Design Document

Team PB-PJ

18 March 2018

Table 1: Team PB-PJ

Name
Matthew Ferderber
Matthew Dugal
Mylene Haurie
Viktoriya Malinova
Eric Morgan
Artem Khomich
Kai Nicoll-Griffith
Maxmilien Malderle

1 Introduction

The design document provides a detailed view into the design of the application and justifies the various design decisions made by the programmers. The architecture design section provides a high-level view of the architectural choices while the detailed design section focuses on the detailed implementation of the architecture and all aspects of the system. The Subsystem Interface Specification describes the two major Subsystems of the application and all of their components.

2 Architectural Design

The My Money application uses the Model View Controller architecture to seperate code into logical components (models, view, and controllers).

Models

Models are used to provide a simple interface for the data used by the application uses. Whenever changes are made in the model (by the controller), the view is notified and updated.

Views

Views are the visual representation of the models. Any data that needs to be shown to the user is given to the view through models and is updated when changes are made to the model. The view can also be updated by the controller. When an action is performed by the user, the view delivers the action to the controller which provides the logic behind the requested action.

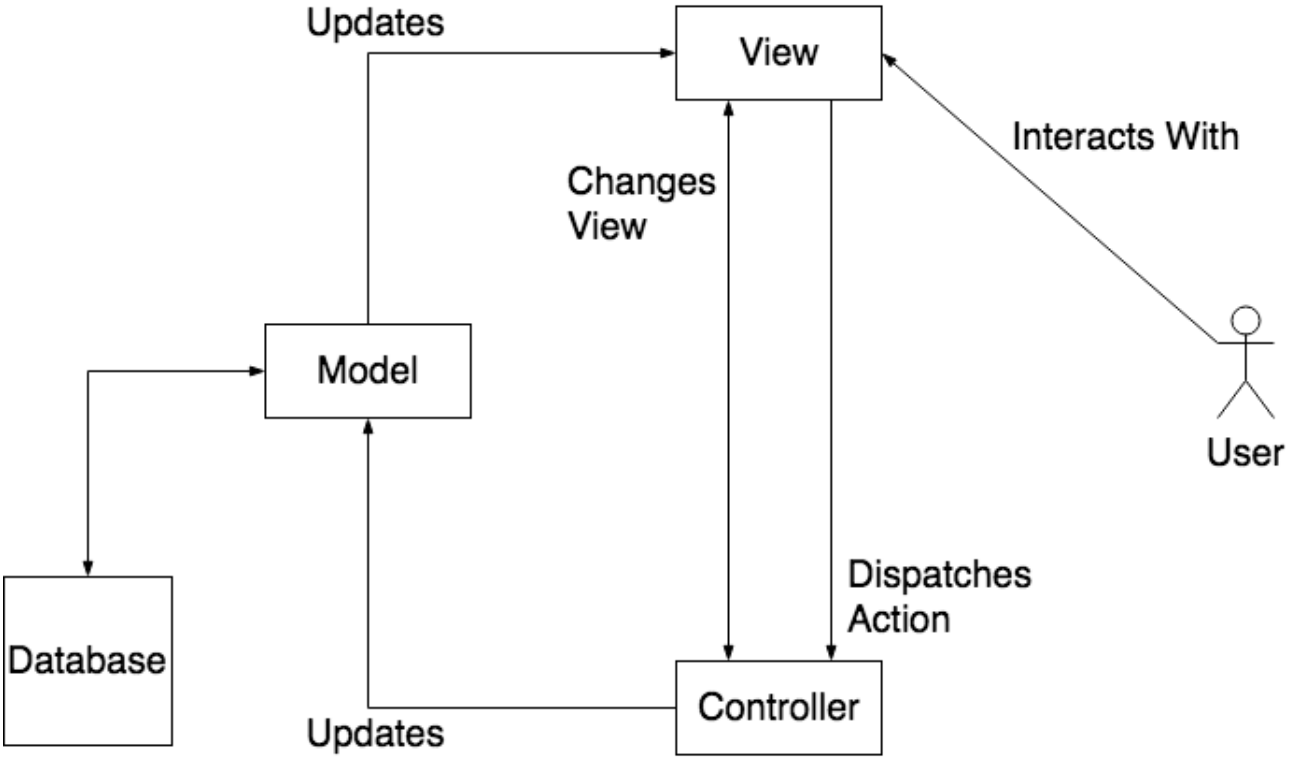
Controllers

Controllers provide the logic that allows for interaction between the user and the application. When an action is performed by the user it is relayed from the view to the responsible controller. The controller uses its internal state to decide what the outcome of the action should be. When the outcome is decided, the controller can update which view the application displays or the model that is bound to the current view.

Benefits

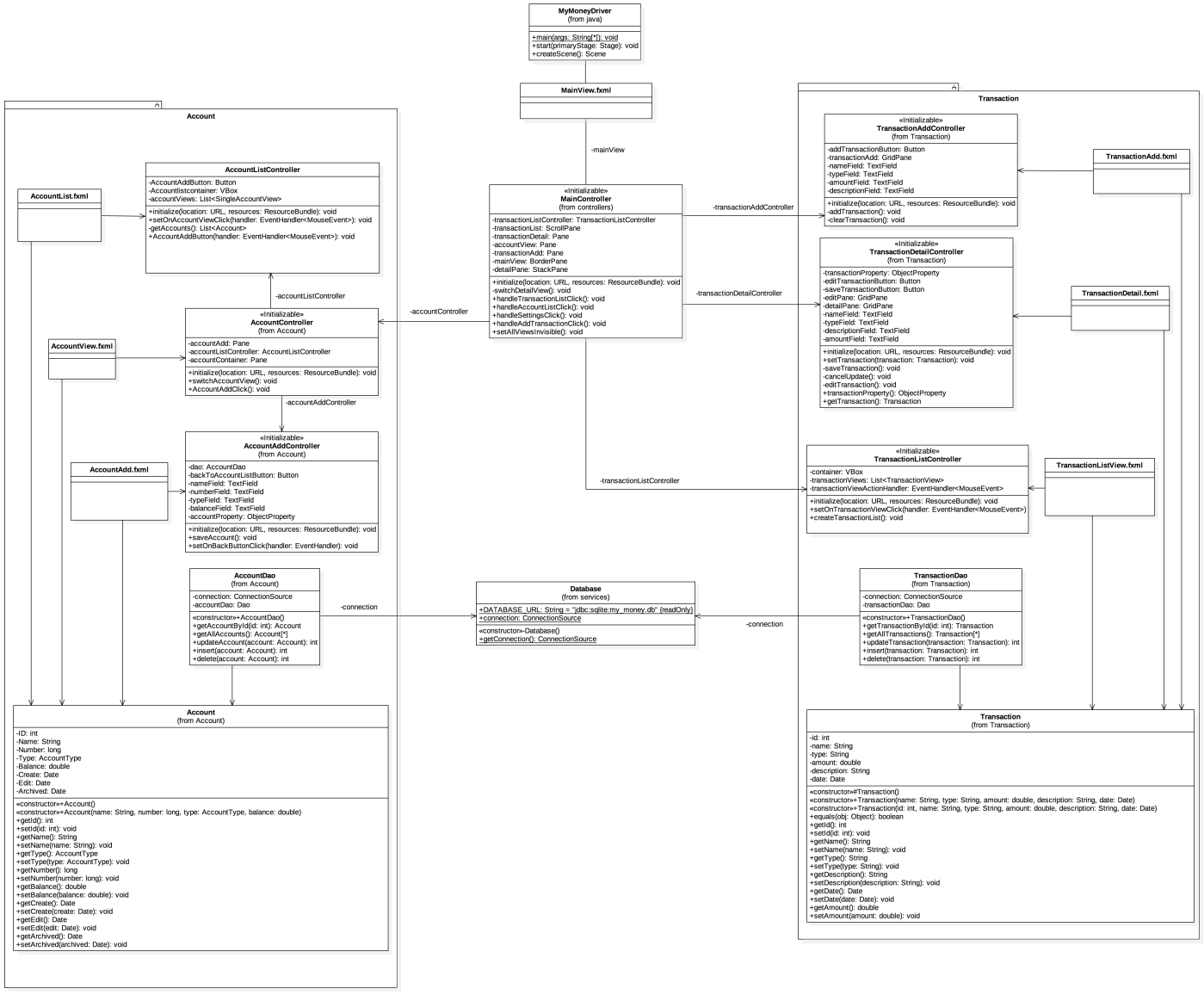
MVC provides many benefits for the application. It allows for seperation of display, logic and data access into components. Because components are seperated, the same

portions of the project can be worked on by multiple people in parallel. Without having to worry about the status of the whole application, a view can be updated to display data differently, the model can be modified to contain more data, and the controller can handle new actions.



This diagram shows the interaction between the models, views, and controllers. As described above, the user interacts with the view which displays data from the model. The view can send actions to the controller which prompts an update of the view/model. The model (and Data Access Objects) interact with the database to update and retrieve data.

2.1 Architectural Diagram



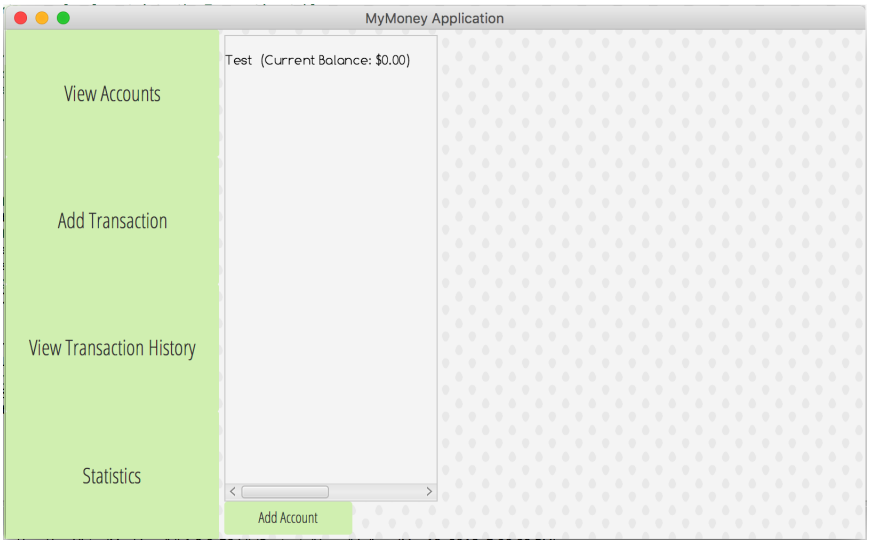
The MVC architecture of the system allows for the project to be modularized into multiple submodules. This diagram shows the Account and Transaction submodules which make up the main functionality of the system. Organizing the system like this allows for major re-use of components such as the models (Account and Transaction) which are used for every view in their respective module.

2.2 Subsystem Interface Specifications

Account Subsystem Interface

The Account Subsystem is the system that allows the user to manage their accounts used to store transactions. Accounts are typically used to represent real bank accounts or similar real-world accounts that transactions can take place with. There are three main views associated with the account system. The detailed view, list view, and add view.

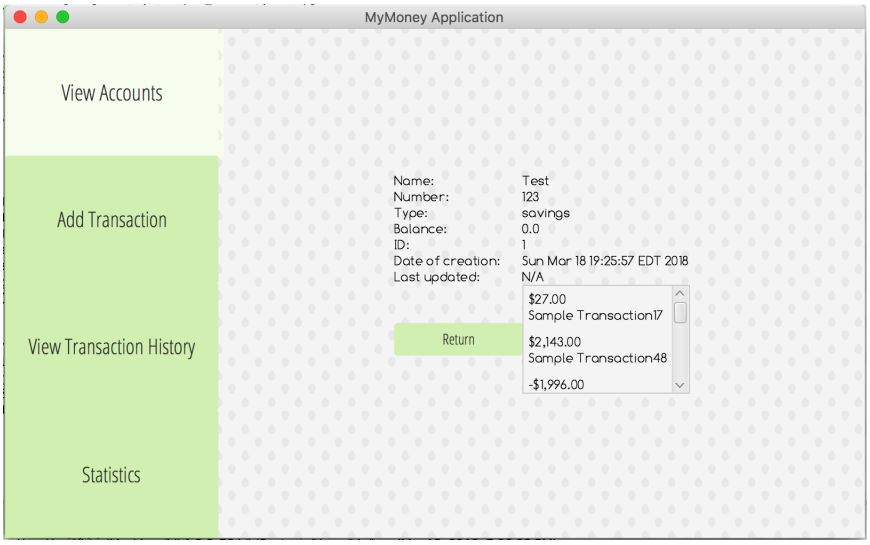
Account List View



The Account list is responsible for showing an up-to-date list of all of the accounts in the application. From the Account List, the user can enter into the Detailed view and the Add view by double clicking on a list item or clicking on the add Account button respectively.

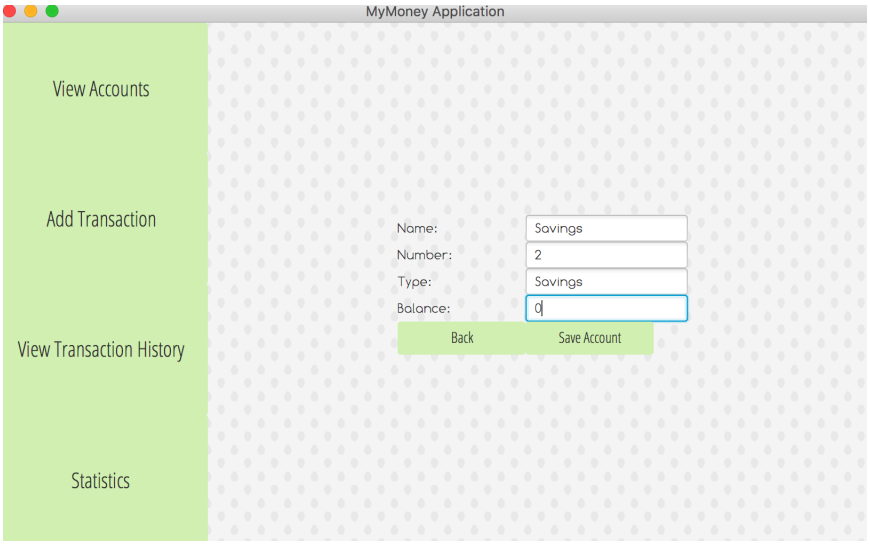
The AccountListController is used to manage switching between views related to the Account List (as seen in the class diagram). The AccountListController will either call the "AccountViewActionHandler" or the "AccountAddClick" event handlers when the view is interacted with by the user.

Account Detailed View



The Detailed view represents one Account and is responsible for showing all of the details associated with the selected account. Due to the informational nature of the detail view, there are no actions the user can perform in the view.

Account Add View



The Account Add view is used for creating new accounts in the system. When creating an account, the user enters the relevant account details (Account name, number, account type, and initial balance).

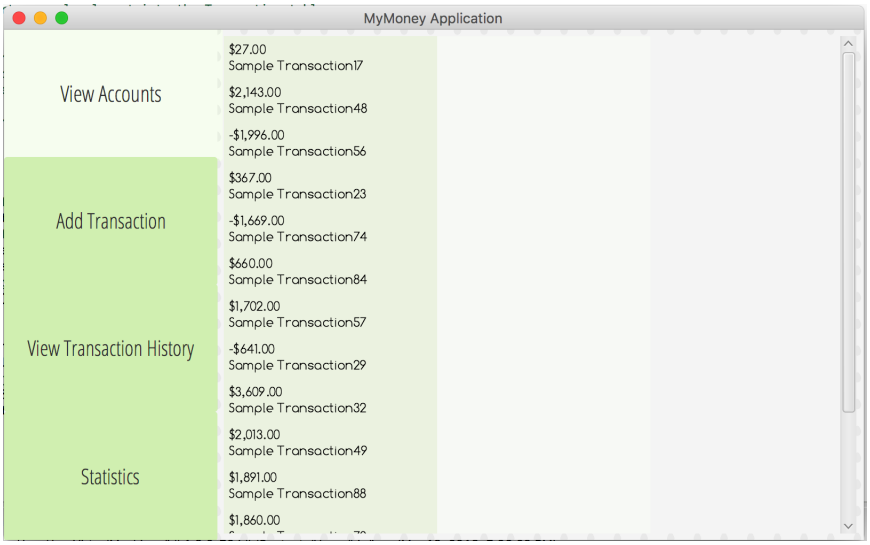
When the user saves the account, the AccountAddController is passed the action by the AccountAddView. The "saveAccount" method is called which retrieves the new accounts

details from the view and inserts it into the database using the AccountDAO. Once inserted, the user is returned to the Account List view where they can continue interacting with the system.

Transaction Subsystem Interface

The Transaction Subsystem allows the user to add, delete, and edit transactions associated with an account. Transactions represent real-world currency transactions and have a Name, type, amount, description, and associated account. Transactions have three main views. The Transaction list view, detail view, and add view.

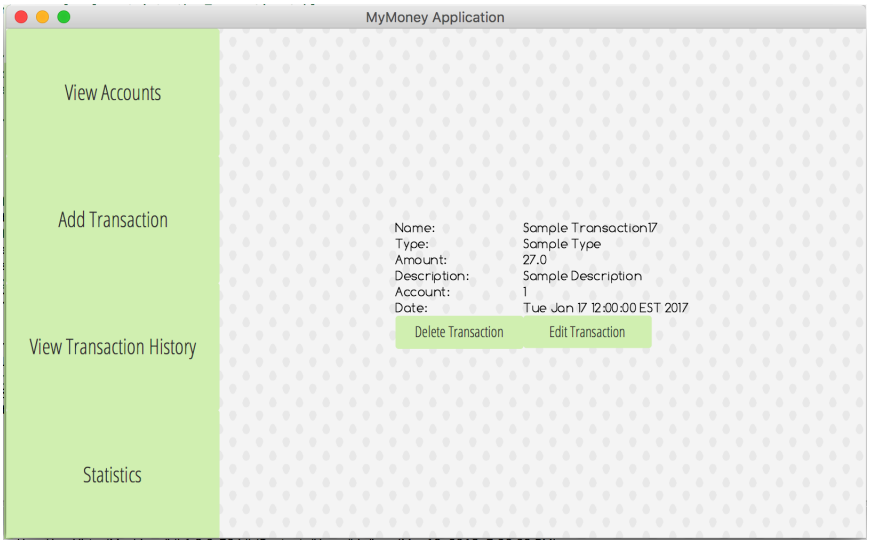
Transaction List View



The Transaction List is used to show all of the transactions and a brief description of what they represent (amount and name). The list is expandable using the scroll wheel which allows the user to view all of their transactions sorted by date in descending order.

From the Transaction List, the user can click on any transaction to be brought to the Detailed Transaction View. When a transaction is clicked, the event handler passed to the TransactionListController by the MainController is executed which the main view uses to decide which transaction to create in the TransactionDetail. In this way, the MainController acts as a dispatcher which takes events and handles the high-level event of displaying different views.

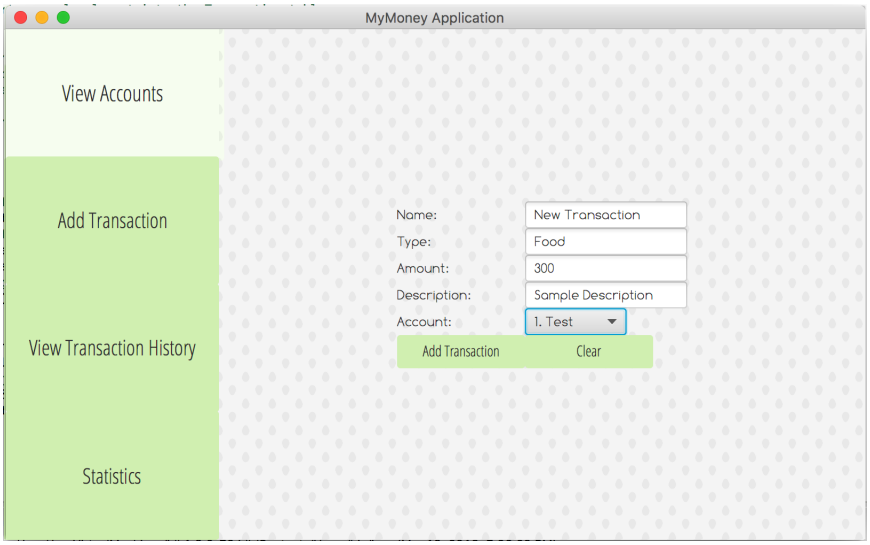
Transaction Detail View



The Transaction Detail view displays a full description of the selected transaction. From this view, the user can see the creation date, and all of the original data from the creation of the transaction.

From the Transaction Detail, the user has two actions: Delete and Edit. When the user presses delete, the TransactionDetailController's "deleteTransaction" method is called and the transaction is deleted. Once deleted, the user is returned to the Transaction List view. When the Edit action is invoked, the TransactionDetailController's "editTransaction" method is called which changes the view to allow all fields to be edited.

Transaction Add View



The Transaction Add view allows the user to add new Transactions to specified accounts. From the Add view, the user can enter the name, type, amount, description, and associated account of a transaction and insert it into the database.

From the Add view, the user is presented with two actions: Add and Clear. When Clear is pressed, the TransactionAddController's "clearTransaction" method is called which removes any data filled into the view by the user. The add action calls the "addTransaction" method of the TransactionAddController to insert a new Transaction entry into the database using the TransactionDAO.

3 Detailed Design

Complete description of the system design, describing one subsystem separately in respective subsection. UML class diagrams are to be used, as well as a short textual description describing the purpose of each class.

3.1 Module $\langle Model \rangle$:

3.1.1 Detailed Design Diagram

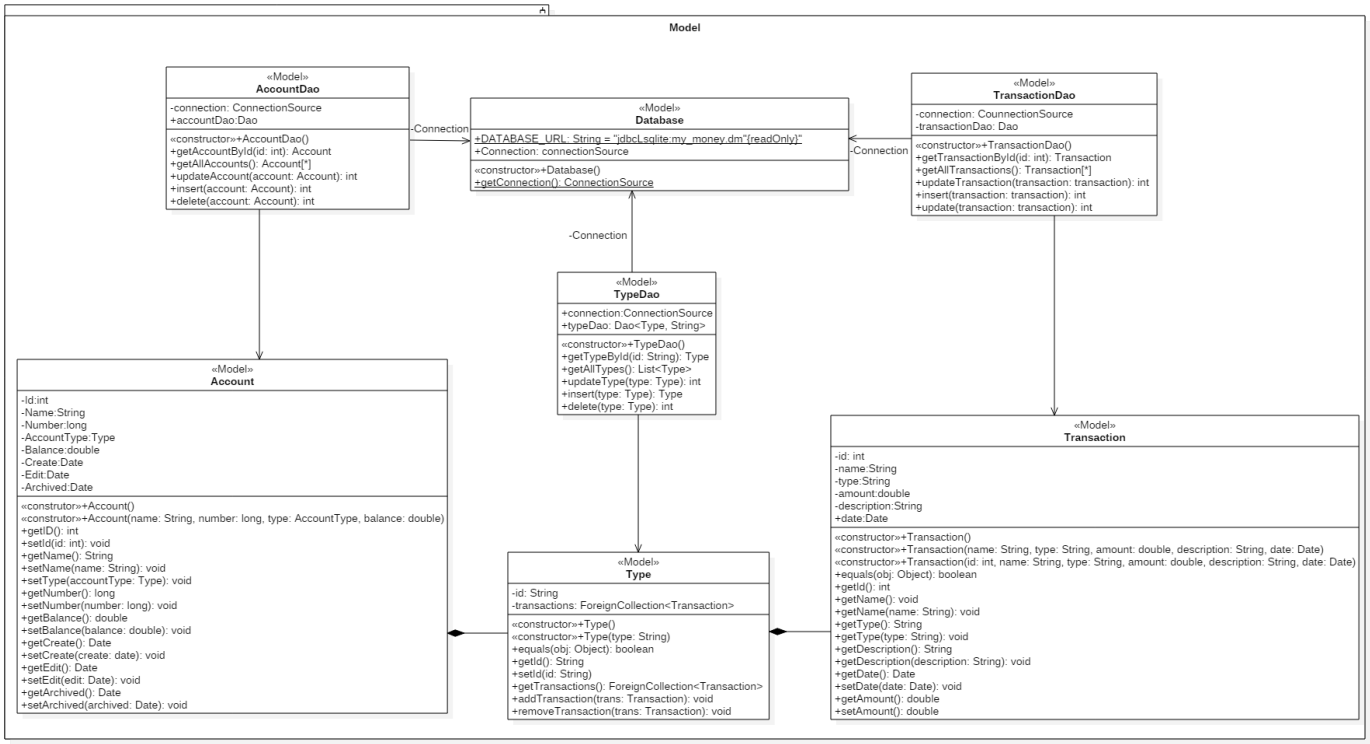


Figure 1: Model Module Class Diagram

The module of the model classes serves primarily to populate and create container objects. 'Account', 'Type' and 'Transaction' all have a 'Data Access Object' which draws information from the central database service using SQL queries. All information is used to generate views through access to Dao's by controllers.

3.1.2 Units Description

3.1.2.1 Class Transaction:

<table>
<tr><td>Class Name</td><td colspan="4">Transaction</td></tr>
<tr><td>Description</td><td colspan="4">The Transaction class holds information on financial data that is to be sent to the server, processed by statistics, deleted or anything requiring data manipulation.</td></tr>
<tr><td rowspan="7">Attributes</td><td>Visibility</td><td>Data Type</td><td>Name</td><td>Description</td></tr>
<tr><td>Private</td><td>int</td><td>id</td><td>An id of the transaction</td></tr>
<tr><td>Private</td><td>String</td><td>name</td><td>Name of user</td></tr>
<tr><td>Private</td><td>String</td><td>type</td><td>The type of action done</td></tr>
<tr><td>Private</td><td>double</td><td>amount</td><td>The quantity acted on</td></tr>
<tr><td>Private</td><td>String</td><td>description</td><td>A description of what it is</td></tr>
<tr><td>Private</td><td>Date</td><td>date</td><td>The date of the transaction</td></tr>
<tr><td rowspan="20">Methods</td><td>Visibility</td><td colspan="2">Name</td><td>Description</td></tr>
<tr><td>public</td><td colspan="2">Transaction()</td><td>default constructor</td></tr>
<tr><td>public</td><td colspan="2">Transaction(String name, String type, double amount, String description, Date date)</td><td>argument constructor</td></tr>
<tr><td>public</td><td colspan="2">Transaction(int id, String name, String type, double amount, String description, Date date)</td><td>argument constructor with given id</td></tr>
<tr><td>public</td><td colspan="2">equals(Object obj)</td><td>Return a boolean from comparison</td></tr>
<tr><td>public</td><td colspan="2">getId()</td><td>return ID</td></tr>
<tr><td>public</td><td colspan="2">setId(int id)</td><td>set the ID</td></tr>
<tr><td>public</td><td colspan="2">getName()</td><td>get the name</td></tr>
<tr><td>public</td><td colspan="2">setName(String name)</td><td>set the name</td></tr>
<tr><td>public</td><td colspan="2">getType()</td><td>get the type</td></tr>
<tr><td>public</td><td colspan="2">setType(String type)</td><td>set the type</td></tr>
<tr><td>public</td><td colspan="2">getDescription()</td><td>get the description</td></tr>
<tr><td>public</td><td colspan="2">setDescription(String description)</td><td>set the description</td></tr>
<tr><td>public</td><td colspan="2">getDate()</td><td>get the date</td></tr>
<tr><td>public</td><td colspan="2">setDate(Date date)</td><td>set the date</td></tr>
<tr><td>public</td><td colspan="2">getAmount()</td><td>get the amount</td></tr>
<tr><td>public</td><td colspan="2">setAmount(double amount)</td><td>set the amount</td></tr>
</table>

3.1.2.2 Transaction Methods:

Method Name	Transaction()
Class Name	Transaction
Functionality	default constructor
Input	-
Output	Transaction Class
Pseudo Code	-

Method Name	Transaction(String name, String type, double amount, String description, Date date)
Class Name	Transaction
Functionality	Argument Constructor
Input	Private field variables
Output	Transaction Class
Pseudo Code	*Set all class data fields to argument values

Method Name	Transaction(int id, String name, String type, double amount, String description, Date date)
Class Name	Transaction
Functionality	Argument Constructor with id paramater
Input	Private field variables
Output	Transaction Class
Pseudo Code	BEGIN *Set all class data fields to argument values END

Method Name	equals(Object obj)
Class Name	Transaction
Functionality	Return boolean based on variable comparision of Transaction objects
Input	-
Output	Boolean of transaction comparision
Pseudo Code	BEGIN Transaction t = (Transaction) obj RETURN (t NOT null AND id EQUALS t.id AND name.equals(t.name) AND type.equals(t.type) AND description.equals(t.description) AND amount EQUALS t.amount AND date.equals(t.date)) END

Method Name	getId()
Class Name	Transaction
Functionality	return ID paramater
Input	-
Output	ID value
Pseudo Code	BEGIN return this.id END

Method Name	setId(int ID)
Class Name	Transaction
Functionality	sets the ID paramater
Input	ID integer
Output	-
Pseudo Code	BEGIN this.id = ID END

Method Name	getName()
Class Name	Transaction
Functionality	return account holder's name
Input	ID -
Output	Account holder's name
Pseudo Code	BEGIN return this.name END

Method Name	setName(String name)
Class Name	Transaction
Functionality	change the account holder's name
Input	ID Name string
Output	-
Pseudo Code	BEGIN this.name = name END

Method Name	getType()
Class Name	Transaction
Functionality	Get the account type
Input	-
Output	Type string
Pseudo Code	BEGIN return this.type END

Method Name	setType(String type)
Class Name	Transaction
Functionality	change the account type
Input	Type string
Output	-
Pseudo Code	BEGIN this.type = name END

Method Name	getDescription()
Class Name	Transaction
Functionality	Return the description
Input	-
Output	Description string
Pseudo Code	BEGIN return this.description END

Method Name	setDescription(String description)
Class Name	Transaction
Functionality	Set the description
Input	Description string
Output	-
Pseudo Code	BEGIN this.description = description END

Method Name	getDate()
Class Name	Transaction
Functionality	Get the date of the account
Input	-
Output	Date object of transaction
Pseudo Code	BEGIN return this.date END

Method Name	setDate(Date date)
Class Name	Transaction
Functionality	Set the date of the account
Input	Date object of transaction
Output	-
Pseudo Code	BEGIN this.date = date END

Method Name	getAmount()
Class Name	Transaction
Functionality	Get the amount returned.
Input	-
Output	return quantity transacted as a double
Pseudo Code	BEGIN return this.amount END

Method Name	setAmount(double amount)
Class Name	Transaction
Functionality	Set the amount returned.
Input	amount transacted as a double
Output	-
Pseudo Code	BEGIN this.amount = amount END

3.1.2.3 Class TransactionDao:

Class Name	TransactionDao			
Description	A transaction Dao is the way that transaction connect to the database. A Transaction Dao has reference to a set of Transaction objects and performs operations on networks with them.			
Attributes	Visibility	Data Type	Name	Description
	private	ConnectionSource	connection	The address of a connection
	private	Dao $\langle Transaction, Integer \rangle$	transactionDao	the Transaction it is connected to
Methods	Visibility	Name	Description	
	public	TransactionDao()	Default constructor	
	public	getTransactionById(int id)	Return the record with a certain ID	
	public	getAllTransactions(int number)	Return all Transaction objects contained	
	public	updateTransaction(Transaction transaction)	updates an sql record	
	public	insert(Transaction transaction)	inserts a transaction	
	public	delete(Transaction transaction)	Removes a transaction	

3.1.2.4 TransactionDao Methods:

Method Name	TransactionDao()
Class Name	TransactionDao
Functionality	default constructor
Input	-
Output	TransactionDao Class
Pseudo Code	-

Method Name	getTransactionById(int id)
Class Name	TransactionDao
Functionality	Get from the server a transaction of a certain ID
Input	Id to search for
Output	Transaction class found
Pseudo Code	BEGIN SQL Query the Database for this.id END

Method Name	getAllTransactions()
Class Name	TransactionDao
Functionality	Returns a List of all Transactions in the database
Input	-
Output	Transaction List
Pseudo Code	BEGIN SQL Query the Database for any values END

Method Name	updateTransaction(Transaction transaction)
Class Name	TransactionDao
Functionality	Find a transaction by ID and update the records
Input	Transaction to update
Output	integer success code
Pseudo Code	BEGIN SQL Query the Database to update certain ID values END

Method Name	insert(Transaction transaction)
Class Name	TransactionDao
Functionality	Inserts a record into the database
Input	Transaction to insert
Output	Integer success code
Pseudo Code	BEGIN SQL Query the Database to insert certain Transactions END

Method Name	delete(Transaction transaction)
Class Name	TransactionDao
Functionality	Delete a record from the database
Input	Transaction to delete
Output	Integer success code
Pseudo Code	BEGIN SQL Query the Database to delete certain Transactions END

3.1.2.5 Class Type

Class Name	Type			
Description	The Type class represents the different sorts of account types users can create. Types are associated with transactions in that they hold a ForeignCollection of Transactions associated with the Type			
Attributes	Visibility	Data Type	Name	Description
	private	String	id	The unique identifier of a Type
	private	ForeignCollec	transactions	A set of transactions associated with the class.
Methods	Visibility	Name	Description	
	public	Type()	Default constructor	
	public	Type(String type)	Constructor specifying Type	
	public	equals(Object obj)	Checks the equivalence against this Object and argument Object	
	public	getId()	Returns Id string	
	public	setId(String id)	Sets the Id string	
	public	getTransactions()	Returns the set of transactions associated with class	
	public	addTransaction(Transaction trans)	Adds another transaction to the class	
	public	removeTransaction(Transacti trans)	Removes a transaction associated with the class	

3.1.2.6 Type Methods:

Method Name	Type()
Class Name	Type
Functionality	Default constructor to create a Type object
Input	-
Output	Type Object
Pseudo Code	BEGIN END

Method Name	Type(String type)
Class Name	Type
Functionality	Argument constructor to create a Type object
Input	Type string
Output	Type Object
Pseudo Code	BEGIN this.type = type END

Method Name	equals(Object obj)
Class Name	Type
Functionality	Check equivalence between objects
Input	Object to compare
Output	Boolean of comparison
Pseudo Code	BEGIN Type t = (Type) obj RETURN t NOT null AND id.equals(t.id) END

Method Name	getID()
Class Name	Type
Functionality	Return ID of Type
Input	-
Output	Type ID
Pseudo Code	BEGIN return this.id END

Method Name	setId()
Class Name	Type
Functionality	Set ID value of type
Input	Type ID
Output	-
Pseudo Code	BEGIN this.id = id END

Method Name	getTransactions()
Class Name	Type
Functionality	Returns transactions associated with the type
Input	-
Output	Foreign Collection of Transactions
Pseudo Code	BEGIN return this.transactions END

Method Name	addTransaction(Transaction trans)
Class Name	Type
Functionality	Adds a transaction value to the transaction array
Input	Transaction to add
Output	-
Pseudo Code	BEGIN transactions.add(trans) END

Method Name	removeTransaction(Transaction trans)
Class Name	Type
Functionality	Remove a transaction value from the transaction array
Input	Transaction to remove
Output	-
Pseudo Code	BEGIN transactions.remove(trans) END

3.1.2.7 Class TypeDao

Class Name	TypeDao			
Description	The TypeDao is the class which communicates Type information back and forth with the database.			
Attributes	Visibility	Data Type	Name	Description
	public	connection	ConnectionSource	A connection point with the database
	public	Dao$\langle Type, String \rangle$	typeDao	A set of Type's being connected to the database
Methods	Visibility	Name	Description	
	public	TypeDao()	Default constructor	
	public	getTypeById(String id)	Gets Types with certain Ids	
	public	getAllTypes()	Gets all data	
	public	updateType(Type type)	update the Type with the server	
	public	insert(Type type)	insert a Type with the server	
	public	delete(Type type)	delete the Type with the server	

3.1.2.8 TypeDao Methods:

Method Name	TypeDao()
Class Name	TypeDao
Functionality	Default constructor to create a TypeDao object
Input	-
Output	TypeDao Object
Pseudo Code	-

Method Name	getTypeById(String id)
Class Name	Type
Functionality	Returns all types of certain Id
Input	Id to find
Output	Type Object
Pseudo Code	BEGIN SQL Query database for Id string END

Method Name	getAllTypes()
Class Name	Type
Functionality	Returns all Types in database
Input	-
Output	List of Type Object
Pseudo Code	BEGIN SQL Query database for all Types END

Method Name	updateType(Type type)
Class Name	Type
Functionality	Updates types of certain Id
Input	Type to find
Output	Integer success code
Pseudo Code	BEGIN SQL Query database to update Type END

Method Name	insert(Type type)
Class Name	Type
Functionality	Inserts Types into database
Input	Type to insert
Output	integer success code
Pseudo Code	BEGIN SQL Query to insert data into database END

Method Name	delete(Type type)
Class Name	Type
Functionality	Delete Types from database
Input	Type to remove
Output	integer success code
Pseudo Code	BEGIN SQL Query to remove data from database END

3.1.2.9 Class Account

Class Name	Account			
Description	An account holds a user's specific information on who they are and what their overall finances are like. It is connected to the database by AccountDao			
Attributes	Visibility	Data Type	Name	Description
	private	int	id	The unique identifier of Account
	private	String	Name	Name of the account holder
	private	long	Number	Account number
	private	Type	AccountType	Type of account
	private	double	Balance	Holdings of account
	private	Date	Create	Date created of account
	private	Date	Edit	Date edited of account
	private	Date	Archived	Date archived of account
Methods	Visibility	Name	Description	
	public	Account()	Default Account constructor	
	public	Account(name:String, number:long, type:AccountType, balance:double)	Argument Account constructor	
	public	getId()	Returns unique identifier	
	public	setId(int id)	Sets unique identifier	
	public	getName()	Returns name of account holder	
	public	setName(name:String)	sets name of account holder	
	public	getType()	sets the account Type	
	public	setType(accountType:Type)	sets the account Type	
	public	getNumber()	Returns account number	
	public	setNumber(number: long)	Sets account number	
	public	getBalance()	Returns account holdings	
	public	setBalance(balance:double)	Sets the account balances	
	public	getCreate()	Get the creation date	
	public	setCreate(create:Date)	Set the creation date	
	public	getEdit()	Gets edited date	
	public	setEdit(edited:Date)	Sets edited date	
	public	getArchived()	Returns date archived	
	public	setArchived(archived:Date)	Sets date archived	

3.1.2.10 Account Methods:

Method Name	Account()
Class Name	Account
Functionality	Default Account constructor
Input	-
Output	Account Object
Pseudo Code	-

Method Name	Account(name:String, number:long, type:AccountType, balance:double)
Class Name	Account
Functionality	Argument Account constructor
Input	Private fields to be set
Output	Account Object
Pseudo Code	BEGIN Set each private field with the respective arguments END

Method Name	getId()
Class Name	Account
Functionality	Gets account ID
Input	-
Output	Account Id
Pseudo Code	BEGIN return this.id END

Method Name	setId(int id)
Class Name	Account
Functionality	Sets account ID
Input	Account Id
Output	-
Pseudo Code	BEGIN this.id = id END

Method Name	getName()
Class Name	Account
Functionality	Gets account Name
Input	-
Output	Account Name
Pseudo Code	BEGIN return this.name END

Method Name	setName(String name)
Class Name	Account
Functionality	Sets account Name
Input	Account Name
Output	-
Pseudo Code	BEGIN this.name = name END

Method Name	getType()
Class Name	Account
Functionality	Gets account Type
Input	-
Output	Account Type
Pseudo Code	BEGIN return this.type END

Method Name	setType(Type type)
Class Name	Account
Functionality	Sets account Type
Input	Account Type
Output	-
Pseudo Code	BEGIN this.type = type END

Method Name	getNumber()
Class Name	Account
Functionality	Gets account number
Input	-
Output	Account Number
Pseudo Code	BEGIN return this.number END

Method Name	getNumber(long number)
Class Name	Account
Functionality	Sets account number
Input	Account Number
Output	-
Pseudo Code	BEGIN this.number = number END

Method Name	setNumber()
Class Name	Account
Functionality	Sets account number
Input	Account Number
Output	-
Pseudo Code	BEGIN this.number = number END

Method Name	getBalance()
Class Name	Account
Functionality	Gets account balance
Input	-
Output	Account Balance
Pseudo Code	BEGIN return this.balance END

Method Name	setBalance()
Class Name	Account
Functionality	Sets account balance
Input	Account Balance
Output	-
Pseudo Code	BEGIN this.balance = balance END

Method Name	getCreate()
Class Name	Account
Functionality	Gets account creation Date
Input	-
Output	Account creation Date
Pseudo Code	BEGIN return this.create END

Method Name	setCreate(create)
Class Name	Account
Functionality	Sets account creation Date
Input	Account creation Date
Output	-
Pseudo Code	BEGIN this.create = create END

Method Name	getEdit()
Class Name	Account
Functionality	Gets account edit Date
Input	-
Output	Account edit Date
Pseudo Cod e	BEGIN return this.edited END

Method Name	setEdit(edited)
Class Name	Account
Functionality	Sets account edit Date
Input	Account edit Date
Output	-
Pseudo Code	BEGIN this.edited = edited END

Method Name	getArchived()
Class Name	Account
Functionality	Gets account archived Date
Input	-
Output	Account archive Date
Pseudo Code	BEGIN return this.archived END

Method Name	setArchived(archived)
Class Name	Account
Functionality	Sets account archived Date
Input	Account archive Date
Output	-
Pseudo Code	BEGIN this.archived = archived END

Class Name	AccountDao			
Description	An Account Dao is the way that account info is transmitted to and from the database.			
Attributes	Visibility	Data Type	Name	Description
	private	ConnectionSource	connection	The address of a connection
	private	Dao	AccountDao	The Dao connecting to the database
Methods	Visibility	Name	Description	
	public	AccountDao()	Default constructor	
	public	getAccountById(id:int)	Return the account with a certain ID	
	public	getAllAccounts()	Return an array of all accounts	
	public	updateAccount(account:Account)	Updates an sql record	
	public	insert(account:Account)	Inserts an account	
	public	delete(account:Account)	Removes an account	

3.1.2.11 Class AccountDao

3.1.2.12 AccountDao Methods:

Method Name	AccountDao()
Class Name	AccountDao
Functionality	Default AccountDao constructor. Connects to database and sets connection paramaters.
Input	-
Output	AccountDao object
Pseudo Code	-

Method Name	getAccountById(id:int)
Class Name	AccountDao
Functionality	Get and account of certain ID from database
Input	Id to search
Output	Account found
Pseudo Code	BEGIN SQL Query to find date from database END

Method Name	getAllAccounts()
Class Name	AccountDao
Functionality	Get all accounts from database
Input	Accounts to search
Output	List of accounts found
Pseudo Code	BEGIN SQL Query to find account from database END

Method Name	updateAccount(Account account)
Class Name	AccountDao
Functionality	Update account in database
Input	Account to update
Output	Integer success code
Pseudo Code	BEGIN SQL Query to find account from database END

Method Name	insert(Account account)
Class Name	AccountDao
Functionality	Insert account in database
Input	Account to update
Output	Integer success code
Pseudo Code	SQL Query to insert account into database END

Method Name	delete(Account account)
Class Name	AccountDao
Functionality	Delete account from database
Input	Account to Delete
Output	Integer success code
Pseudo Code	BEGIN SQL Query to Delete account from database END

3.1.2.13 Class Database

Class Name	Database			
Description	The database contains all information for users from accounts and types to transactions.			
Attributes	Visibility	Data Type	Name	Description
	static public	String	DATABASE_URL	the URL of the database
	public	ConnectionSource	connection	The address of a connection
Methods	Visibility	Name	Description	
	public	Database()	Default constructor	
	public	getConnection()	return connection to the database	

3.1.2.14 Database Methods:

Method Name	Database()
Class Name	Database
Functionality	Default Database constructor.
Input	-
Output	Database object
Pseudo Code	-

Method Name	getConnection()
Class Name	Database
Functionality	returns a connection to the database.
Input	-
Output	Database connection
Pseudo Code	return connection

3.2 Module $\langle View \rangle$:

3.2.1 Detailed Design Diagram

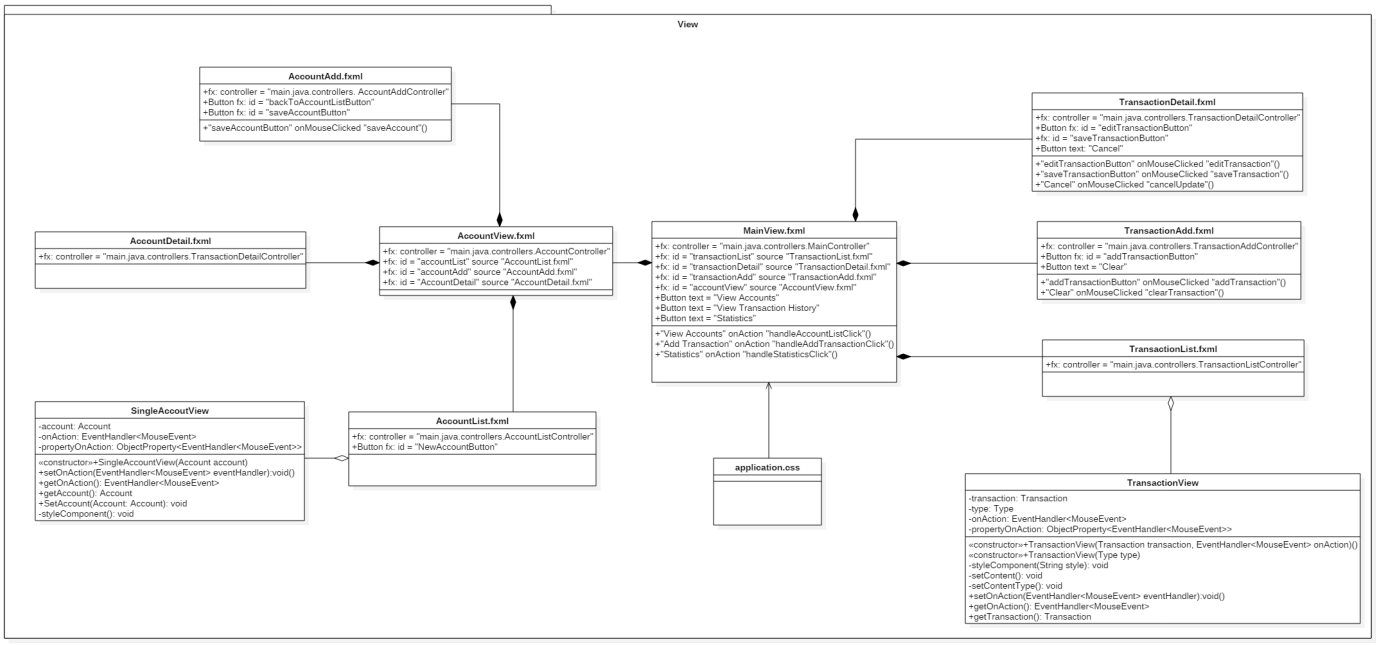


Figure 2: View Module Class Diagram

The module of the view classes serves as a way to allow the user to visualize his/her data and allow him/her to manipulate it in the ways the application allows. By clicking on screen objects the user can transfer to different views. Views are .fxml files generated by a FXMLLoader class.

3.2.2 Units Description

3.2.2.1 XML MainView.fxml

Class Name	MainView.fxml			
Description	MainView.fxml is the XML file that generates the main navigation screen. It contains within it as sources the other views and shifts to them when certain controller functions are called			
Attributes	Visibility	Data Type	Name	Description
	public	fx:controller	main.java.controllers.MainController	Reference to controller
	public	fx:id	"transactionList"source"TransactionI	Id of item containing the Transactionlist.fxml
	public	fx:id	"transactionDetail"source"Transactio	Id of item containing the TransactionDetail.fxml
	public	fx:id	"transactionAdd" source "TransactionAdd.fxml"	Id of item containing the TransactionAdd.fxml
	public	fx:id	"accountView" source "AccountView.fxml"	Id of item containing the AccountView.fxml
	public	Button	"View Accounts"	A button that can be pressed shifting to account view
	public	Button	"View Transaction History"	button that can be pressed shifting to a transaction view
	public	Button	"Statistics"	button that can be pressed shifting to a statistical view

3.2.2.2 XML TransactionList.fxml

Class Name	TransactionList.fxml			
Description	A view containing items the user can interact with to manipulate transaction settings			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.java.controllers.TransactionAd	Reference to controller

3.2.2.3 Class TransactionView

Class Name	TransactionView			
Description	TransactionView is a set of buildable transactions created by the controller to be placed inside of the TransactionList.fxml View pane.			
Attributes	Visibility	Data Type	Name	Description
	private	Transaction	transaction	The transaction to be viewed
	private	Type	type	Type of the transaction
	private	EventHandler $\langle MouseEvent \rangle$	onAction	the variable to hold the mouseclick eventHandler
	private	ObjectProperty $\langle eventHandler \langle MouseEvent \rangle \rangle$	propertyOnAction	Adds additional methods to a given variable
Methods	Visibility	Name	Description	
	public	TransactionView(Transaction transaction, EventHandler $\langle MouseEvent \rangle$ onAction)	Default constructor for TransactionView	
	public	TransactionView(Type type)	Type based constructor of TransactionView	
	private	styleComponent(String style)	Adds stylesheet class the component	
	private	setContent()	Sets the datafields to be represented in the view	
	private	setContentType()	Set to view content with Types	
	public	setOnAction(EventHandler $\langle MouseEvent \rangle$ eventHandler)	Set an action to be performed on a mouseclick event	
	public	getOnAction()	Get the action that will be performed on a mouse click event	
	public	getTransaction()	Get the transaction being viewed	

3.2.2.4 TransactionView Methods:

Method Name	TransactionView(Transaction transaction, EventHandler ⟨*MouseEvent*⟩ onAction)
Class Name	TransactionView
Functionality	Creates a TransactionView object
Input	datafield arguments
Output	TransactionView object
Pseudo Code	BEGIN Set the class fields to the arguments and call styleComponent("transaction-view"); END

Method Name	TransactionView(Type type)
Class Name	TransactionView
Functionality	Creates a TransactionView object using type
Input	datafield arguments
Output	TransactionView object
Pseudo Code	BEGIN Set the class fields to the arguments and call styleComponent("transaction-view"); END

Method Name	styleComponent(String style)
Class Name	TransactionView
Functionality	Add a style class name to the component
Input	Stylesheet class
Output	-
Pseudo Code	this.getStyleClass().add(style);

Method Name	setContent()
Class Name	TransactionView
Functionality	Prepare the content inside of the view for display
Input	-
Output	-
Pseudo Code	BEGIN NumberFormat formatter =¿ Formate to currency Get child nodes of class Create text out of the values in this class; Add this text to child nodes END

Method Name	setContentType()
Class Name	TransactionView
Functionality	Adds the type field to contents that will be displayed in view
Input	-
Output	-
Pseudo Code	BEGIN Get the child nodes Add the type id to child nodes END

Method Name	setOnAction(EventHandler $\langle MouseEvent \rangle$ eventHandler)
Class Name	TransactionView
Functionality	Adds an action that will be performed on events
Input	An event handler to handle events
Output	-
Pseudo Code	BEGIN Set onAction field Set an action to be performed on mouseclick END

Method Name	getOnAction()
Class Name	TransactionView
Functionality	Returns the action to be performed during events
Input	-
Output	The event handler to handle events
Pseudo Code	BEGIN Return propertyOnAction.get() END

Method Name	getTransaction()
Class Name	TransactionView
Functionality	Returns the transaction being viewed
Input	-
Output	The transaction fo this class
Pseudo Code	BEGIN Return this.transaction END

3.2.2.5 XML TransactionAdd.fxml

Class Name	TransactionAdd.fxml			
Description	TransactionAdd.fxml contains the necessary items for users to add transactions to their account			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.java.controllers.TransactionAd	Reference to controller
	public	Button fx: id	"addTransactionButton"	Button to add new transactions
	public	Button text	"Clear"	Button to clear data

3.2.2.6 XML TransactionDetail.fxml

Class Name	TransactionDetail.fxml			
Description				
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.java.controllers.TransactionDe	Reference to controller
	public	Button fx: id	"editTransactionButton"	Button to edit transaction data
	public	Button text	"Cancel"	Button to leave menu with no adjustments

3.2.2.7 XML AccountView.fxml

Class Name	AccountView.fxml			
Description	AccountView is the main coordinator between the AccountList, AccountAdd and AccountDetail views. It is one central pane with controllers.			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.java.controllers.AccountContro	Reference to controller
	public	fx: id	"accountList" source "AccountList.fxml"	Reference to AccountList.fxml
	public	fx: id	"accountAdd" source "AccountAdd.fxml"	Reference to AccountAdd.fxml
	public	fx: id	"AccountDetail" source "AccountDetail.fxml"	Reference to AccountDetail.fxml

3.2.2.8 XML AccountList.fxml

Class Name	AccountList.fxml			
Description	AccountList displays a list of all the accounts the software has available for the user to interact with. It relies on an aggregate of SingleAccountView classes to properly display the information on all accounts.			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.jav	Reference to controlle
	public	Button fx: id	"NewAcc	Button to create a new account

3.2.2.9 Class SingleAccountView

Class Name	SingleAccountView			
Description	SingleAccountView is a set of buildable accounts created by the controller to be placed inside of the AccountList.fxml View pane.			
Attributes	Visibility	Data Type	Name	Description
	private	Account	Account	The account processed for display
	private	EventHandler $\langle MouseEvent \rangle$	onAction	the variable to hold the mouseclick eventHandler
	private	ObjectProperty $\langle eventHandler \langle MouseEvent \rangle \rangle$	propertyOnActior	Adds additional methods to a given variable
Methods	Visibility	Name	Description	
	public	SingleAccountView(Account account)	Constructor with account to use	
	public	setOnAction(EventHandler $\langle MouseEvent \rangle$ eventHandler):void()	Sets the handler that will proccess events	
	public	getOnAction()	Get the handler that will process events	
	public	getAccount()	Get the account being worked on	
	public	setAccount()	Set the account being worked on	
	private	styleComponent()	Add a style class to the component being worked on	

3.2.2.10 SingleAccountView Methods:

Method Name	SingleAccountView(Account account)
Class Name	SingleAccountView
Functionality	Creates an AccountView object with an account
Input	datafield arguments
Output	SingleAccountView object
Pseudo Code	BEGIN Set the class fields to the arguments and call styleComponent() The format each node with a currency formatter and insert text into nodes END

Method Name	setOnAction(EventHandler $\langle MouseEvent \rangle$ eventHandler)
Class Name	SingleAccountView
Functionality	Sets the event that will occur on events
Input	The event handler which will activate on events
Output	-

Pseudo Code	BEGIN
	this.setOnMouseClicked(eventHandler);
	END

Method Name	getOnAction()
Class Name	SingleAccountView
Functionality	Gets the event that will occur on events
Input	-
Output	The eventHandler

Pseudo Code	BEGIN
	return propertyOnAction.get();
	END

Method Name	getAccount()
Class Name	SingleAccountView
Functionality	Get the account being viewed
Input	-
Output	Account
Pseudo Code	BEGIN return account END

Method Name	styleComponent()
Class Name	SingleAccountView
Functionality	Adds a style class to current component
Input	-
Output	-

Pseudo Code	BEGIN
	this.getStyleClass().add("account-view");
	END

3.2.2.11 XML AccountAdd.fxml

Class Name	Transaction(String name, String type, double amount, String description, Date date)			
Description	AccountAdd is the view which allows the user to create new accounts. It is a single gridplane with fields for the user to fill out.			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.java.controllers. AccountAddController"	Reference to controller
	public	Button fx: id	"backToAccountListButton"	Button for the user to go back and erase their entries
	public	Button fx: id	"saveAccountButton"	Button for the user to confirm and submit their entries

3.2.2.12 XML AccountDetail.fxml

Class Name	AccountDetail.fxml			
Description	The AccountDetail view displays to the user all their information relating to a certain account. It is a gridplane with added data from the controller to show account current information from the database.			
Attributes	Visibility	Data Type	Name	Description
	public	fx: controller	"main.jav	Reference to controller ler"

3.3 Module ⟨*Controller*⟩:

3.3.1 Detailed Design Diagram

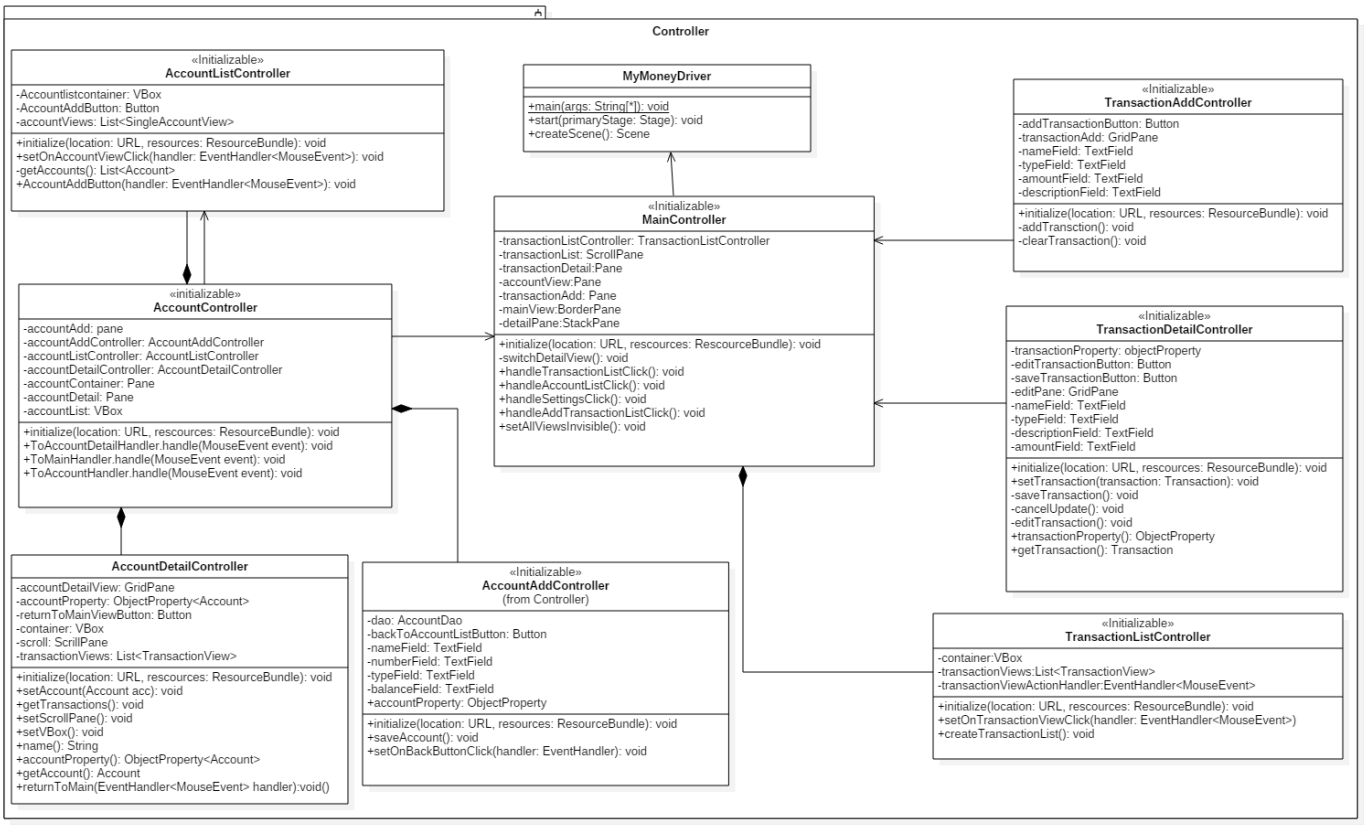


Figure 3: Controller Module Class Diagram

3.3.2 Units Description

Controller classes and methods use the model's 'Data Access Objects' to retrieve information from the database and store it in the respective model classes(Transaction, Type Account) to be placed into the dynamic elements of the view .fxml files. Controllers also set visibility and invisibility of views.

3.3.2.1 Class MyMoneyDriver

Class Name	MyMoneyDriver		
Description	The starter code for the application using Java's main function		
Methods	Visibility	Name	Description
	public static	main(args: String[*])	begins the program
	public	start(primaryStages:Stage)	Sets the window properties and drives the controllers to create application
	public	createScene()	Sets up the MainView.fxml class and CSS

3.3.2.2 MyMoneyDriver Methods:

Method Name	main(args: String[*])
Class Name	MyMoneyDriver
Functionality	Begins the application
Input	start fields
Output	-
Pseudo Code	BEGIN launch javafx environment END

Method Name	start(primaryStages:Stage)
Class Name	MyMoneyDriver
Functionality	Sets up the JavaFX window properties
Input	Window to be used
Output	-

Pseudo Code	BEGIN
	primaryStage.setTitle("MyMoney Application");
	primaryStage.setScene(createScene());
	primaryStage.setResizable(false);
	primaryStage.show();
	END

Method Name	createScene()
Class Name	MyMoneyDriver
Functionality	Creates the mainview window and stylesheet
Input	-
Output	-
Pseudo Code	BEGIN Load the mainview.fxml with FXMLLoader and add stylesheet. Set view size END

3.3.2.3 Class MainController

Class Name	MainController			
Description	The main controller controls the MainView fxml files and coordinates which display should be used from the main.			
Attributes	Visibility	Data Type	Name	Description
	private	TransactionListController	transactionListController	A reference to a transactionList-Controller
	private	ScrollPane	transactionList	Scrollplane that gets filled with Transaction-View objects
	private	Pane	transactionDetail	Holds information on transactions.
	private	Pane	accountView	A view object of accounts to be controlled
	private	Pane	transactionAdd	A view object of Transactions to be controlled. Deals with transaction creation.
	private	BorderPane	mainView	A view object of Main to be controlled
	private	StackPane	detailPane	A view object of Main to be controlled
Methods	Visibility	Name	Description	
	public	initialize(location:URL, rescources:RescourceBundle)	provides class with necessary controller info	
	private	switchDetailView()	Transfers the current seen object to detailview	
	public	handleTransactionListClick()	Use the transaction list controller to shift views	
	public	handleAccountListClick()	Use the account list controller to shift views	
	public	handleSettingsClick()	Shift views to settings	
	public	handleAddTransactionListClick()	Switch views add transaction list through it's controller	
	public	setAllViewsInvisible()	Set all panes to invisible with their controllers	

3.3.2.4 MainController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	MainController
Functionality	Sets all views invisible, sets account view visible and then sets up transactionListController to handle events
Input	Database URL and view resource
Output	-
Pseudo Code	BEGIN setAllViewsInvisible() accountView.setVisible(true) transactionListController.setOnTransactionViewClick(new TransactionViewClickHandler()) END
Method Name	switchDetailView()
Class Name	MainController
Functionality	Flips the views over from list to detail
Input	-
Output	-
Pseudo Code	BEGIN transactionList.setVisible(NOT transactionList.isVisible()) transactionDetail.setVisible(NOT transactionDetail.isVisible()) END
Method Name	handleTransactionListClick()
Class Name	MainController
Functionality	Switches view to transaction detail view and builds it's list
Input	-
Output	-
Pseudo Code	BEGIN transactionListController.createTransactionList() setAllViewsInvisible() transactionList.setVisible(true) END
Method Name	handleAccountListClick()
Class Name	MainController
Functionality	Switches view to account detail view
Input	-
Output	-

Pseudo Code	BEGIN
	setAllViewsInvisible();
	accountView.setVisible(true);
	END

Method Name	handleAddTransactionClick()
Class Name	MainController
Functionality	Switches view to add transaction detail view
Input	-
Output	-

Pseudo Code	BEGIN
	setAllViewsInvisible();
	transactionAdd.setVisible(true);
	END

Method Name	setAllViewsInvisible()
Class Name	MainController
Functionality	Switches all views to invisible
Input	-
Output	-

Pseudo Code	BEGIN
	transactionAdd.setVisible(false);
	accountView.setVisible(false);
	transactionList.setVisible(false);
	transactionDetail.setVisible(false);
	statisticsView.setVisible(false);
	END

3.3.2.5 Class TransactionAddController

Class Name	TransactionAddController			
Description	The TransactionAddController is the controller object for the TransactionAdd.fxml file for adding transactions. It's methods relate to the manipulation of view items on this scene.			
Attributes	Visibility	Data Type	Name	Description
	private	Button	addTransactionButton	Button is pressed to confirm the addition of a new transaction
	private	GridPane	transactionAdd	Form container for transaction add data
	private	TextField	nameField	New transaction field data
	private	TextField	typeField	New transaction field data
	private	TextField	amountfield	New transaction field data
	private	TextField	descriptionField	New transaction field data
Methods	Visibility	Name		Description
	public	initialize(location:URL, resources: ResourceBundle)		Initializes the controller and creates connection to database
	public	addTransction()		Communicate with the server to send field data to the server
	public	clearTransaction()		Set all the data fields to blank

3.3.2.6 TransactionAddController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	TransactionAddController
Functionality	Sets up a connection to the Dao and Database then fills out data fields with items
Input	Database URL and view resource
Output	-

Pseudo Code	BEGIN
	AccountDao dbAccount=new AccountDao()
	for (Account e: dbAccount.getAllAccounts())
	comboBox.getItems().add(e.getId()+". "+e.getName())
	END

Method Name	addTransaction()
Class Name	TransactionAddController
Functionality	Reads fields and sends the data to the Database for saving through the DAO after formating
Input	-
Output	-
Pseudo Code	BEGIN transDAO = new TransactionDao() typeDao = new TypeDao *GET DATA FIELDS t = new Transaction(*DATA FIELDS) transDAO.insert(t) clearTransaction() END

Method Name	clearTransaction()
Class Name	TransactionAddController
Functionality	Set all data fields to blank
Input	-
Output	-
Pseudo Code	BEGIN *SET ALL DATA FIELDS TO ”” END

3.3.2.7 Class TransactionDetailController

Class Name	TransactionDetailController			
Description	The TransactionDetailController is the controller object for the TransactionDetail.fxml file for viewing specifics about transactions. It's methods relate to the manipulation of view items on this scene.			
Attributes	Visibility	Data Type	Name	Description
	private	objectProperty $\langle Transaction \rangle$	transactionProperty	Extended Transaction class with ObjectProperty methods
	private	Button	editTransactionButton	Button to trigger the editing pane
	private	Button	saveTransactionButton	Button to trigger the saving pane
	private	TextField	editField	Form field for editing
	private	TextField	nameField	Form field for editing
	private	TextField	descriptionField	Form field for editing
	private	TextField	amountField	Form field for editing
Methods	Visibility	Name	Description	
	public	initialize(location:URL, resources:ResourceBundle)	Open connection to the database	
	public	setTransaction(transaction: Transaction)	Set transactionProperty to argument	
	private	saveTransaction()	send transaction to database from form fields	
	private	cancelTransaction()	Go back to the detailPane[TransactionDetailControler]	
	private	editTransaction()	Open edit pane and fill out forms with the transaction called to be edited	
	public	transactionProperty()	returns transactionProperty	
	public	getTransaction()	returns the transaction linked to transactionProperty	

3.3.2.8 TransactionDetailController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	TransactionDetailController
Functionality	Sets up a connection to the Dao and Database then fills out data fields with items
Input	Database URL and view resource
Output	-

Pseudo Code	BEGIN
	AccountDao dbAccount=new AccountDao()
	for (Account e: dbAccount.getAllAccounts())
	comboBox.getItems().add(e.getId()+". "+e.getName())
	END

Method Name	setTransaction(transaction: Transaction)
Class Name	TransactionDetailController
Functionality	Sets the transactionProperty to transaction and makes detailPane visible
Input	Transaction to set as the ObjectProperty
Output	-
Pseudo Code	BEGIN Create TransactionDao Create TypeDao Transaction t = transactionProperty.get() Get Data from fields and place it into the Transaction t. Insert into database if it's a new piece of data. Otherwise update records. If record of Transaction no longer has any fields (size ¡ 1) delete it from the records. Set current Pane invisible and make detail Pane visible END

Method Name	cancelUpdate()
Class Name	TransactionDetailController
Functionality	Make detailPane visible and editPane invisible
Input	-
Output	-
Pseudo Code	BEGIN editPane.setVisible(false) detailPane.setVisible(true) END

Method Name	editTransaction()
Class Name	TransactionDetailController
Functionality	Make detailPane invisible and editPane visible then set the TextField values to those of the Transaction object
Input	-
Output	-
Pseudo Code	BEGIN editPane.setVisible(true) detailPane.setVisible(false) get transaction from transactionProperty Set each textfield field to transaction values END

Method Name	transactionProperty()
Class Name	TransactionDetailController
Functionality	Return the transaction property
Input	-
Output	ObjectProperty¡Transaction¿

Method Name	getTransaction()
Class Name	TransactionDetailController
Functionality	Return the transaction from the transaction property
Input	-
Output	Transaction

3.3.2.9 Class TransactionListController

Class Name	TransactionListController
Description	The TransactionListController is the controller object for the TransactionList.fxml file. It's methods relate to the manipulation of view items on this scene as well as produce additonal view objects inside of the TransactionList.fxml scene such as the TransactionView.

Attributes

Visibility	Data Type	Name	Description
private	VBox	container	A container object for holding TransactionView objects
private	VBox	containerType	A container object for holding TransactionView objects sorted by type
private	List⟨*TransactionView*⟩	transactionViews	A List of transactionView Objects to be placed in VBox
private	List⟨*TransactionView*⟩	transactionViewsBy	A List of transactionView Objects by type to be placed in a VBox
private	EventHandler⟨*MouseEvent*⟩	transactionViewActi	A handler for click events on given view objects

Methods

Visibility	Name	Description
public	initialize(in location:URL, in resources:ResourceBundle)	Generate the two initial transaction view array lists
public	setOnTransactionViewClick(handler)	set the basic TransactionView click handler
public	createTransactionList()	clears previous settings, opens a connection to the database through the dao and creates all TransactionViews from database data.
public	getTransactions()	uses the TransactionDao to retrieve all transactions from the database in a list

3.3.2.10 TransactionListController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	TransactionListController
Functionality	Creates two new arraylists for containing TransactionViews
Input	Database URL and view resource
Output	-
Pseudo Code	BEGIN Generate 2 new TransactionView ArrayLists END
Method Name	setOnTransactionViewClick(EventHandler¡MouseEvent¿ handler)
Class Name	TransactionListController
Functionality	Sets the event handler for TransactionView click events
Input	The handler to be assigned
Output	-
Pseudo Code	BEGIN transactionViewActionHandler = handler END
Method Name	createTransactionList()
Class Name	TransactionListController
Functionality	Clears both of the Array lists and their respective VBoxes, opens connection to the Dao and creates a set of TransactionView objects from all Transactions and Types stored on the database. It then assigns all Transaction and Type View records into each container after assigning them an event handler.
Input	-
Output	-
Pseudo Code	BEGIN Clear both transactionView ArrayLists Clear both VBox contents Open connection to database Retrieve all transactions and types For all data in array lists assign them the click event handler Assign all data to their respective VBoxes END

Method Name	getTransactions()
Class Name	TransactionListController
Functionality	Creates a TransactionDao connection to the database and then retrieves all of the database's transactions
Input	-
Output	List $\langle Transaction \rangle$

Pseudo Code	BEGIN
	TransactionDao transactionDao = new TransactionDao();
	return transactionDao.getAllTransactions();
	END

3.3.2.11 Class AccountListController

Class Name	AccountListController			
Description	The AccountListController controls AccountList.fxml to fill a VBox container with a list of all accounts in the database and interactions with generated SingleAcountView objects it generates.			
Attributes	Visibility	Data Type	Name	Description
	private	Button	NewAccountButton	A button to be pressed to trigger new accounts being added to the database.
	private	VBox	accountListContaine	The location where SingleAccountView objects will be stored
	private	List⟨*SingleAccountView*⟩	accountViews	A List of all accounts to be placed into the VBox
	private	AccountController	accountController	Reference to the AccountController object this object is contained in
	private	EventHandler ⟨*MouseEvent*⟩	accountViewActionH	The handler that will activate on account view clicks
	private	Button	returnToMainViewB	Button to return user back to main view
Methods	Visibility	Name	Description	
	public	initialize(location:URL, resources:ResourceBundle)	Initializes the acountViews arraylist	
	public	setupAccounts(EventHandler handler)	Assigns the handler field to argument and array list variables, clears vboxes and adds SingleAccountViews to them	
	private	getAccounts()	creates an AccountDao then retrieves and returns all Accounts in the database	
	public	AccountAddClick(handler:Ev	Assigns an event handler to the NewAccountButton	

3.3.2.12 AccountListController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	AccountListController
Functionality	Creates a new arraylist for containing AccountViews
Input	Database URL and view resource
Output	-
Pseudo Code	BEGIN Generate a new SingleAccountView ArrayList END

Method Name	setupAccounts(EventHandler$\langle MouseEvent \rangle$)
Class Name	AccountListController
Functionality	Sets the SingleAccountView click handler, clears the ArrayList for each accountViews and the VBox, then retrieves all accounts, creates SingleAccountView items, places them in the container and adds event handlers to each of them.
Input	EventHandler to be triggered on any SingleAccountView click
Output	-
Pseudo Code	BEGIN Assign handler Clear ArrayList Clear VBox contents Get all accounts from database Create them and assign them their handler Add them to the VBox END

Method Name	AccountAddClick(handler:EventHandlerList$\langle MouseEvent \rangle$)
Class Name	AccountListController
Functionality	Sets the event handler for the add account button
Input	The handler to be assigned
Output	-
Pseudo Code	BEGIN NewAccountButton.setOnMouseClicked(handler); END

Method Name	getAccounts()
Class Name	AccountListController
Functionality	Creates an AccountDao connection to the database and then retrieves all of the database's accounts
Input	-
Output	List $\langle Account \rangle$

Pseudo Code	BEGIN
	AccountDao AccountDao = new AccountDao();
	return accountDao.getAllAccounts();
	END

3.3.2.13 Class AccountController

Class Name	AccountController			
Description	The AccountController class controls AccountView.fxml in order to display the view the user wants to see. It contains the controllers for AccountAddController, AccountListController and AccountDetailController in order to respond to user requests to view certain Views			
Attributes	Visibility	Data Type	Name	Description
	private	Pane	accountAdd	The pane to hold AccountAdd views
	private	AccountAddController	accountAddController	The controller of AccountAdd views
	private	AccountListController	accountListController	The controller of AccountList views
	private	AccountDetailController	accountDetailController	The controller of AccountDetail views
	private	Pane	accountContainer	The pane to hold AccountList views
	private	Pane	accountDetail	The pane to hold AccountDetail views
	private	VBox	accountList	The container to store a list of accounts
Methods	Visibility	Name	Description	
	public	initialize(location: URL, resources: ResourceBundle)	Initialize by setting accountList pane to visible and attaching eventHandlers to each controllers navigation buttons	
	public	ToAccountDetailHandler.handle (MouseEvent event)	Switches views to the AccountDetail.fxml view	
	public	ToMainHandler.handle (MouseEvent event)	Switches views to the MainView.fxml view	
	public	ToAccountHandler.handle (MouseEvent event)	Switches to the AccountAdd.fxml View	

3.3.2.14 AccountController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	AccountController
Functionality	Sets all views to invisible except the AccountList view the adds event handlers to each controller's navigation buttons
Input	Database URL and view resource
Output	-
Pseudo Code	BEGIN 1. Set visibility all to false, except account list 2. Assign event handlers to each controller's navigation buttons using ToAccountDetailHandler.handle(MouseEvent event), ToMainHandler.handle(MouseEvent event) and ToAccountHandler.handle(MouseEvent event) END
Method Name	ToAccountDetailHandler.handle(MouseEvent event)
Class Name	AccountController
Functionality	Sets up the Accounts contained in the AccountDetailController then sets the AccountDetail view to be visible and all others to be invisible
Input	The event being fired on the handler
Output	-
Pseudo Code	BEGIN Create SingleAccountView from event source Gets Account from the SingleAccountView Sets the accountDetailController's account to the Account Set all panes invisible except AccountDetail END
Method Name	ToMainHandler.handle(MouseEvent event)
Class Name	AccountController
Functionality	Sets all panes to be invisible except from AccountList
Input	The event being fired on the handler
Output	-
Pseudo Code	BEGIN Set all panes invisible except AccountList END

Method Name	ToAccountAddHandler.handle(MouseEvent event)
Class Name	AccountController
Functionality	Sets all panes to be invisible except from AccountAdd
Input	The event being fired on the handler
Output	-
Pseudo Code	BEGIN Set all panes invisible except AccountAdd END

3.3.2.15 Class AccountAddController

Class Name	AccountAddController			
Description	AccountAddController communicates with the database to store accounts into it's database through user manipulation of view objects.			
Attributes	Visibility	Data Type	Name	Description
	private	AccountDao	dao	Connection to the database for accounts
	private	Button	backToAccountList-Button	Button to redirect user to account list
	private	TextField	nameField	Field for account creation.
	private	TextField	numberField	Field for account creation.
	private	TextField	typeField	Field for account creation.
	private	TextField	balanceField	Field for account creation.
	public	ObjectProperty	accountProperty	Adds additional methods to a given variable
Methods	Visibility	Name	Description	
	public	initialize(location:URL, resources:ResourceBundle)	Sets a connection to the database	
	public	saveAccount()	Retrieve data in TextFields and send result to database	
	public	setOnBackButtonClick(hand	Attatch listener to button to go back	

3.3.2.16 AccountAddController Methods:

Method Name	initialize(location:URL, rescources:RescourceBundle)
Class Name	AccountAddController
Functionality	Sets up a connection to the Dao and Database
Input	Database URL and view resource
Output	-
Pseudo Code	BEGIN AccountDao dbAccount=new AccountDao() END

Method Name	saveAccount()
Class Name	AccountAddController
Functionality	Reads fields and saves to database
Input	Database URL and view resource
Output	-

Pseudo Code	BEGIN
	AccountDao dbAccount=new AccountDao()
	a = Get Account
	if (a == null) {
	a = new Account();
	a.setCreate(new Date());
	a.setEdit(null);
	a.setArchived(null);
	}
	
	*GET VALUES FROM FIELDS
	
	a.setName(name);
	a.setBalance(balance);
	a.setNumber(number);
	a.setType(type);
	a.setTypeName(typeName);
	try {
	if (accountProperty.get() == null) {
	dao.insert(a);
	} else {
	dao.updateAccount(a);
	}
	} catch (Exception ex) {
	}
	END

Method Name	setOnBackButtonClick(EventHandler⟨$MouseEvent$⟩ handler)
Class Name	AccountAddController
Functionality	Adds an event listener to the back button for clicks
Input	An event handler
Output	-
Pseudo Code	BEGIN backToAccountListButton.setOnMouseClicked(handler) END

4 Dynamic Design Scenarios

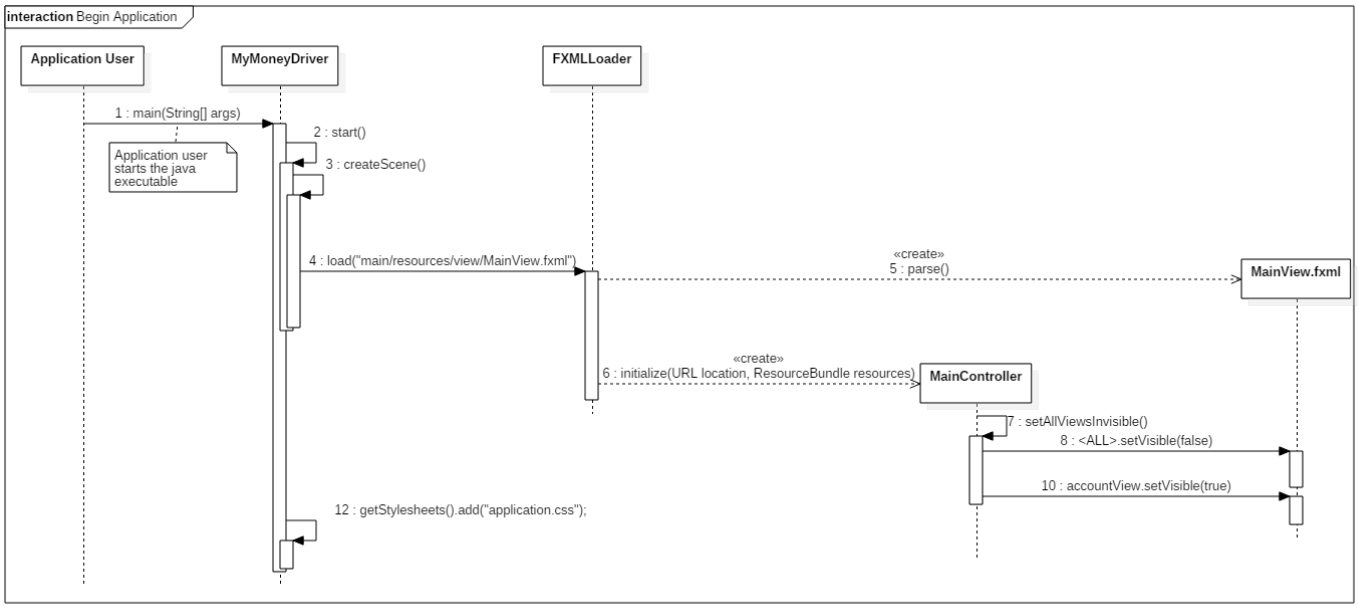


Figure 4: Sequence Diagram of Main View Generation

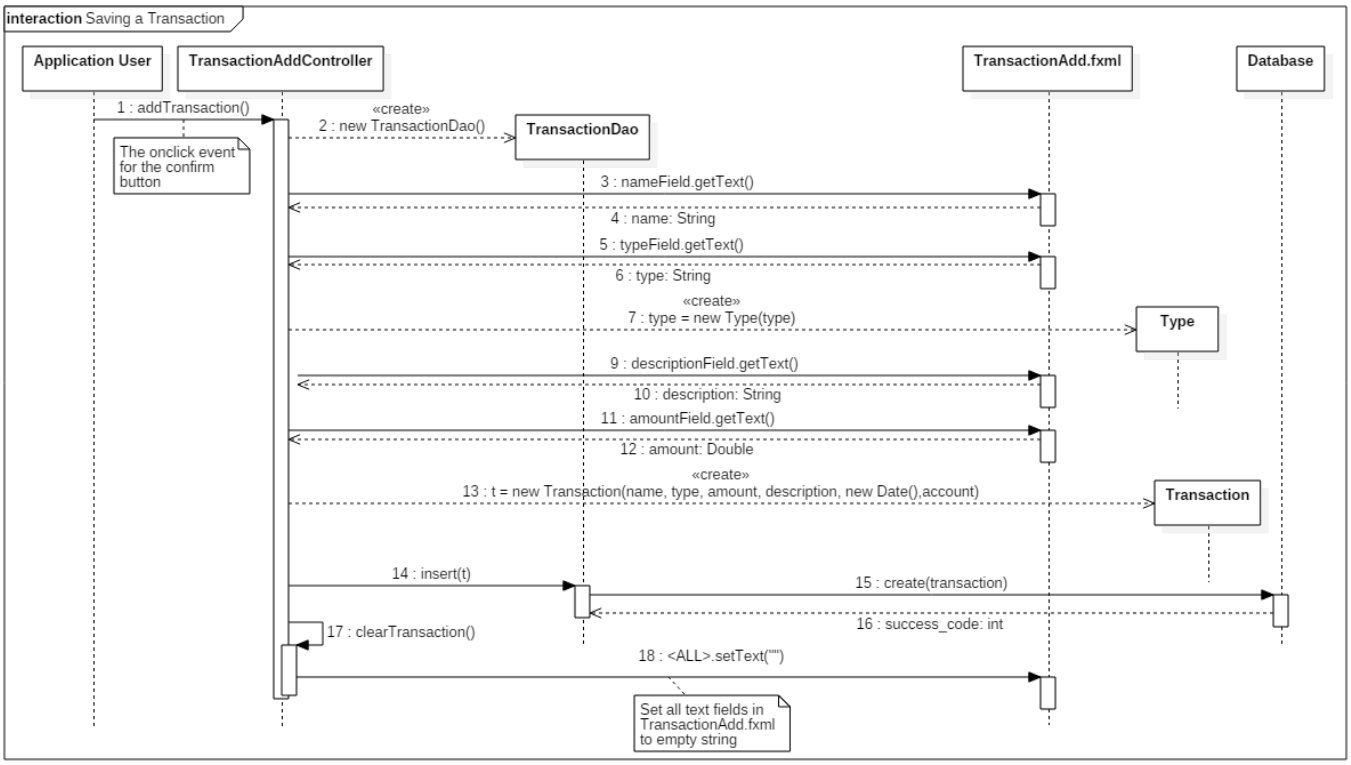


Figure 5: Sequence Diagram of Transaction Saving

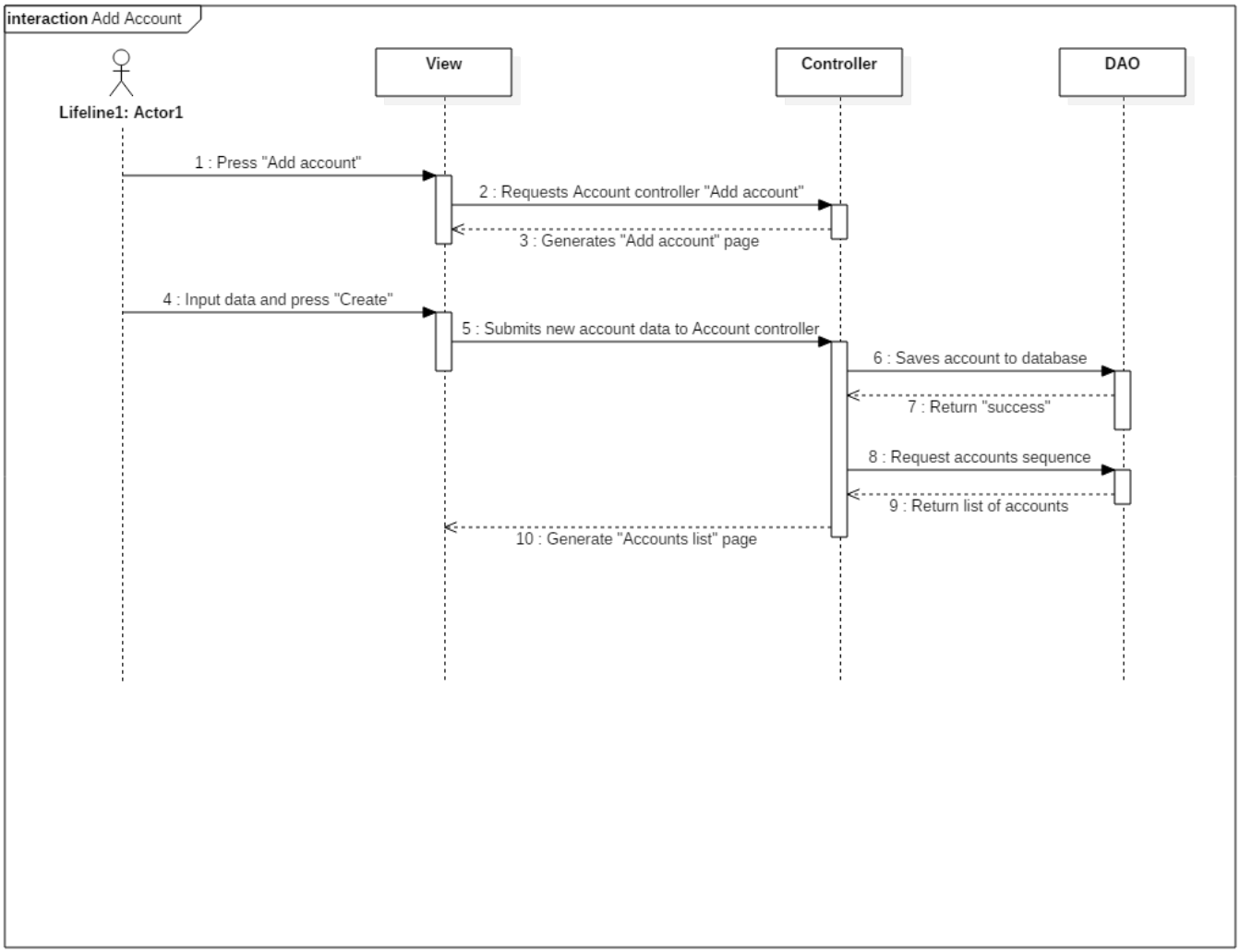


Figure 6: Sequence Diagram of Account Adding

5 Justification of Use-Cases and Rational

5.1 Drop Down Menus

Drop down menus are capable of conserving space in the application. By using drop downs a user can select which account he wants and which he doesn't. These dropdowns though simple in function are very practical in use.

5.2 Statistics

Statistical information is a must have on all financial applications and some users may need additional tools to conceptualize their financial situation. While the data may not

be complex it is a must have to include at least a limited set of statistical functionality.

5.3 Styling (CSS)

User experience is enhanced by good design. Interactions must be intuitive to the user via color, patterns, fonts and effects all drawing attention to what the user should be clicking on. Features of software should be easily discoverable and enhance the feedback given since this increases the enjoyability of using a product. Customizability is not a concern and giving the user custom control over the design does not really matter as the one design they are given is intuitive.

5.4 Deleting Transactions

There must be a way to communicate with the database to remove contents that are no longer relevant. In terms of giving the user control over his or her own experience we must trust that the user knows when it's best to remove traces of his financial history from the logbooks. While there are certain dangers inherent with letting the application have the power to delete data, it's assumed that a person using financial software has the good judgment not to take excessive risks.

5.5 View Account Transactions

A user will need to see his or her account transactions. Since data needs to be permanent it is necessary to put it on a database. This feature allows for users to navigate the view and click on systems to view their transaction history. Since the system already connects to the database to make insertion and delete requests, it is only natural that the user be given access to see his entire transaction table. It's assumed that the user wants to see all of the data so that freedom is given to him.

5.6 Adding a Transaction

Entering in information for transaction is essential to building a database and is given four fields of data to add to a transaction. Any more data than that is deemed unnecessary for the scope of the project. Name, type, amount and description make up all the data a user can submit for a transaction since it's assumed that things like, interest rate, credit limits or anything more complicated is best left for the user to calculate himself. Also it is inherent that every transaction is marked by a unique ID number that is not set by the user himself.

5.7 Editing an Existing Transaction

Users will inevitably make mistakes, when they do they will need to edit their data. It is assumed that a user can make errors on any field except for the ID therefore they are given the freedom to modify any of this data at any time from a list of all transactions

they have made. Assuming they remember the name of their transaction, they can find it in a list and edit it.

5.8 Adding a Bank Account

Setting up an account is the first thing a user must do when running this software and the most fundamental. By this logic an account must be set up to use the product. Accounts are given four identification fields. Your name, your bank account number, the type of account used and the balance. Upon saving the data a table is produced in the database holding your account info. The assumption is that the user can enter any kind of bank account type he wants and edit it as he sees fit. There are more possibilities of what the user can do on the application if everything is left as a simple template he can fill out and not limit the choices that can be done by giving out radio boxes or select bars.

5.9 Set Saving Goals

Since saving goals are a main reason why an individual might use a financial planning application this feature is included in the product. It is assumed that these individuals will want this goal to be fullfilled on a certain calender date. The application has textfields to fill out specifying the account balance they want to achieve by a certain date in time. Since the user is in need of keeping track of his spending against the goal, this saving's goal is presented on the account page to allow the user to constantly be reminded by it.

5.10 Generate Monthly Report

Users will want to know how they are doing every month. A simple report displaying information on their spending, saving and frequency of use is the end goal of this product and is represented in the monthly reports. While it aims to offer a comprehensive guide of what the user has done with their money, it does not allow for multiple account views. This is due to privacy issues of people sharing computers. Finally it is important for a user to check how he is doing against his set saving goals. While no bar graphs, pie charts and histograms are generated, hopefully the user can be satisfied knowing he has met his goals and that he should continue using the software.