Project Design

Team Sudoers

Raysean Jones-Dent, Tonye Andre Martial, Matthew Mitchell, Kristine Dudley, Woo Choi, Justin Kim

April 9, 2019

CMSC 495 7982

Dr. Janell Robinson

University of Maryland University College

# ATM Machine Project Design

## Design Summary

Or ATM machine project is intended to simulate basic operations that a user would perform on a real ATM machine. As stated in the project plan, our ATM project will be a java-based application and the user interface will be developed using JavaFX.

The following diagram represents a high-level overview of the layered application design approach we will use to implement this project.

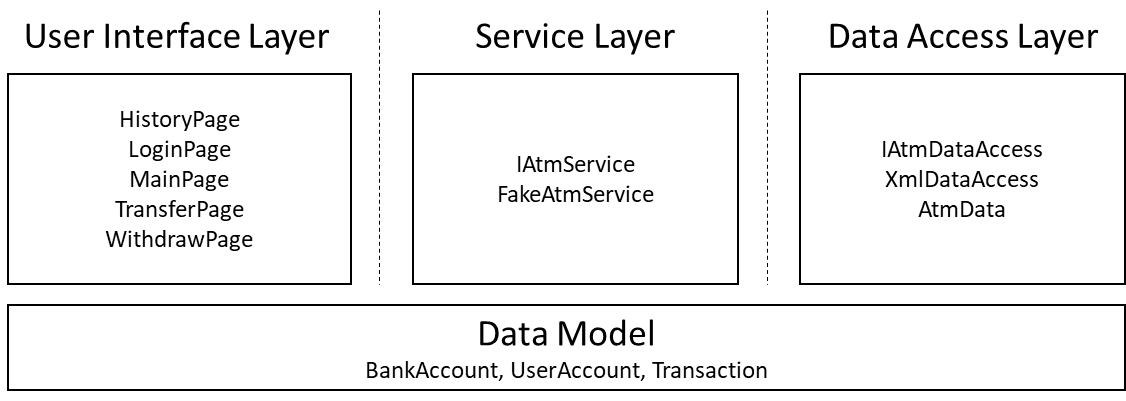


Figure 1Application Architecture Overview

## and Types

As shown above, the data model used to represent information in our application consists of three basic data structures:

In addition to the above data structures, several other Java

user JavaFX T. Output typically be JavaFX F,, /or

## Classes, Methods, Fields, Interfaces, FXML

|  |  |  |
| --- | --- | --- |
| **Files** | **Purpose** | **Attributes/Methods** |
| VirtualAtm.java | JavaFX application entry point | **Methods**  Start()  Main() |
| BaseAtmController.java | Base class from which all UI controllers will derive. Will contain helper methods and references to facilitate navigation, atm service access, language translation. | **Attributes**  ResourceBundle resources  URL location  IAtmService atmService  ObjectProperty<Locale> locale  IAtmService atmService  Stage primaryStage  **Methods**  void initialize(URL url, ResourceBundle rb)  IAtmService getAtmService()  void showMainPage()  void showLoginPage()  void showDepositPage()  void showHistoryPage()  void showTransferPage()  void showWithdrawPage()  void setStage(Stage stage)  void showPage(String fxmlPath)  showError(String message)  void setLanguageId(String langId, String country)  String getTranslatedText(final String key, final Object... args)  StringBinding createTranslatedTextBinding(final String key, Object... args)  showError(String message) |
| DepositPage.fxml | Holds layout and control information for the Deposit Page | N/A |
| DepositPageController.java | The “controller” for the deposit page. Contains method which validate user input for making a deposit and trigger the operation on the atm service layer. | **Atributes**  Label topLabel  Label checkingAmountLabel  Label savingsAmountLabel  Label lastTransactionDateLabel  TextField depositAmount  String selectedAccountType  ComboBox<String> fromAccount  **Methods**  void initialize(URL url, ResourceBundle rb)  void refresh()  void handleDepositAccountType(ActionEvent event)  void handleDepositAction(ActionEvent event)  void handleLogoutAction(ActionEvent event)  boolean validateUserInput()  double parseDepositAmount(String text)  void handleReturnAction(ActionEvent event) |
| HistoryPage.fxml | Holds layout and control information for the History Page | N/A |
| HistoryPageController.java | The “controller” for the history page. Contains methods which will allow the user to logout of the application or return to the main page. | **Attributes**  Label topLabel  Label checkingAmountLabel  Label savingsAmountLabel  Label lastTransactionDateLabel  TableView<Transaction> historyTableView  TableColumn<Transaction, Date> dateColumn  TableColumn<Transaction, String> typeColumn  TableColumn<Transaction, Double> amountColumn  **Methods**  void initialize(URL url, ResourceBundle rb)  void handleLogoutAction(ActionEvent event)  void handleReturnAction(ActionEvent event)  void refresh() |
| LoginPage.fxml | Holds layout and control information for the Login Page | N/A |
| LoginPageController.java | The “controller” for the login page. Contains methods which validate user input for logging into the application and changing languages. | **Attributes**  Label welcomeText  TextField userName  PasswordField userPin  Button loginButton  **Methods**  void initialize(URL url, ResourceBundle rb)  void handleLoginAction(ActionEvent event)  void handleDeutschAction(ActionEvent event)  void handleEnglishAction(ActionEvent event)  void handleFrenchAction(ActionEvent event)  void handleKoreanAction(ActionEvent event)  void handleLoginAction(ActionEvent event)  void handleSimplifiedChineseAction(ActionEvent event)  void handleSpanishAction(ActionEvent event)  boolean validateUserInput() |
| MainPage.fxml | Holds layout and control information for the Login Page | N/A |
| MainPageController.java | The “controller” for the main page. Contains methods which allow the user to display the account history, deposit, transfer, and withdraw pages. | **Attributes**  Label topLabel  Label checkingAmountLabel  Label savingsAmountLabel  Label lastTransactionDateLabel  **Methods**  void initialize(URL url, ResourceBundle rb)  void handleAccountHistoryAction(ActionEvent event)  void handleDepositAction(ActionEvent event)  void handleLogoutAction(ActionEvent event)  void handleTransferAction(ActionEvent event)  void handleWithdrawAction(ActionEvent event) |
| TransferPage.fxml | Holds layout and control information for the Login Page | N/A |
| TransferPageController.java | The “controller” for the deposit page. Contains method which validate user input for making a tran and trigger the operation on the atm service layer. | **Attributes**  Label topLabel  Label checkingAmountLabel  Label savingsAmountLabel  Label lastTransactionDateLabel  TextField transferAmount  ComboBox<String> fromAccount  ComboBox<String> destinationAccount  **Methods**  void initialize(URL url, ResourceBundle rb)  void handleLogoutAction(ActionEvent event)  void handleTransferAction(ActionEvent event)  boolean validateUserInput()  void refresh()  void handleToAccount(ActionEvent event)  void handleFromAccount(ActionEvent event)  double parseWithdrawalAmount(String text) |
| WithdrawPage.fxml | Holds layout and control information for the Login Page | N/A |
| WithdrawPageController.java | The “controller” for the withdraw page. Contains methods which will handle user input validation and trigger the operation on the service layer. | **Attributes**  Label topLabel  Label checkingAmountLabel  Label savingsAmountLabel  Label lastTransactionDateLabel  TextField otherDepositAmount  String selectedAccountType  double withdrawAmount  ComboBox<String> fromAccount  ComboBox<String> selectAmount  **Methods**  void initialize(URL url, ResourceBundle rb)  void handleConfirmAction(ActionEvent event)  void handleLogoutAction(ActionEvent event)  void handleSelectAccountChange(ActionEvent event)  void handleSelectionAmountChanged(ActionEvent event)  void refresh()  boolean validateUserInput()  double parseWithdrawalAmount(String text)  void handleReturnAction(ActionEvent event) |
| IAtmService | The interface definition of the atm service layer. | **Methods**  boolean login(String username, String password)  void logout()  UserAccount getLoggedInUser()  BankAccount getCheckingAccount()  BankAccount getSavingsAccount()  List<Transaction> getAccountHistory(UserAccount user)  Transaction getLastTransaction()  void withdraw(double amount, BankAccount account)  void transfer(double amount, BankAccount source, BankAccount destination)  void deposit(double amount, BankAccount destination) |
| FakeAtmService.java | Concrete implementation of the IAtmService interface. Will be used for initial user interface development. Contains methods that will generate fixed information used by the user interface developer. | **Attributes**  final int MAX\_FAILED\_LOGINS  final int LOCKOUT\_SECONDS  final List<UserAccount> userAccounts  final List<Transaction> transactions  final BankAccount checkingAccount  final BankAccount savingsAccount  UserAccount currentUser  **Methods**  boolean login(String username, String password)  void logout()  UserAccount getLoggedInUser()  BankAccount getCheckingAccount()  BankAccount getSavingsAccount()  List<Transaction> getAccountHistory(UserAccount user)  Transaction getLastTransaction()  void withdraw(double amount, BankAccount account)  void transfer(double amount, BankAccount source, BankAccount destination)  void deposit(double amount, BankAccount destination) |
| LocalAtmService.java | Concrete implementation of the IAtmService interface. Performs application business logic. Retrieves and stores information using the XML backed data access layer. | **Attributes**  final int MAX\_FAILED\_LOGINS  final int LOCKOUT\_SECONDS  final String XML\_FILE\_PATH  final IAtmDataAccess dataAccessLayer  UserAccount currentUser  **Methods**  boolean login(String username, String password)  void logout()  UserAccount getLoggedInUser()  BankAccount getCheckingAccount()  BankAccount getSavingsAccount()  List<Transaction> getAccountHistory(UserAccount user)  Transaction getLastTransaction()  void withdraw(double amount, BankAccount account)  void transfer(double amount, BankAccount source, BankAccount destination)  void deposit(double amount, BankAccount destination) |
| IAtmDataAccess.java | Interface definition for the data access layer used by the project. | **Methods**  Boolean Save(Boolean force)  List<UserAccount> getAllUserAccounts()  UserAccount findUserAccount(String userName)  List<BankAccount> findAllBankAccounts(UserAccount account)  BankAccount findBankAccount(long accountNumber)  void addUserAccount(UserAccount account)  void addBankAccount(BankAccount account)  void addTransaction(Transaction transaction)  List<Transaction> getTransactionsForUser(UserAccount user) |
| XmlDataAccess.java | Concrete implementation of the IAtmDataAccess interface. Used to store data records into an xml file. | **Attributes**  String filePath  Boolean dirty  AtmData dataCache  **Methods**  XmlDataAccess(String xmlPath)  Boolean Save(Boolean force)  Boolean Save(String path, Boolean force)  Boolean Load()  List<UserAccount> getAllUserAccounts()  List<BankAccount> findAllBankAccounts(UserAccount account)  UserAccount findUserAccount(String userName)  void addBankAccount(BankAccount account)  void addTransaction(Transaction transaction)  void addUserAccount(UserAccount account)  void ReadFile(String path)  void WriteFile(String path, AtmData data) |
| AtmData.java | Helper class to facilitate storing the application data records into a single xml file. | **Attributes**  List<UserAccount> userAccountsList  List<BankAccount> bankAccountsList;  List<Transaction> transactionsList  **Methods**  List<BankAccount> getBankAccounts()  setBankAccounts(List<BankAccount> accounts)  List<UserAccount> getUserAccounts()  void setUserAccounts(List<UserAccount> accounts)  List<Transaction> getTransactions()  setTransactions(List<Transaction> transactions) |
| UserAccount.java | Simple data structure that represents all data for a user account. Contains accessors and mutators for each data member. | **Attributes**  long id  String firstName  String lastName  String cellNumber  String email  String userName  String pin  int failedLoginCount  Date lastFailedLogin  **Methods**  String getFirstName()  void setFirstName(String value)  String getLastName()  void setLastName(String value)  String getCellNumber()  void setCellNumber(String value)  String getEmail()  void setEmail(String value)  String getUserName()  void setUserName(String value)  String getPin()  void setPin(String value)  long getID()  void setID(long value)  int getFailedLoginCount()  void setFailedLoginCount(int failedLoginCount)  Date getLastFailedLogin()  void setLastFailedLogin(Date value) |
| BankAccount.java | Simple data structure that represents the data needed for a bank account. Contains accessors and mutators for each data member. | **Attributes**  long userId  String accountType  long accountNumber  double accountBalance  **Methods**  long getAccountNumber()  void setAccountNumber(long accountNumber)  double getAccountBalance()  void setAccountBalance(double accountBalance)  long getUserId()  void setUserId(long userId)  String getAccountType()  void setAccountType(String accountType) |
| Transaction.java | Simple data structure that represents all data for a transaction. Contains accessors and mutators for each data member. | **Attributes**  long id  long bankAccountId  Date date  double amount  String activityType  **Methods**  long getID()  void setID(long value)  long getBankAccountId()  void setBankAccountId(long bankAccountId)  Date getDate()  void setDate(Date date)  double getAmount()  void setAmount(double amount)  String getActivityType()  void setActivityType(String activityType) |
| Security.java |  | **Attributes**  final int DEFAULT\_SALT\_LENGTH  final int DEFAULT\_HASH\_ITERATIONS  final String FIELD\_SEPERATOR  **Methods**  String createHash(String value)  boolean compareHash(String hash, String value)  byte[] deriveKey(AlgorithmTypes alg, char[] value, byte[] salt, int iterations) |

# UML Diagram

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. This study focuses on the importance of UML diagrams (class, use case, sequence, activity, component, deployment) during software development. In this case, it outlines different relationships (dependencies, generalization or aggregation) of our classes. The following is our rough draft UML diagram.

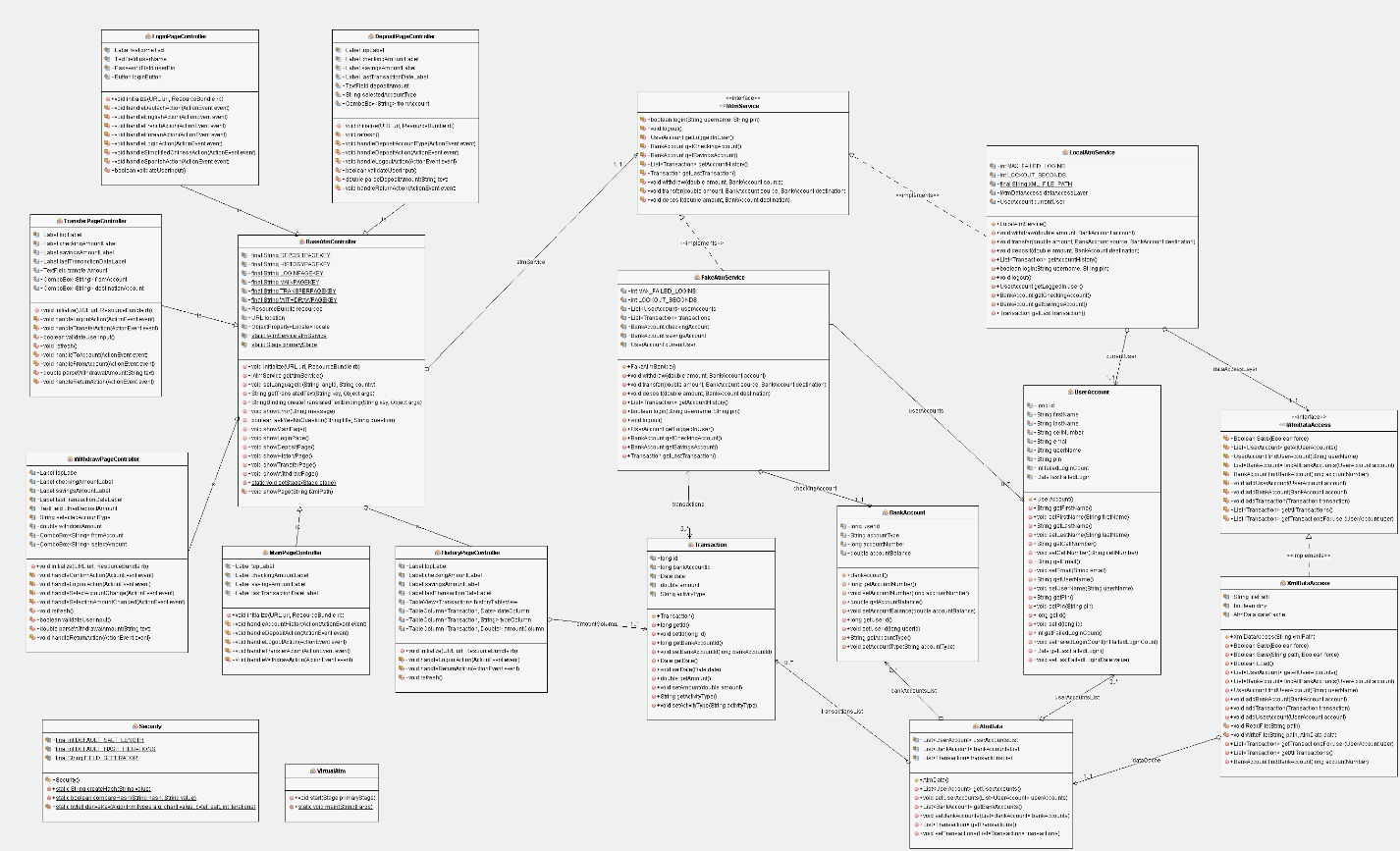


Figure 3Full UML Diagram

## UML Sub-Diagrams

Breaking up the UML diagram allows to the project to comprehend this phase of our project. From our UML diagram, we came up three important sub-diagrams to describe the different layers within the virtual ATM program: User Interface, Service, and Data Access.

## Service Layer

The diagram below outlines the methods in the IAtmService interface. It also demonstrates the relationship between FakeATMService (which is nothing more than a service class intended to test the functionality of the corresponding code) and the IAtmService. This layer will contain the “business logic” of our virtual atm application. It will also act as a mediator between the user interface and the data access layer, so the user interface does not have to worry about where information is stored.

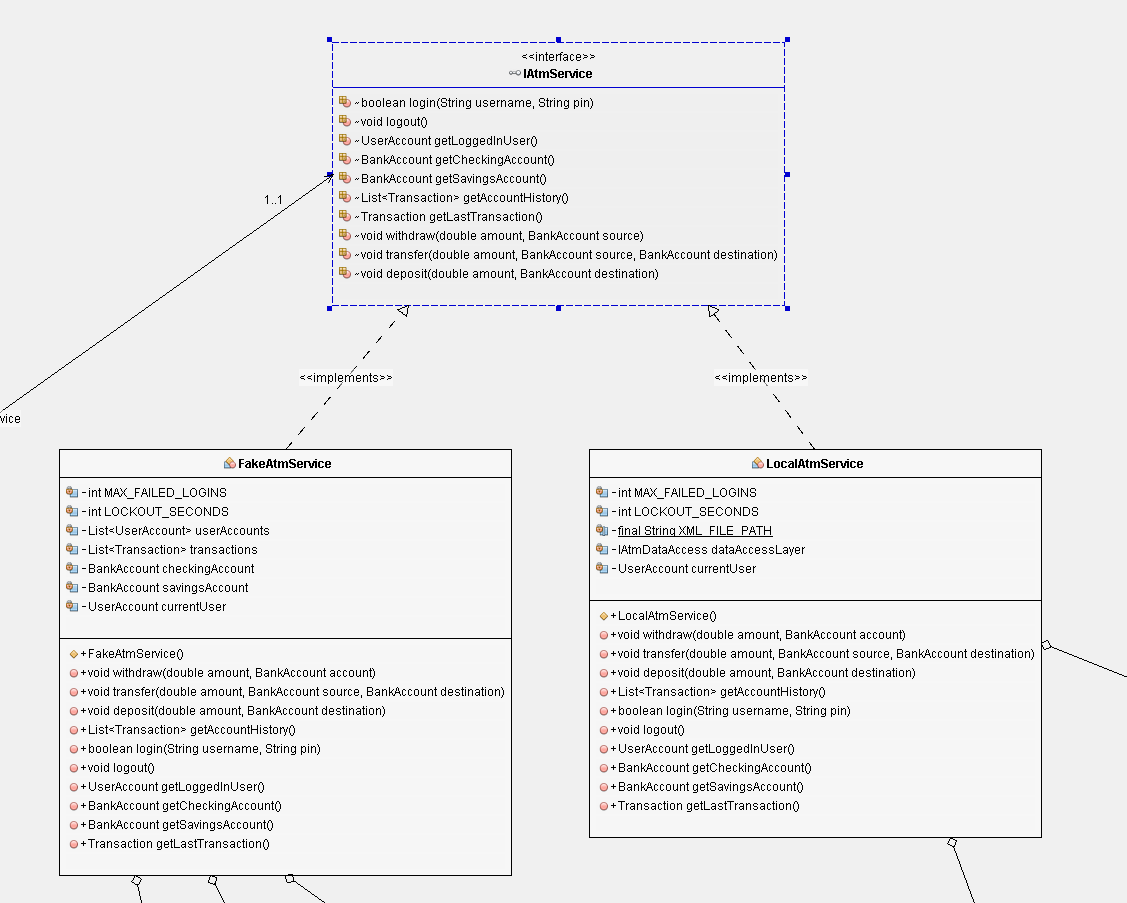


Figure 4ATM Service Layer

## User Interface Layer

Since our project is based on the JavaFX framework, which follows the model-view-controller pattern, we needed to create “controller” classes to represent each page within our User Interface. This sub-diagram contains the most relationships and dependencies because the User Interface (UI) serves as the largest part of the program. The BaseAtmController class is at the center of every other class because it contains common functionality shared between each of the User Interface controllers.

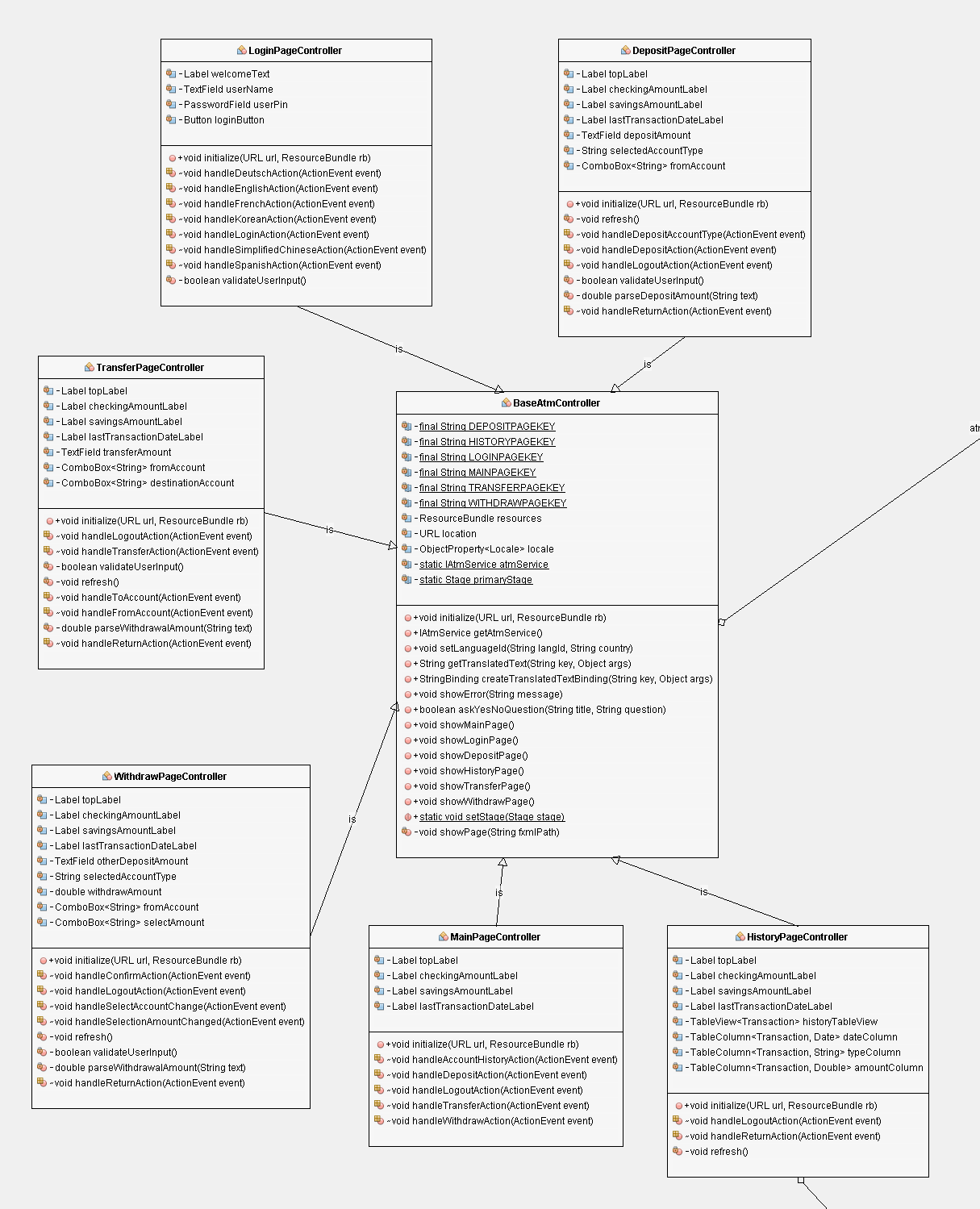


Figure 5User Interface Layer UML

## Data Access Layer

The diagram below shows the combined data access layer and the data model classes for the application. We have defined the interface IAtmDataAccess in order to insulate the rest of the application from future changes to how the data stored. The XmlDataAccess class, which implements the IAtmDataAccess interface, is responsible for serialization and deserialization of the our data model to and from the XML file.

As mentioned previously, the three basic classes BankAccount, UserAccount, and Transaction make up the entire data model for the application. We chose to use these same three classes to store/retrieve information from the data access layer, so they have been incorporated into the IAtmDataAccess interface as well.

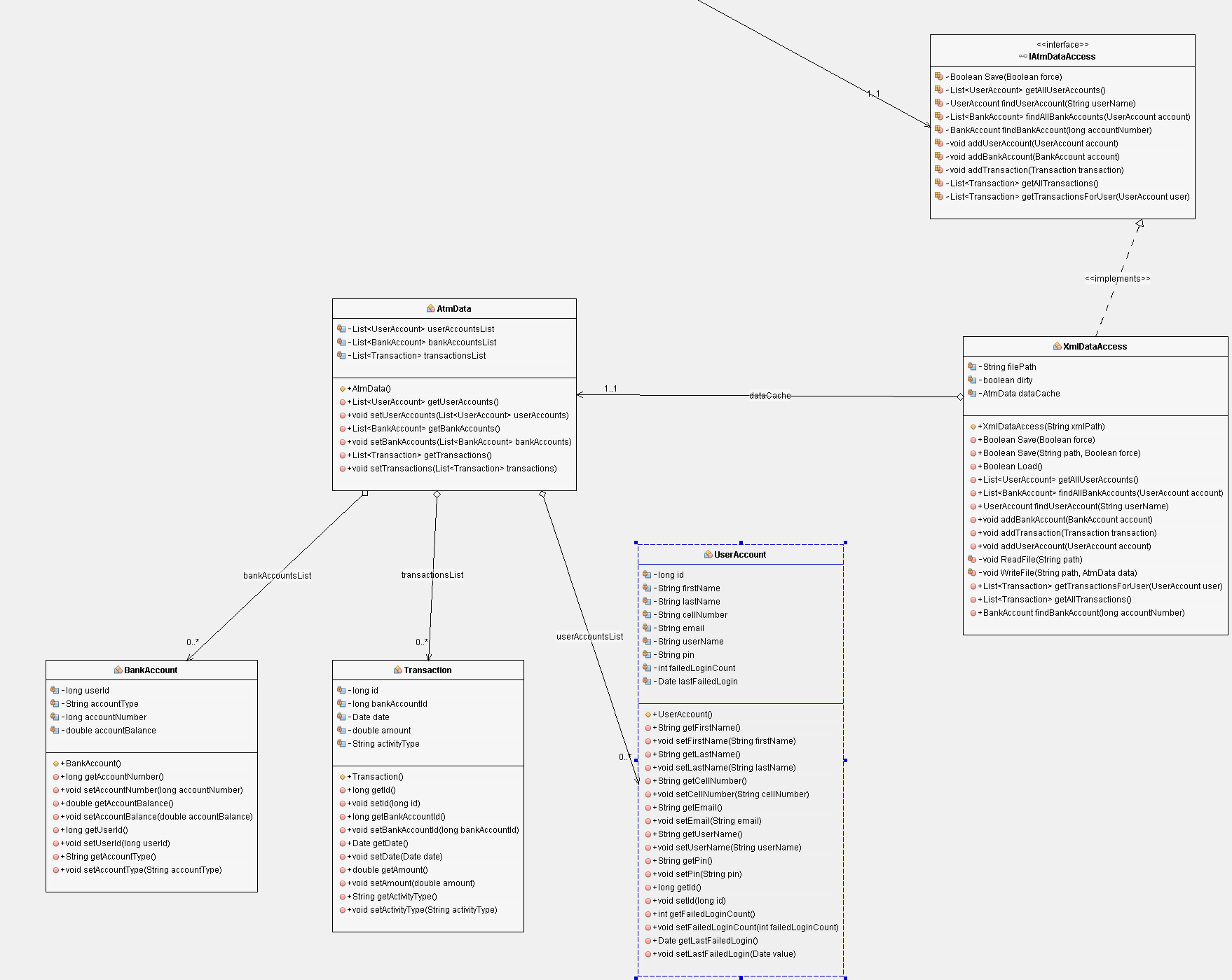


Figure 6Data Model and Access Layer

# ATM Graphics Design

The following are the graphics that were previously submitted along with the User Guide. These graphics played a very important role during the design phase of the application. They paved a way that was initially not considered (using XML files to build our application). But more importantly, the helped us critically think about how our application will look like, will work but also how our windows will communicate (dynamic or static windows).

In Figure 2.1, the LoginPageController class will be called upon to load the ATM Login Page. If the username or password are incorrectly entered or missing, a message box containing an error message will appear (see Figure 2.2).

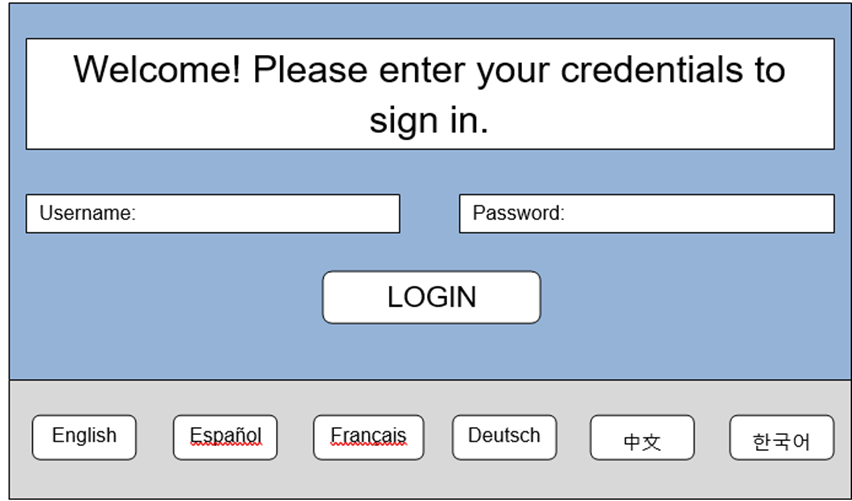


Figure .1 ATM Login Page

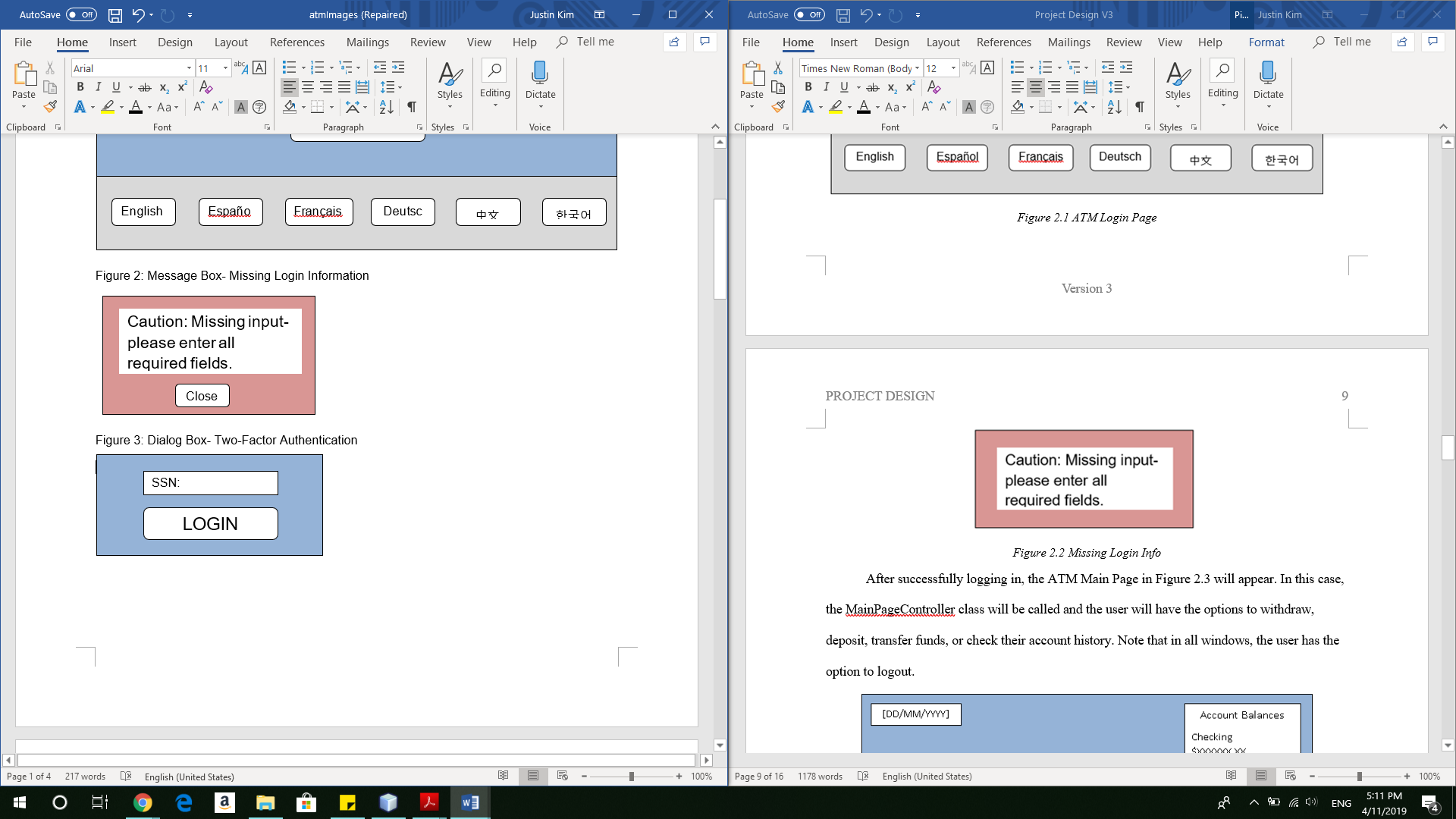


Figure 2.2 Missing Login Info

After successfully logging in, the ATM Main Page in Figure 2.3 will appear. In this case, the MainPageController class will be called and the user will have the options to withdraw, deposit, transfer funds, or check their account history. Note that in all windows, the user has the option to logout.

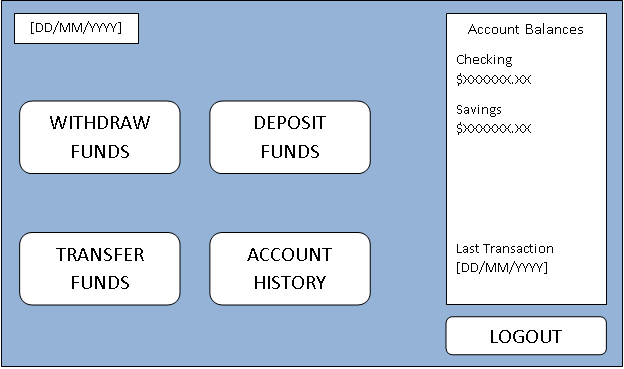


Figure 2.3 ATM Main Page

In the event the user clicks on the Deposit button, a Deposit page will appear (Figure 2.4) The deposit window allows for a customer to deposit cash into an account of their choosing (checking or savings). At this moment, an XML file will be loaded to the main controller and will display an interface to perform the intended operation. Note that the account balances are displayed in all transaction windows.

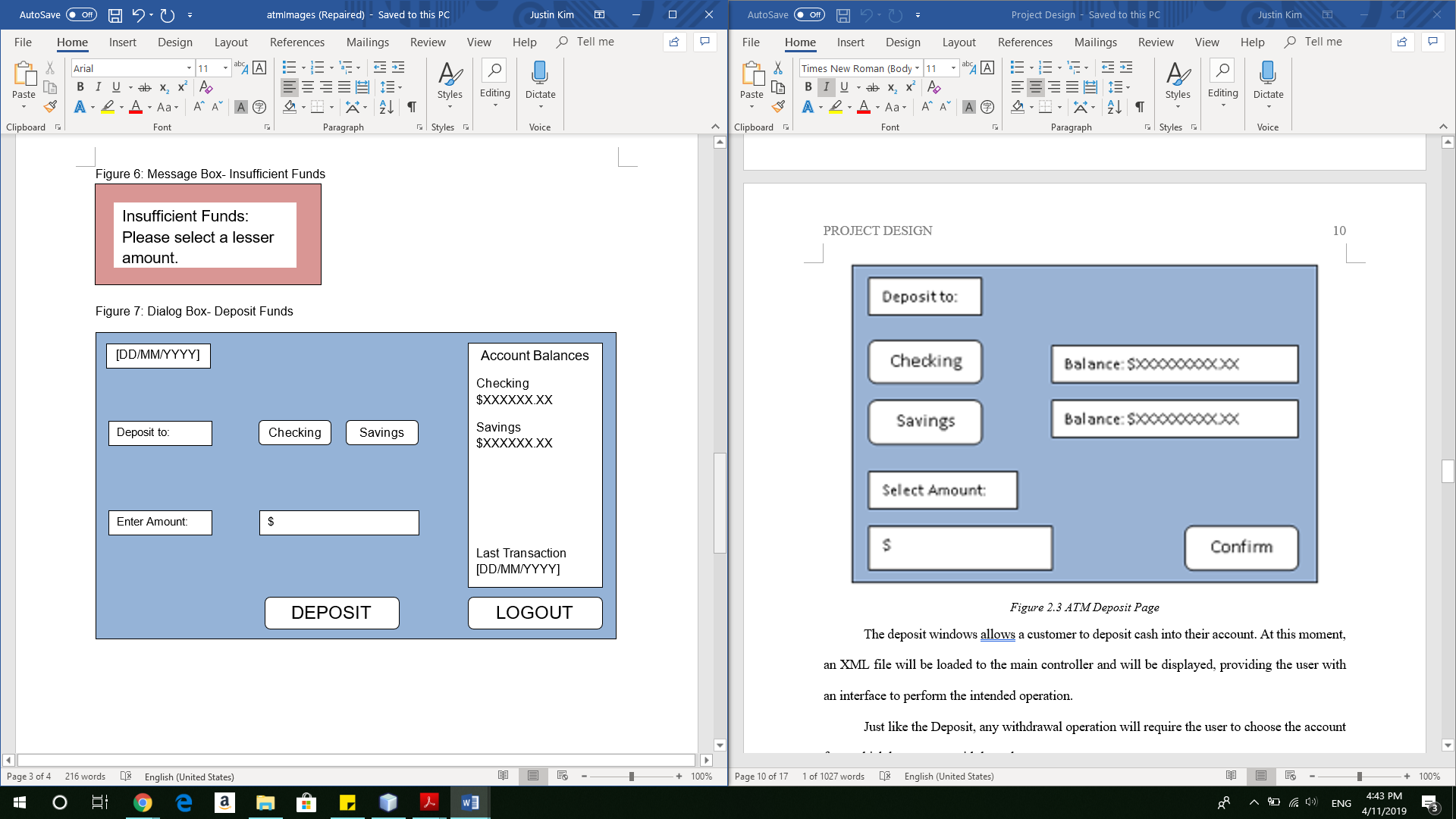


Figure 2.4 ATM Deposit Page

The Withdrawal Page shown in Figure 2.5, allows the user to withdraw cash from either account. This page allows the user to withdraw cash in increments of $20, $40, $60, $80, $100, or $200. The user is also able to enter in a specific amount (in increments of 20). If the account chosen to withdraw from has insufficient funds, a message box (Figure 2.6) will appear, letting the user know that there are not enough funds to withdraw.

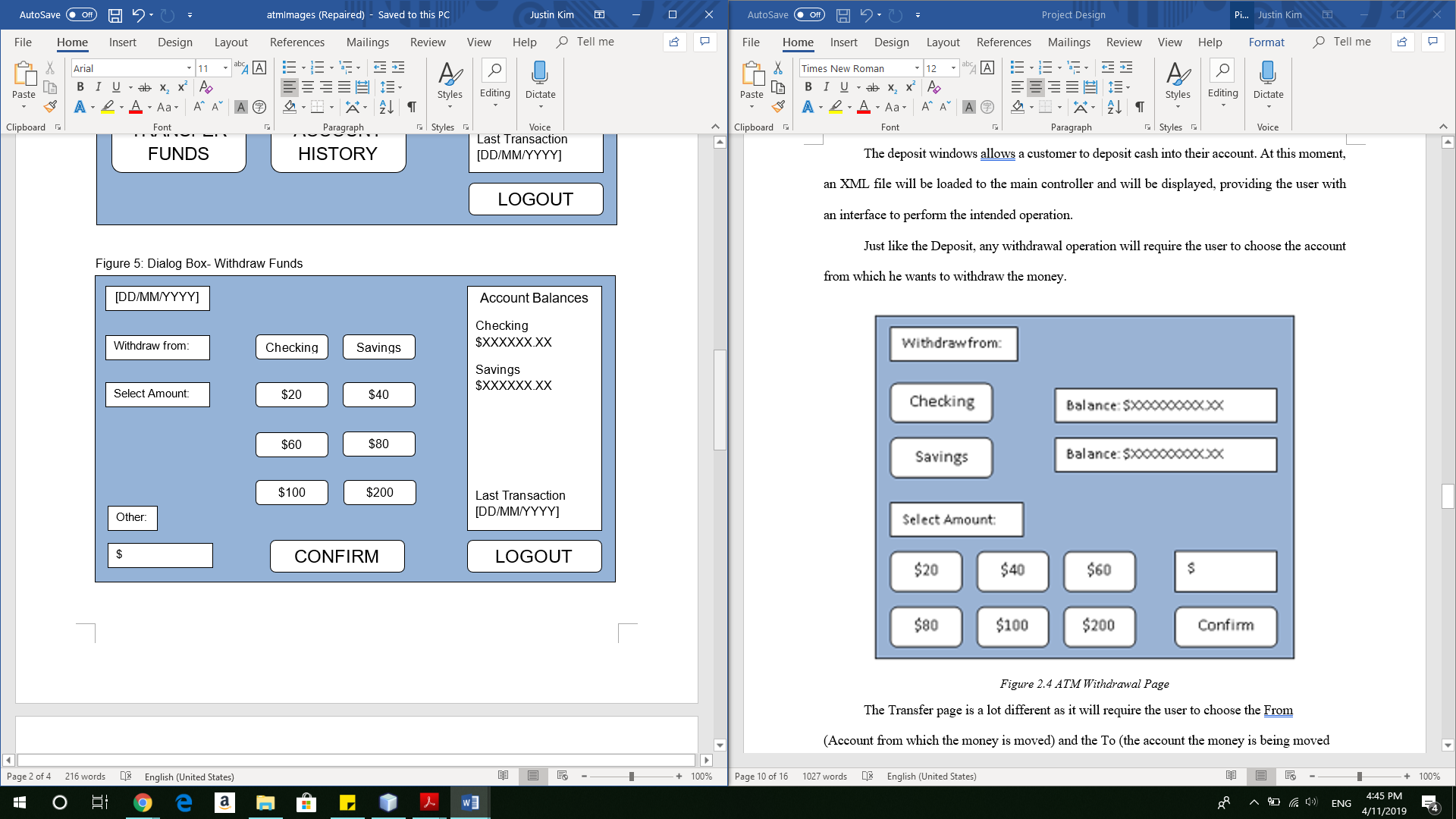


Figure 2.5 ATM Withdrawal Page

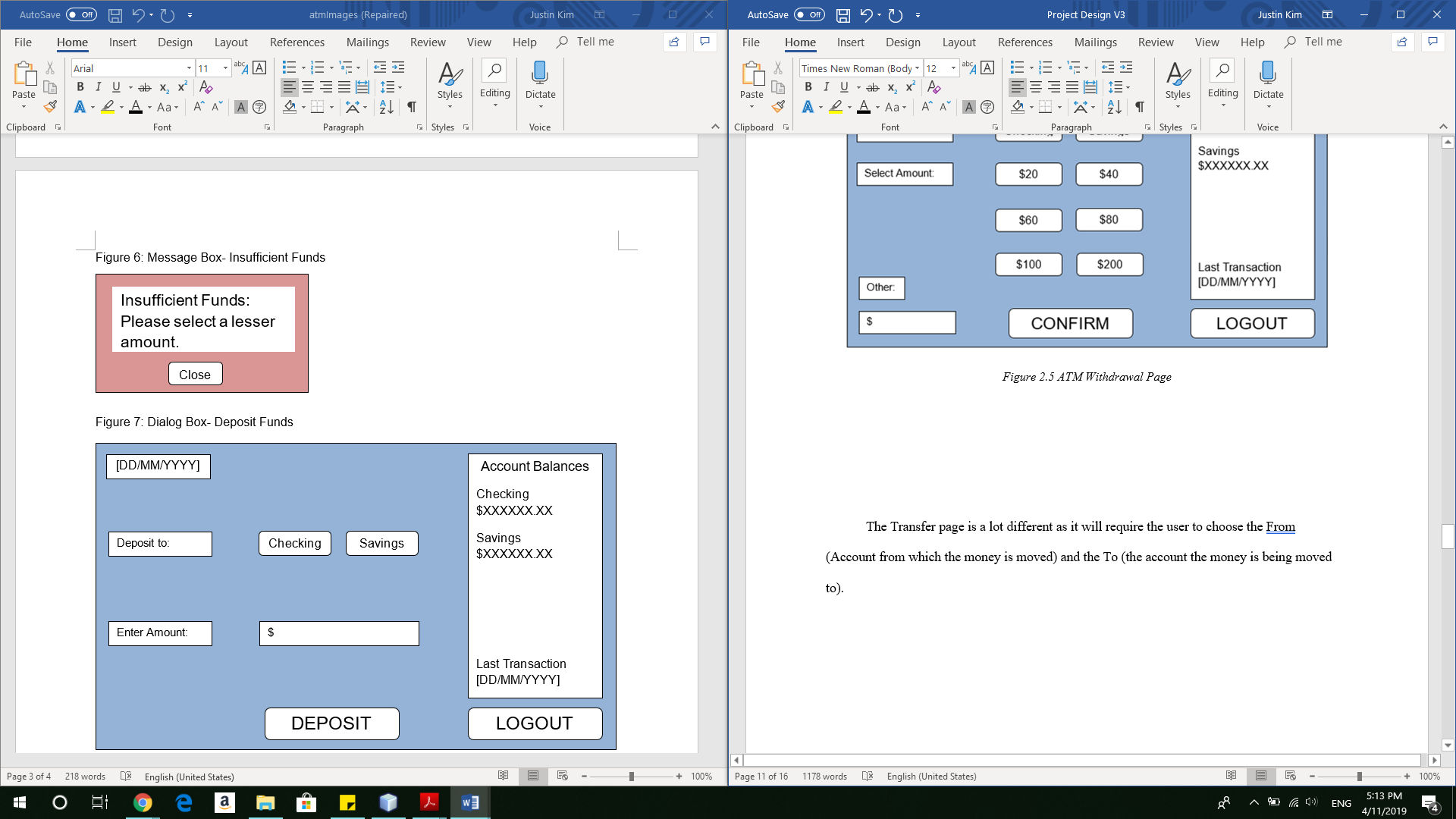


Figure 2.6 Insufficient Funds

The Transfer page (Figure 2.7) is a lot different as it will require the user to choose the From (Account from which the money is moved) and the To (the account the money is being moved to).

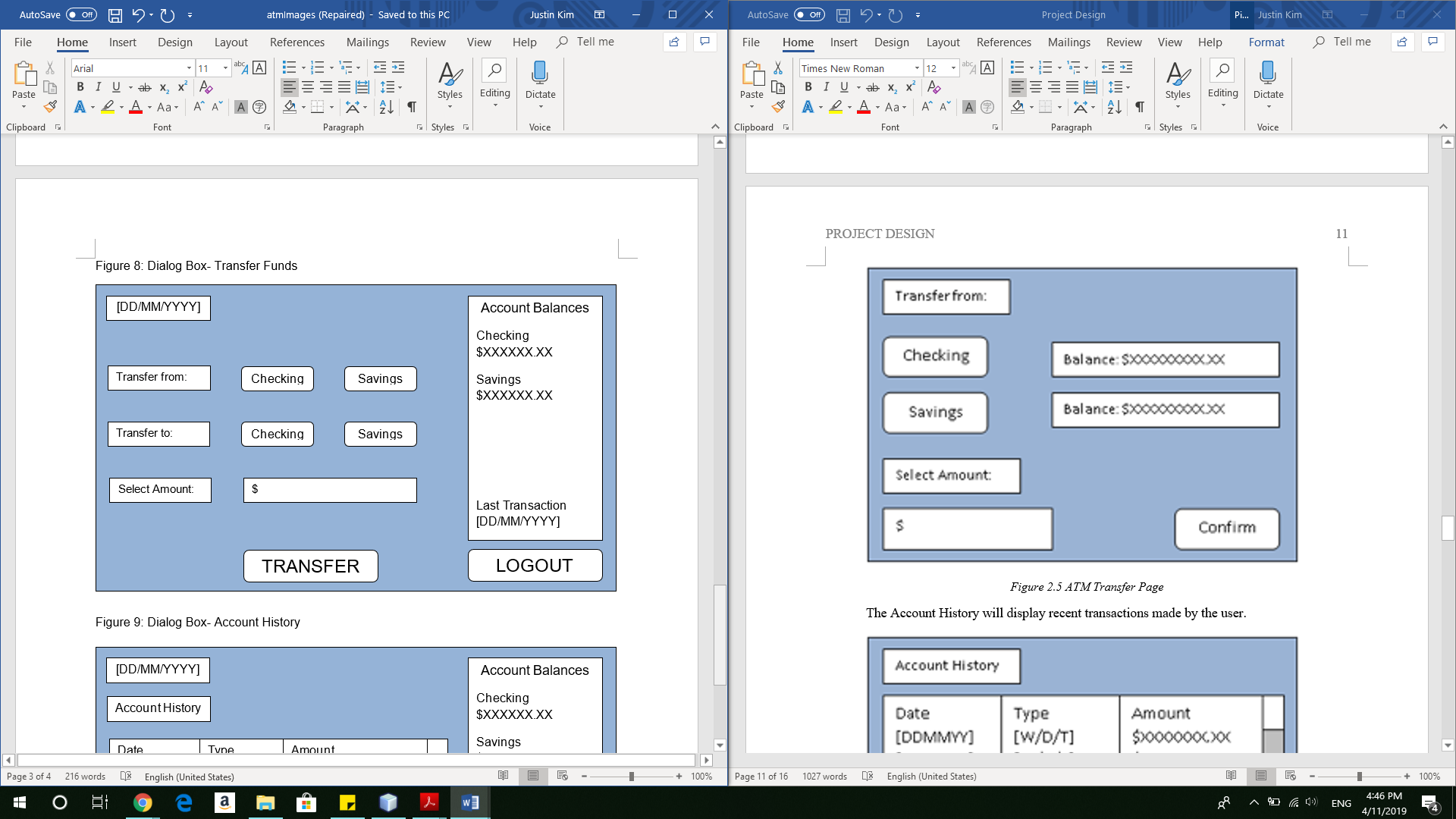


Figure 2.7 ATM Transfer Page

In Figure 2.8, the Account History page will display recent transactions made by the user, showing the date, type of transaction, and amount. There will be an option to return to the main page.

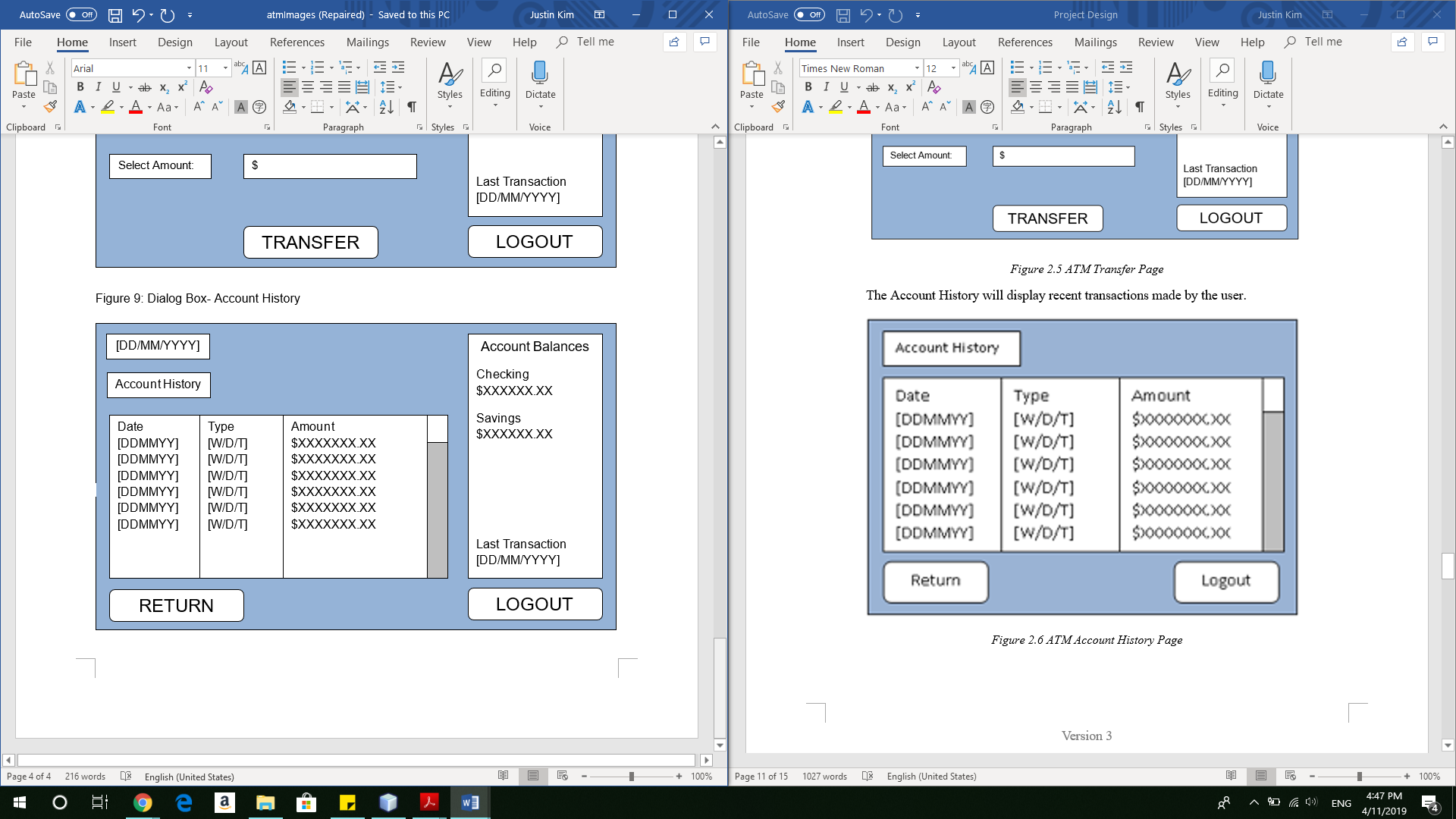


Figure 2.8 ATM Account History Page

# Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input | Expected Output | Actual Output | Result |
| No Username provided | Empty username | Message box stating account username is required |  |  |
| No password provided | Empty password box | Message box stating password required |  |  |
| Login with wrong username and password combination | XML file  Wrong username and password combination | Message box stating could not login |  |  |
| Non-numeric account balance on checking account | XML file | NumberFormatException |  |  |
| Non-numeric account balance on savings account | XML file | NumberFormatException |  |  |
| Username and password wrong 3 times | GUI input | Message box stating login failed. Return to Login page. Account locked. |  |  |
| Transfer $100.01 from checking to savings | GUI input  XML file | Checking account balance decrements by $100.01  Savings account increases by $100.01  XML file undated correctly |  |  |
| Transfer $100.01 from savings to checking | GUI input  XML file | Savings account balance decrements by $100.01  Checking account increases by $100.01  XML file undated correctly |  |  |
| Withdraw $100 from checking | GUI input  XML file | Checking account balance decrements by $100  XML file updated correctly |  |  |
| Withdraw $100 from savings | GUI input  XML file | Savings account balance decrements by $100  XML file updated correctly |  |  |
| Transfer attempt from Checking to Savings account with insufficient balance in Checking | GUI input | Message box stating transfer aborted because of insufficient balance |  |  |
| Transfer attempt from Savings to Checking account with insufficient balance in Savings | GUI input | Message box stating transfer aborted because of insufficient balance |  |  |
| Withdrawal attempt from Savings account with insufficient balance in Savings | GUI input | Message box stating withdrawal aborted because of insufficient balance |  |  |
| Withdrawal attempt from Checking account with insufficient balance in Checking | GUI input | Message box stating withdrawal aborted because of insufficient balance |  |  |
| Withdrawal attempt from Checking account with entered amount not multiple of 20s | GUI input | Message box telling the customer to enter an amount multiple of 20s |  |  |
| Withdrawal attempt from Savings account with entered amount not multiple of 20s | GUI input | Message box telling the customer to enter an amount multiple of 20s |  |  |
| Deposit $100.01 to Checking | GUI input  XML file | Checking account balance increments by $100.01  XML file updated correctly  Main page refreshes account balance |  |  |
| Deposit $100.01 to Savings | GUI input  XML file | Savings account balance increments by $100.01  XML file updated correctly  Main page refreshes account balance |  |  |
| Add a large number into the deposit to trigger a stackoverflow error | GUI input | StackOverflow Error |  |  |
| Withdraw more money than is in the ATM at that moment | GUI input | Message box alerting customer that transaction could not be completed |  |  |
| An attempt to withdraw $-20 | GUI input | Message box alerting customer to enter a positive amount |  |  |
| An attempt to deposit $-20 | GUI input | Message box alerting customer to enter a positive amount |  |  |
| All tests will be duplicated with each language used | GUI input  XML file | All languages are translated correctly. |  |  |