
LaKeesha Patterson Georgia State Univ

Abstract

In this present era, social media attracts people to present their views [1] has a lot of impact in our lives. We all are well aware of the fact that social media has a lot of influence over people's mental Health like depression, dementia, schizophrenia etc. Although the usage of social media platform like Facebook, Instagram , twitter and software engineering together is not well understood, these mechanisms influence the software development practices. Software developers use and integrate into a wide range of tools ranging from code editing web-based portals . In our research project we would like to discuss about software engineering practices implemented in our project "Citadel Financial Inc.". We used resourced data, Javascript, HTML and CSS, Java, Python, and NoSQL in our project. Using this data gives the way into utilizing the machine learning models and can be extended to deep learning methods such as CNN, AE [2] and in real live scenarios such as twitter analysis. We used the most compatible architectural model, which is the singleton design pattern for our project.

1 Introduction

On a high level our product is a banking system to keep track of different accounts for a user with the ability to deposit and withdraw for anyone wanting to streamline their banking experience at our specific bank. Our product makes the process of withdrawing and depositing money a streamlined experience, also being able to track cash flow from one's account without the need to physically go to a bank.

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

There are alternatives because other banks also have mobile and web based applications, but our project is worth developing because there are many moving pieces that go into a full-functioning banking system and it will be good to learn how to fit all these different components into a single cohesive system. There are a lot of competitors for banking/money management, but some examples include Bank of America, Suntrust, Wells Fargo, but we can make something simpler and more intuitive than what already exists.

Our top-level objective is to have a browser-based banking system that is easy for anyone to use. It is different because our solution is simpler and more intuitive than the competition with a target customer of people who want to easily manage their finances. The scope of our product allows users to log in, check their accounts, transfer money between accounts, withdraw and deposit money, and see an action history with a stretch goal of loans.

We can build a simple banking system using Java as a backend and a frontend of Javascript, HTML, and CSS. An interesting component of our project is that it involves the manipulation of complex objects and relationships (i.e. users, accounts, cash flow, debit, credit, etc.) and the storing of those objects and relationships.

2 High Level Breakdown

Using a context model and use cases we breakdown how our product is supposed to work at a high level.

2.1 Context Model

Login/Authentication System: Allows users to enter their credentials and gain access to their specific account functions.

Account Management System: Allows users to view information about their account (checking/savings balances, withdrawals, deposits, transfers, etc.).

Debit/Credit System: Allows users to withdraw from their available debit or credit balances to make pur-

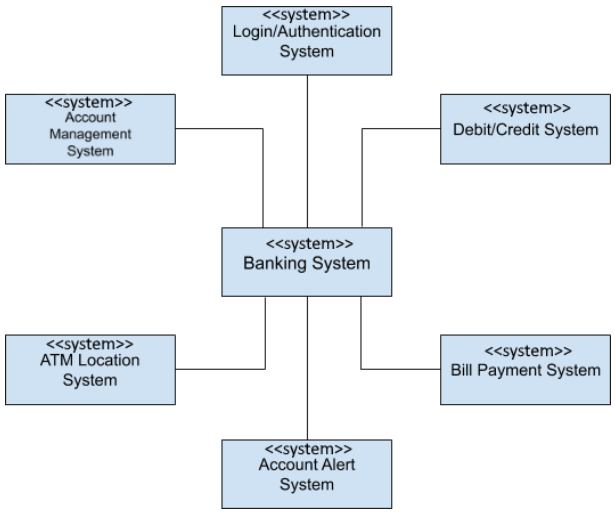

Figure 1: Context Model

chases and allows them to receive refunds.

Account Alert System: Alerts users to account details/updates that may require their attention (overdrafts, low balances, large withdrawals, etc.).

Bill Payment System: Allows users to schedule payments to specified recipients ahead of time and with continuity for bill payments.

ATM Locations System: Allows the user to find bank branches and ATMs close to their current location, providing details about that branch and available services.

2.2 Use Cases

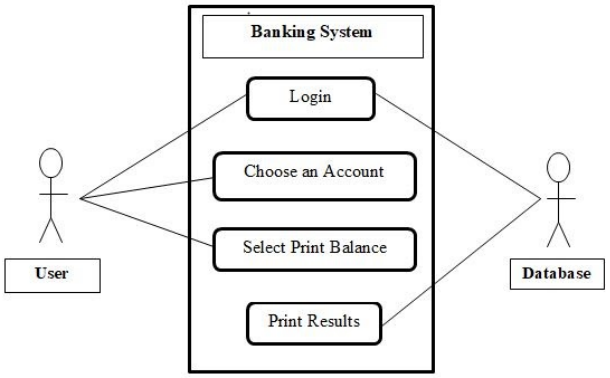

Figure 2: Use Case Diagram for Use Cases 1 and 2

Use Case no. 1 Use Case Name: Print Checking Balance

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Checking), Choose Print Balance option, Balance is displayed

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 1

Use Case number: 1

Introduction: The customer is able to see their account balance displayed on the screen.

Inputs: Checking account number

Requirements Description: Access account number and return balance

Outputs: Checking account balance

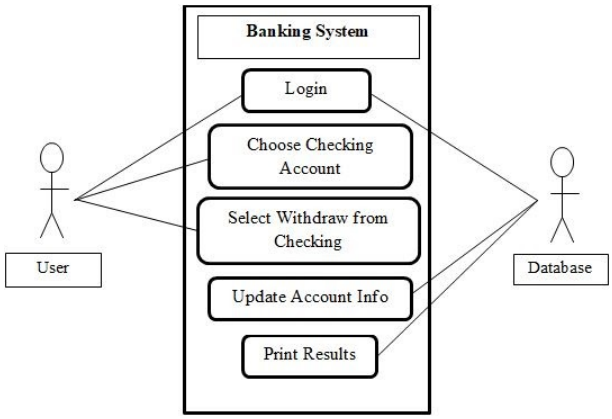

Figure 3: Use Case Diagram for Use Case 3

Use Case no. 2 Use Case Name: Print Savings Balance

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Savings), Choose Print Balance option, Balance is displayed

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 2

Use Case number: 2

Introduction: The customer is able to see their account balance displayed on the screen.

Inputs: Savings account number

Requirements Description: Access account number and return balance

Outputs: Savings account balance

Use Case no. 3 Use Case Name: Withdraw money from checking

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Checking), Choose Withdraw option, Amount is deducted from checking balance

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 3

Use Case number: 3

Introduction: After the customer's account balance is verified, the withdrawal amount is deducted from the customer's balance.

Inputs: Checking account number

Requirements Description: Access account number, verify balance then subtract withdraw amount from checking account

Outputs: Checking account balance

Use Case no. 4 Use Case Name: Withdraw money from savings

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Savings), Choose Withdraw option, Amount is deducted from savings balance

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 4

Use Case number: 4

Introduction: After the customer's account balance is verified, the withdrawal amount is deducted from the customer's balance.

Inputs: Savings account number

Requirements Description: Access account number, verify balance then subtract withdraw amount from savings account

Outputs: Savings account balance

Use Case no. 5 Use Case Name: Deposit money in checking account

Actors: Customer (initiator), ATM, Bank System

Description: The customer insert their card into ATM, ATM would require PIN, and the customer would log in with it, Customer would select what they will deposit with between cash/check, Customer would select in which account they will deposit their cash/check, Customer would insert cash and ATM would count the cash, If Customer insert check ATM would read the check numbers, After finishing counting cash / reading check, ATM would accept cash/check, Bank System would generate receipt and print it out, Customer would take the printed receipt

Exception Path: if access is denied, then an error message will be displayed, and the customer will not be allowed to proceed to the next step.

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 5

Use Case number: 5

Introduction: Customers will select if they will deposit money with cash or check and in which account between checking or saving accounts. ATM will proceed and bank system will save the record to the system

Inputs: checking account number, counts of cash / check number. Requirements Description: Access checking account, valid check numbers, counts and value of cash, and receipt

Outputs: receipt information

Use Case no. 6 Use Case Name: Deposit money in saving account

Actors: Customer (initiator), ATM, Bank System

Description: The customer insert their card into ATM, ATM would require PIN, and the customer would log in with it, Customer would select what they will deposit with between cash/check, Customer would select in which account they will deposit their cash/check, Customer would insert cash and ATM would count the cash, If Customer insert check ATM would read the check numbers, After finishing counting cash / reading check, ATM would accept cash/check, Bank System would generate receipt and print it out, Customer would take the printed receipt

Exception Path: if access is denied, then an error message will be displayed, and the customer will not be

allowed to proceed to the next step.

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 6

Use Case number: 6

Introduction: Customers will select if they will deposit money with cash or check and in which account between checking or saving accounts. ATM will proceed and bank system will save the record to the system

Inputs: saving account number, counts of cash / check number.

Requirements Description: Access saving account, valid check numbers, counts and value of cash, and receipt

Outputs: receipt information

Use Case no. 7 Use Case Name: Transfer money from checking to saving

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Checking), Choose Transfer option, Amount is deducted from checking balance, Amount is added to savings balance

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 7

Use Case number: 7

Introduction: The customer's account balance is verified, the withdrawal amount is deducted from the customer's checking account balance then added to the customer's savings account balance.

Inputs: Checking account number

Requirements Description: Access checking account, verify transfer amount, subtract amount from checking account balance and add amount to savings account balance

Outputs: Savings account number

Use Case no. 8 Use Case Name: Transfer money from saving to checking

Actors: Customer, Virtual ATM

Description: Sign-in, Choose Account (Savings), Choose Transfer option, Amount is deducted from savings balance, Amount is added to checking balance

Alternate Path: Customer can press the back button return to the previous menu or press cancel to return to the login screen.

Pre-condition: Customer login successfully

Requirement number: 8

Use Case number: 8

Introduction: The customer's account balance is verified, the withdrawal amount is deducted from the customer's savings account balance then added to the customer's checking account balance.

Inputs: Savings account number

Requirements Description: Access savings account, verify transfer amount, subtract amount from savings account balance and add amount to checking account balance.

Outputs: Checking account number

Use Case no. 9 Use Case Name: Spend money in checking using debit card

Actors: Customer (initiator), Business (payee), Bank System

Description: Use Debit Card, Amount approved by bank, Amount deducted from checking balance, Amount transferred to payee's account

Alternate Path: Account can have insufficient funds, in which case the customer and payee will be notified and the transaction will be cancelled

Pre-condition: Customer has a valid debit card tied to a checking account, has sufficient funds in checking account to cover purchase

Requirement number: 9

Use Case number: 9

Introduction: The customer's debit card and PIN are verified, the withdrawal amount is checked to ensure it is less than the checking account balance, then the withdrawal is completed and the funds are deducted from the checking account balance and transferred to the specified payee's account

Inputs: Debit card number, PIN, Payee account number

Requirements Description: Access checking account, verify transaction amount, subtract amount from checking account balance and transfer amount to payee account.

Outputs: Checking account number

Use Case no. 10 Use Case Name: Refund money back to debit card

Actors: Business (initiator), Customer, Bank System

Description: Refund initiated to debit card, Amount transferred from business' account to customer checking account, Amount added to checking balance

Alternate Path: *N/A because the business/refunder is the initiator*

Pre-condition: Customer has a valid debit card tied to a checking account

Requirement number: 10

Use Case number: 10

Introduction: The customer's debit card and PIN are verified, the refund amount is transferred to checking account, and checking account balance is updated to show new balance

Inputs: Debit card number, PIN, Refunder's account number

Requirements Description: Access checking account, verify transaction amount, transfer amount from refunder's account and add amount to checking account balance

Outputs: Checking account number

Use Case no. 11 Use Case Name: Schedule bill payments to a specified recipient

Actors: Customer (initiator), Business (payee), Bank System

Description: Input payee's information, Specify date(s) for payment to recur, Date is checked to see if a bill is due today, Bill amount deducted from checking account balance, Bill amount transferred to payee's account

Alternate Path: Account can have insufficient funds, in which case the customer will be notified and the transaction will be cancelled

Pre-condition: Successful customer login, Customer has sufficient funds to cover bill payment, payee's account number and payment information is valid

Requirement number: 11

Use Case number: 11

Introduction: The specified date arrives, the withdrawal amount is checked to ensure it is less than the checking account balance, then the withdrawal is completed and the funds are deducted from the checking account balance and transferred to the specified payee's account

Inputs: Date, Payee account number

Requirements Description: Access checking account, verify date, verify transaction amount, subtract amount from checking account balance and transfer amount to payee account.

Outputs: Checking account number

Use Case no. 12 Use Case Name: Alert Customer when account balance is low

Actors: Bank System, Customer

Description: Customer sets dollar amount for when low account balance alerts will start, Account reaches or goes under low balance amount, Alerts sent to customer through contact information on file, Alerts sent daily until account balance exceeds low balance amount again

Alternate Path: Customer's account balance never goes below low balance amount

Pre-condition: Successful customer login, Customer has checking/savings account, Customer has set low balance amount for when alerts will start

Requirement number: 12

Use Case number: 12

Introduction: The customer sets a low balance amount (this is the dollar amount for when low balance alerts will be sent out), and after each posted transaction or at the end of each day the account balance is checked to ensure it is not below the low balance amount. If it is, a low balance alert is sent to the customer daily using their contact info until the account balance exceeds the low balance amount again

Inputs: Checking/savings account balance, Low balance amount

Requirements Description: Access checking/savings account balance, access low balance amount (specified by customer), access contact information

Outputs: Account alerts

Use Case no. 13 Use Case Name: Find ATM/branch locations close to current location

Actors: Customer (initiator), GPS, Bank System

Description: Customer requests list of nearby ATMs/branches, GPS determines customer's current location, Location is used to determine nearby locations, List of nearby locations is returned to customer with ATM/branch information

Alternate Path: Customer's location cannot be determined, in which case the customer will be prompted to try again later

Pre-condition: Successful customer login, Customer location can be determined using GPS Requirement number: 13

Use Case number: 13

Introduction: The customer requests a list of nearby ATM/branch locations, the customer's location is determined using GPS, the customer's location is used to determine what ATMs/branches are nearby, and a list of these places is returned to the customer

Inputs: Customer GPS location

Requirements Description: Access customer's current GPS location

Outputs: Nearby ATM/branch locations

3 Design Measures and Patterns

Using many different kinds of diagrams and modeling techniques we further break down our product.

3.1 Class Diagram

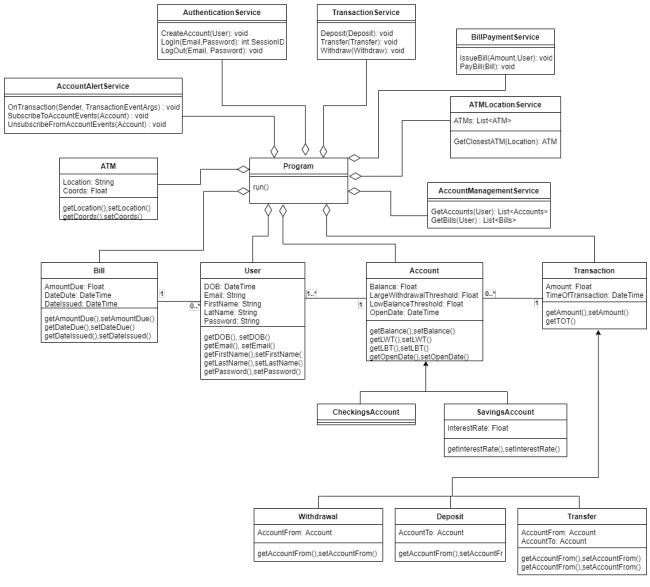

Figure 4: Class Diagram

3.1.1 Parent Tables

Customer:

customer_id (PK)	PRIMARY KEY, INT, NOT NULL
first_name	VARCHAR(25), NOT NULL
last_name	VARCHAR(25), NOT NULL
phone_num	VARCHAR(10), NOT NULL, UNIQUE
email	VARCHAR(50), NOT NULL, UNIQUE
password	VARCHAR(60), NOT NULL

ATM_Machine:

atm_id (PK)	PRIMARY KEY, INT, NOT NULL
state	VARCHAR(100), NOT NULL
city	VARCHAR(100), NOT NULL
street_address	VARCHAR(200), NOT NULL

Account:

account_id (PK)	PRIMARY KEY, INT, NOT NULL
customer_id (FK to customer table)	FOREIGN KEY, INT, NOT NULL
balance	FLOAT, NOT NULL
type	VARCHAR(25), NOT NULL

3.1.2 Child Tables

Purchase:

purchase_id (PK)	PRIMARY KEY, INT, NOT NULL
account_id (FK to account table)	FOREIGN KEY, INT, NOT NULL
customer_id (FK to customer table)	FOREIGN KEY, INT, NOT NULL
recipient (FK to customer table)	FOREIGN KEY, INT, NOT NULL
date	TIMESTAMP, NOT NULL
amount	FLOAT, NOT NULL

Transfer:

transfer_id (PK)	PRIMARY KEY, INT, NOT NULL
account_id (FK to account table)	FOREIGN KEY, INT, NOT NULL
customer_id (FK to customer table)	FOREIGN KEY, INT, NOT NULL
account_to (FK to account table)	FOREIGN KEY, INT, NOT NULL
date	TIMESTAMP, NOT NULL
amount	FLOAT, NOT NULL

Scheduled_payment:

payment_id (PK)	PRIMARY KEY, INT, NOT NULL
account_id (FK to account table)	FOREIGN KEY, INT, NOT NULL
customer_id (FK to customer table)	FOREIGN KEY, INT, NOT NULL
recipient (FK to customer table)	FOREIGN KEY, INT, NOT NULL
amount	FLOAT, NOT NULL
recurring	BOOLEAN, NOT NULL
recurring_date	DATE, NULL
recurring_cycle	VARCHAR(10), NULL

Database Specifications: Firebase Realtime Database via NoSQL

3.2 Transaction Processing System Using 4+1 Model View

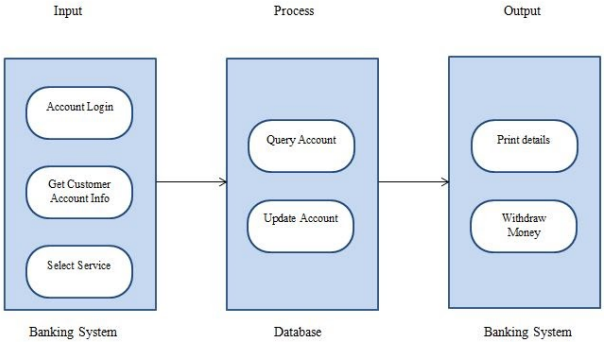

Figure 5: Software Architecture of Banking System

Bank Account Scenarios: Look up account information, Withdraw money, Deposit money, Transfer money

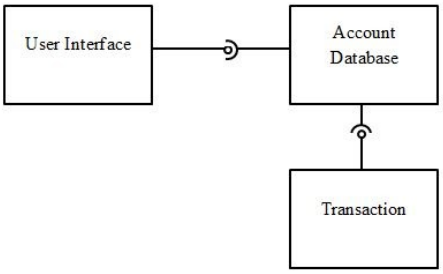

Figure 6: Logical View

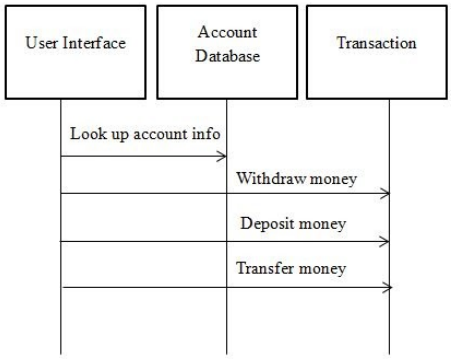

Figure 7: Process View

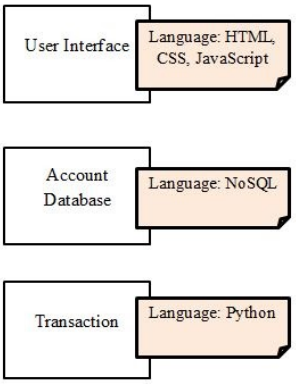

Figure 8: Development View

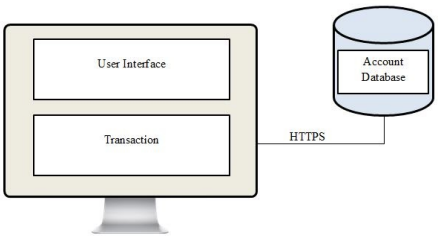

Figure 9: Physical View

3.3 Behavioral Modeling (Dynamic Modeling)

Sequence Diagrams for Use Cases

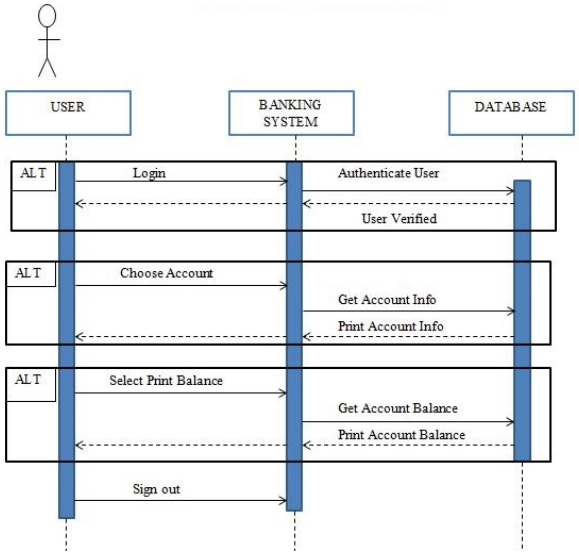

Figure 10: Sequence Diagram (Print)

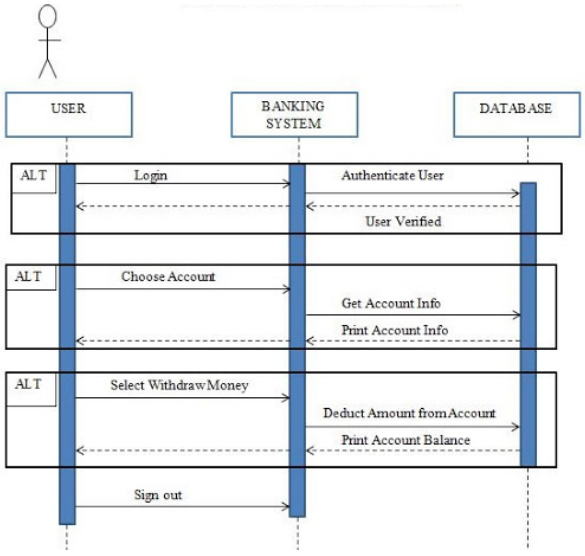

Figure 11: Sequence Diagram (Withdraw)

3.4 Design Patterns

3.4.1 Singleton Design Pattern

For the entire system, we have chosen to implement the singleton design pattern where the client program will be of a single instance that has access to all the various subsystems, objects, etc.

3.4.2 Behavioral Design Pattern

For part of our system we chose the template method of the behavioral design pattern. We chose this method since all the accounts in our banking system will follow the same basic template because they all share a lot of the same behaviors. Also, we like the flexibility to override and/or add any additional methods in order to customize features that pertain only to certain account types.

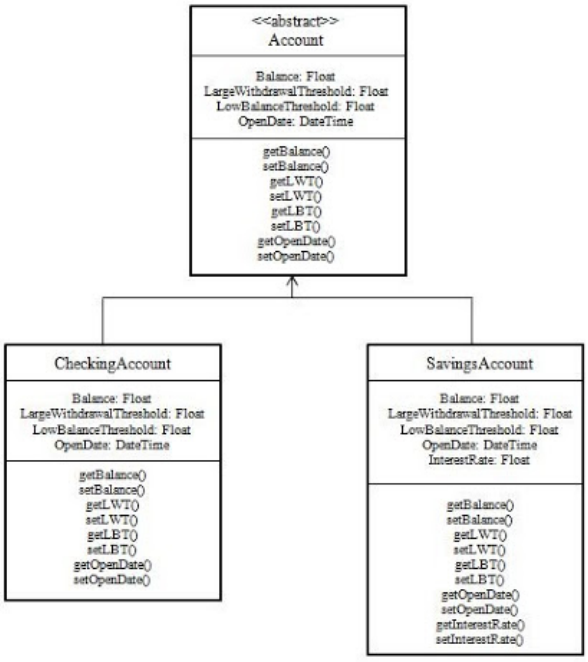

Figure 12: Behavioral Design - Template Method

4 Implementation

4.1 Install dependencies

- Install Yarn at https://classic.yarnpkg.com/en/docs/install/ or through the command line
- Install Node at https://nodejs.org/en/download/ or through the command line
- From project directory:
  - $ python -m venv citadel-venv

  - On Mac/Linux
    - $ source citadel-venv/bin/activate
  - On Windows
    - $ citadel-venv\Scripts\activate.bat
  - $ pip install -r requirements.txt
  - $ cd frontend-app
  - $ yarn install

4.2 Run the code

- Backend - from project directory:
  - $ python app.py
- Frontend - from project directory:
  - $ cd frontend-app
  - $ yarn start
- Go to localhost:3000/register on a web browser

4.3 Run unit tests

- From project directory:
  - $ pytest

5 Testing

5.1 Test Cases

Use Case 1: Printing Account Checking Balances Test Case ID: 1.1

Description: Attempt to retrieve current Checking account balance

Test Inputs: Choosing "Checking" from the dropdown menu

Expected Results: Successful, then prints out the current Checking account balance from the database

Dependencies: Valid Checking account in database

Initialization: None

Test Steps:

1. Choose "Checking" from the dropdown menu
2. Click "Submit" and wait for current Checking account balance to be printed below

Use Case 2: Printing Account Saving Balances Test Case ID: 2.1

Description: Attempt to retrieve current Savings account balance

Test Inputs: Choosing "Savings" from the dropdown menu

Expected Results: Successful, then prints out the current Savings account balance from the database

Dependencies: Valid Savings account in database

Initialization: None

Test Steps:

1. Choose "Savings" from the dropdown menu
2. Click "Submit" and wait for current Savings account balance to be printed below

Use Case 3: Withdrawing Money from Checking Test Case ID: 3.1

Description: Attempt to withdraw money from Checking Account

Test Inputs: A desired withdrawal amount that is less than current Checking Account balance

Expected Results: Successful, then prints out the new Checking Account balance from the database

Dependencies: Current Checking account balance is greater than 0.00

Initialization: None

Test Steps:

1. Choose "Checking" from the dropdown menu
2. Enter desired withdrawal amount under "Withdraw Money"
3. Click "Withdraw" and wait for new Checking Account balance to be printed above

Test Case ID: 3.2

Description: Attempt to withdraw an amount greater than the available balance

Test Inputs: A desired withdrawal amount that is greater than current account balance

Expected Results: Unsuccessful, then displays the message "Value must be greater than [current account balance]" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter desired withdrawal amount under "Withdraw Money"

3. Click “Withdraw” and wait for error message to be displayed below

Test Case ID: 3.3

Description: Attempt to enter a non-numeric value into “Withdraw Money” box

Test Inputs: A non-numeric value (such as “-”)

Expected Results: Unsuccessful, then displays the message “Please enter a number” to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose “Checking”account from the dropdown menu
2. Enter non-numeric value (such as “-”)
3. Click “Withdraw” and wait for message to be displayed below

Test Case ID: 3.4

Description: Attempt to enter a negative value into “Withdraw Money” box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message “Value must be greater than or equal to 0” to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose “Checking” account from the dropdown menu
2. Enter negative value
3. Click “Withdraw” and wait for message to be displayed below

Use Case 4: Withdrawing Money from Savings
Test Case ID: 4.1

Description: Attempt to withdraw money from Savings Account

Test Inputs: A desired withdrawal amount that is less than current Savings Account balance

Expected Results: Successful, then prints out the new Savings Account balance from the database

Dependencies: Current Savings account balance is greater than 0.00

Initialization: None

Test Steps:

1. Choose “Savings” from the dropdown menu
2. Enter desired withdrawal amount under “Withdraw Money”
3. Click “Withdraw” and wait for new Savings Account balance to be printed above

Test Case ID: 4.2

Description: Attempt to withdraw an amount greater than the available balance

Test Inputs: A desired withdrawal amount that is greater than current account balance

Expected Results: Unsuccessful, then displays the message “Value must be greater than [current account balance]” to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose “Savings” account from the dropdown menu
2. Enter desired withdrawal amount under “Withdraw Money”
3. Click “Withdraw” and wait for error message to be displayed below

Test Case ID: 4.3

Description: Attempt to enter a non-numeric value into “Withdraw Money” box

Test Inputs: A non-numeric value (such as “-”)

Expected Results: Unsuccessful, then displays the message “Please enter a number” to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose “Savings” account from the dropdown menu
2. Enter non-numeric value (such as “-”)
3. Click “Withdraw” and wait for message to be displayed below

Test Case ID: 4.4

Description: Attempt to enter a negative value into "Withdraw Money" box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message "Value must be greater than or equal to 0" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" account from the dropdown menu
2. Enter negative value
3. Click "Withdraw" and wait for message to be displayed below

Use Case 5: Transferring Money from Checking to Savings Test Case ID: 5.1

Description: Attempt to transfer money from Checking to Savings Account

Test Inputs: A desired transfer amount that is less than current Checking Account balance

Expected Results: Successful, then prints out the new Checking Account balance from the database

Dependencies: Current Checking account balance is greater than 0.00

Initialization: None

Test Steps:

1. Choose "Checking" from the dropdown menu
2. Enter desired Transfer amount under "Money Transfer to Savings"
3. Click "Transfer Money" and wait for new Checking Account balance to be printed above
4. Choose "Savings" from the dropdown menu
5. Check Savings account balance to see if it is transferred from Checking account

Test Case ID: 5.2

Description: Attempt to Transfer an amount greater than the available balance

Test Inputs: A desired Transfer amount that is greater than current account balance

Expected Results: Unsuccessful, then displays the message "Value must be greater than [current account balance]" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter desired Transfer amount under "Money Transfer to Savings"
3. Click "Transfer Money" and wait for error message to be displayed below

Test Case ID: 5.3

Description: Attempt to enter a non-numeric value into "Money Transfer to Savings" box

Test Inputs: A non-numeric value (such as "-")

Expected Results: Unsuccessful, then displays the message "Please enter a number" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter non-numeric value (such as "-")
3. Click "Transfer Money" and wait for message to be displayed below

Test Case ID: 5.4

Description: Attempt to enter a negative value into "Money Transfer to Savings" box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message "Value must be greater than or equal to 0" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter negative value
3. Click "Transfer Money" and wait for message to be displayed below

Use Case 6: Transferring Money from Savings to Checking Test Case ID: 6.1

Description: Attempt to transfer money from Savings to Checking Account

Test Inputs: A desired transfer amount that is less than current Savings Account balance

Expected Results: Successful, then prints out the new Savings Account balance from the database

Dependencies: Current Savings account balance is greater than 0.00

Initialization: None

Test Steps:

1. Choose "Savings" from the dropdown menu
2. Enter desired Transfer amount under "Money Transfer to Checking"
3. Click "Transfer Money" and wait for new Savings Account balance to be printed above
4. Choose "Checking" from the dropdown menu
5. Check Checking account balance to see if it is transferred from Savings account

Test Case ID: 6.2

Description: Attempt to Transfer an amount greater than the available balance

Test Inputs: A desired Transfer amount that is greater than current account balance

Expected Results: Unsuccessful, then displays the message "Value must be greater than [current account balance]" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" account from the dropdown menu
2. Enter desired Transfer amount under "Money Transfer to Checking"
3. Click "Transfer Money" and wait for error message to be displayed below

Test Case ID: 6.3

Description: Attempt to enter a non-numeric value into "Money Transfer to Checking" box

Test Inputs: A non-numeric value (such as "-")

Expected Results: Unsuccessful, then displays the message "Please enter a number" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Saving" account from the dropdown menu
2. Enter non-numeric value (such as "-")
3. Click "Transfer Money" and wait for message to be displayed below

Test Case ID: 6.4

Description: Attempt to enter a negative value into "Money Transfer to Checking" box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message "Value must be greater than or equal to 0" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" account from the dropdown menu
2. Enter negative value
3. Click "Transfer Money" and wait for message to be displayed below

Use Case 7: Deposit Money to Checking Test Case ID: 7.1

Description: Attempt to deposit money to Checking Account

Test Inputs: A desired deposit amount that is less than 0.00

Expected Results: Successful, then prints out the new Checking Account balance from the database

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" from the dropdown menu
2. Enter desired deposit amount under "Deposit Money"

3. Click "Deposit" and wait for new Checking Account balance to be printed above

Test Case ID: 7.2

Description: Attempt to enter a non-numeric value into "Deposit Money" box

Test Inputs: A non-numeric value (such as "-")

Expected Results: Unsuccessful, then displays the message "Please enter a number" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter non-numeric value (such as "-")
3. Click "Deposit" and wait for message to be displayed below

Test Case ID: 7.3

Description: Attempt to enter a negative value into "Deposit Money" box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message "Value must be greater than or equal to 0" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Checking" account from the dropdown menu
2. Enter negative value
3. Click "Deposit" and wait for message to be displayed below

Use Case 8: Deposit Money to Savings Test Case ID: 8.1

Description: Attempt to deposit money to Savings Account

Test Inputs: A desired deposit amount that is less than 0.00

Expected Results: Successful, then prints out the new Savings Account balance from the database

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" from the dropdown menu
2. Enter desired deposit amount under "Deposit Money"
3. Click "Deposit" and wait for new Savings Account balance to be printed above

Test Case ID: 8.2

Description: Attempt to enter a non-numeric value into "Deposit Money" box

Test Inputs: A non-numeric value (such as "-")

Expected Results: Unsuccessful, then displays the message "Please enter a number" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" account from the dropdown menu
2. Enter non-numeric value (such as "-")
3. Click "Deposit" and wait for message to be displayed below

Test Case ID: 8.3

Description: Attempt to enter a negative value into "Deposit Money" box

Test Inputs: A negative value

Expected Results: Unsuccessful, then displays the message "Value must be greater than or equal to 0" to the user

Dependencies: None

Initialization: None

Test Steps:

1. Choose "Savings" account from the dropdown menu
2. Enter negative value
3. Click "Deposit" and wait for message to be displayed below

5.2 Test Documentation

Bug	The Test Uncovered the Bug	Description of the Bug	Action was Taken to Fix the Bug
Negative Money Withdrawal	withdrawBelowZero()	It was possible to withdraw negative money which could effectively increase the money in the account by the withdrawal amount	It was made impossible to submit negative money for a withdrawal from the frontend. This action was also separately restricted from the backend endpoint as well
More Than Maximum Money Withdrawal	withdrawAboveBalance()	It was possible to withdraw more money than was within the account which would cause the balance in the account to go negative	It was made impossible to submit more money than was in the balance for a withdrawal from the frontend. This action was also separately restricted from the backend endpoint as well

Test ID	getAccountTest
Purpose of Test	To ensure that that the REST endpoint receives a GET request and gives an HTTP Response of 200 (OK) instead of 400
Test Environment	Tested from the IDE automatically when the application is run
Test Steps	Run the application from the IDE
Test Input	Get request to accounts/checking
Expected Result	HTTP Response 200

Test ID	test_print_account_checking_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and gives an HTTP Response of 200 (OK) instead of 400 and the correct checking balance is returned
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200, Checking accounts balance == 0
Likely Bugs Revealed	Checking balance isn’t being properly queried, the json response is being mangled

Test ID	test_print_account_savings_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and gives an HTTP Response of 200 (OK) instead of 400 and the correct savings balance is returned
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200, Savings accounts balance == 0
Likely Bugs Revealed	Savings balance isn’t being properly queried, the json response is being mangled

Test ID	test_withdraw_money_from_checking_less_than_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and gives an HTTP Response of 200 (OK) and users can withdraw money in their checking account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 200, ”Withdraw Successful” message
Likely Bugs Revealed	Checking balance is allowed a negative number

Test ID	test_withdraw_money_from_savings_less_than_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and gives an HTTP Response of 200 (OK) and users can withdraw money in their savings account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 200, ”Withdraw Successful” message
Likely Bugs Revealed	Savings balance is allowed a negative number

Test ID	test_withdraw_money_from_checking_more_than_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users cannot withdraw more than is in their checking account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 400, ”Withdraw Unsuccessful” message
Likely Bugs Revealed	Checking balance is allowed a negative number

Test ID	test_withdraw_money_from_savings_more_than_balance
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users cannot withdraw more than is in their savings account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 400, ”Withdraw Unsuccessful” message
Likely Bugs Revealed	Savings balance is allowed a negative number

Test ID	test_withdraw_money_from_checking_negative_amount
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users cannot withdraw a negative amount
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 400, ”Withdraw Unsuccessful. Cannot withdraw a negative amount of money.” message
Likely Bugs Revealed	Users being able to add money to their account

Test ID	test_withdraw_money_from_savings_negative_amount
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users cannot withdraw a negative amount
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 400, ”Withdraw Unsuccessful. Cannot withdraw a negative amount of money.” message
Likely Bugs Revealed	Users being able to add money to their account

Test ID	test_deposit_money_to_checking
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users can add money to their checking account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 200, ”Deposit Successful” message
Likely Bugs Revealed	Users cannot deposit into their checking account

Test ID	test_deposit_money_to_savings
Purpose of Test	To ensure that that the REST endpoint receives a GET request and users can add money to their savings account
Test Environment	Pytest
Test Steps	Run “pytest” in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP 200, ”Deposit Successful” message
Likely Bugs Revealed	Users cannot deposit into their savings account

Test ID	test_transfer_money_checking_to_savings
Purpose of Test	To see if money can be moved from a user's checking account to their savings account
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200, "Transfer Successful" message
Likely Bugs Revealed	Money can't be moved from checking to savings

Test ID	test_transfer_money_savings_to_checking
Purpose of Test	To see if money can be moved from a user's savings account to their checking account
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200, "Transfer Successful" message
Likely Bugs Revealed	Money can't be moved from savings to checking

Test ID	test_transfer_money_checking_to_savings_negative_amount
Purpose of Test	To make sure user's cannot move negative amounts of money
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Transfer Unsuccessful. Transfer amount cannot be negative" message
Likely Bugs Revealed	Users can move negative amounts of money

Test ID	test_transfer_money_savings_to_checkings_negative_amount
Purpose of Test	To make sure user's cannot move negative amounts of money
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Transfer Unsuccessful. Transfer amount cannot be negative" message
Likely Bugs Revealed	Users can move negative amounts of money

Test ID	test_transfer_money_checking_to_savings_greater_than_balance
Purpose of Test	To ensure a transfer cannot be made from checking to savings for an amount greater than checking account balance
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Transfer Unsuccessful. Transfer amount exceeds origin account balance." message
Likely Bugs Revealed	Users can make transfers with more money than they have in their checking account

Test ID	test_transfer_money_savings_to_checking_greater_than_balance
Purpose of Test	To ensure a transfer cannot be made from savings to checking for an amount greater than savings account balance
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Transfer Unsuccessful. Transfer amount exceeds origin account balance." message
Likely Bugs Revealed	Users can make transfers with more money than they have in their savings account

Test ID	test_transaction_record_creation_when_transaction_occurs
Purpose of Test	To ensure that a new transaction object is created with the correct attributes each time a transaction takes place
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200
Likely Bugs Revealed	Transaction objects are not created

Test ID	test_payment_from_checking_equal_to_balance
Purpose of Test	To make sure users can make payments from checking account if they have enough money
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 200, "Payment Successful" message
Likely Bugs Revealed	Users cannot make payments

Test ID	test_payment_from_checking_greater_than_balance
Purpose of Test	To make sure users cannot pay with money they do not have from checking
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Payment Unsuccessful. Payment amount exceeds account balance." message
Likely Bugs Revealed	Users can pay with money they do not have

Test ID	test_payment_from_savings_greater_than_balance
Purpose of Test	To make sure users cannot pay with money they do not have from savings
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Payment Unsuccessful. Payment amount exceeds account balance." message
Likely Bugs Revealed	Users can pay with money they do not have

Test ID	test_payment_from_checking_negative_amount
Purpose of Test	To ensure payments cannot be made from checking account using negative amounts
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Payment Unsuccessful. Payment amount cannot be negative." message
Likely Bugs Revealed	Users can make payments using negative account balances

Test ID	test_payment_from_savings_negative_amount
Purpose of Test	To ensure payments cannot be made from savings account using negative amounts
Test Environment	Pytest
Test Steps	Run "pytest" in terminal
Test Input	Test_client which is mock client for the flask backend application
Expected Result	HTTP Response 400, "Payment Unsuccessful. Payment amount cannot be negative." message
Likely Bugs Revealed	Users can make payments using negative account balances

Conclusion

By working on Citadel Financial Inc. we were able to gain a holistic understanding of the software development cycle as well as gain some insight into the domain of banking and finance. By creating the various diagrams (4+1 model view, use-cases diagrams, sequence diagrams, etc.) it became clear to us that a software product is more than just the code that is written. There is a lot of effort that needs to go into the planning and development of the structure of the software product even before a single line of code is written.

We also learned the importance of unit testing every single portion of the software. Programmers are prone to error, and there is a need for a workflow that allows the team to catch any bugs that have made their way into the software. Unit testing before pushing and deploying code is the key because it allows the team to catch any errors in the system before the current version of the system is committed to the repository. A specific example of this in our case was when we were able to find a bug where a negative amount was able to be withdrawn from an account, when in fact this should not have been possible. We found this bug from running our suite of unit tests, in which the specific unit tests regarding negative amount withdrawal failed, hence the importance of testing before making big changes to a system.

We would like to acknowledge Professor Tushara Sadasivuni for her lectures on the various topics in the software development field (diagramming, unit testing, software development cycle, etc.) which greatly helped us throughout the course of this semester to be able to complete this project.

References

[1] S. T. Sadasivuni and Y. Zhang, "Using gradient methods to predict twitter users' mental health with both covid-19 growth patterns and tweets," *second IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence (HCCAI 2020) September 21-23, 2020 Irvine, CA, USA*.

[2] J. K. Mandivarapu, B. Camp, and R. J. Estrada, "Self-net: Lifelong learning via continual self-modeling," *Frontiers in Artificial Intelligence*, vol. 3, p. 19, 2020.