COMP 354
Design Document for myMoney

Team PA-PK

April 9, 2018

Table 1: Team

Name	ID Number
Anne-Laure Ehresmann	27858906
Marc-Antoine Dube	40029307
Kadeem Caines	26343600
Abdel Rahman Jawhar	27192142
Keith Dion	40036340
Hrachya Hakobyan	40041555
Andrew-Smith	40034936
Dongyu Chen	27241909
Yauheni Karaniuk	40005680
Renny Xu	40005262
Wei Wang	40041116

Contents

List of Figures

List of Tables

1 Introduction and Purpose

The goal of this document is to define the design for the desktop application myMoney. The majority of the design decisions have been taken with the Requirements document in mind, one may thus want to look at this document first to have a clear picture of the problem in mind as well as the requirements demanded for the solution. This document presents an implementation of a possible solution to answer this problem. Its design is outlined through an Architectural Design (AD), a Detailed design (DD) and Dynamic Design Scenarios (DDS) for the application. The AD focuses on high-level project decomposition, the DD describes the overarching system design (which includes the UML design, divided into multiple subsections), and the DDS displays how the subsystems interact with one another in order to produce system-level services. This document may thus be used to plan, coordinate, and guide the development of the software, estimate and allocate necessary resources for proper execution, and then actually implement the software for the system. It seeks, above all, to serve as a precise and stable reference throughout development.

Check section 4.3 for a glossary of terms and abbreviation.

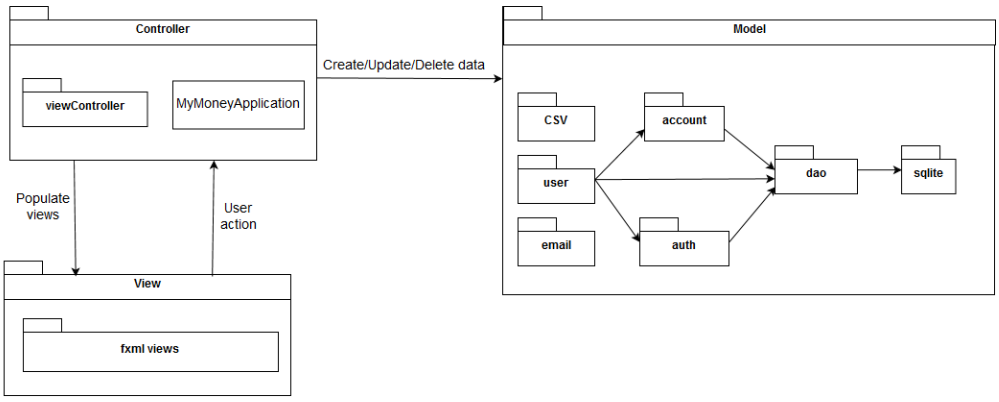
2 Scope

This document contains everything to do with the development decisions and design of the system, all of which are derived from the requirements, which are not described in this document. Also not included in here is any testing of the system, which verifies that the requirements are met. It is merely a blueprint for a system that should, in theory, successfully pass any tests that would be done in correspondence with the requirements.

3 Architectural Design

The myMoney application uses the MVC architectural pattern. MVC imposes a clear separation of concerns, and thus emphasises a high degree cohesion and low coupling. It is traditionally used for user interface applications, offers easy adaptability and maintainability, and a natural rigidity in module structure. All of these advantages made MVC a fitting choice for our architecture.

3.1 Architectural Diagram



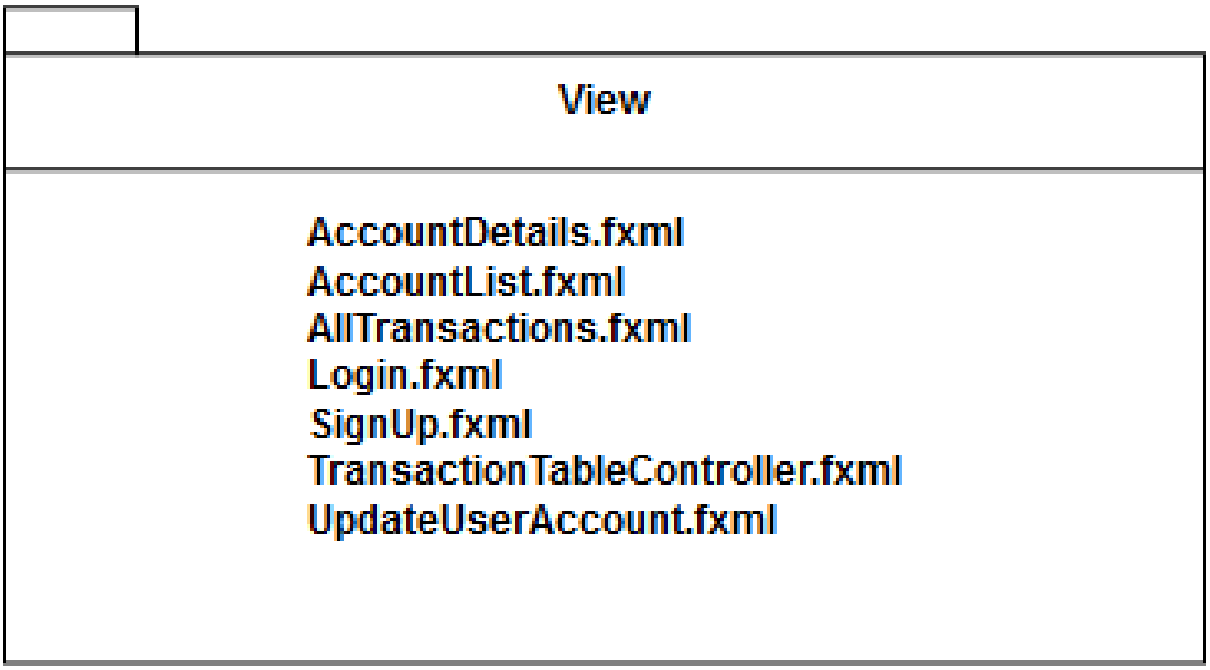
The *view*, our graphical user interface, is implemented through the JAVAFX front-end framework. It consists of table views, menus, labels and other GUI components, which are then populated by the view controllers with data from the model. The view is the only component that the user interacts with. It reports any user-triggered events (mouse clicks, text entries...) to the view controllers, and renders the updated model received from the controller. A more in-depth view of the interaction between the user and the view can be seen in the dynamic models, in section 5.

The *controllers*. The controller populates the views with the model and translates user input into service calls to manipulate the model. When the model is changed, it repopulates the views with new data.

The *model* is consists of two parts: the application logic, and the data. The application logic is organised in a layered structure of services, which each manage a different section of the system (session management, accounts, users, etc). Each service performs validation related to its system domain on its inputs, and then delegates the calls to the services of the layer below. The Database Access Objects (DAO) lie at the bottom of the hierarchy and directly communicate with the database. For examples in the controllers, their services, their intercommunication, and their validation, see section 3.2. The data is stored in an SQLite database. Our system actually employs two databases; The first, a local database, and the second, a "remote" database (also local, but acts as if it were remote) used to simulate the bank servers. The data itself consists of model classes, (e.g. bank account, account transaction, user account), which corresponds to the domain model of our application. See section 3.2 for more details on these services and the databases.

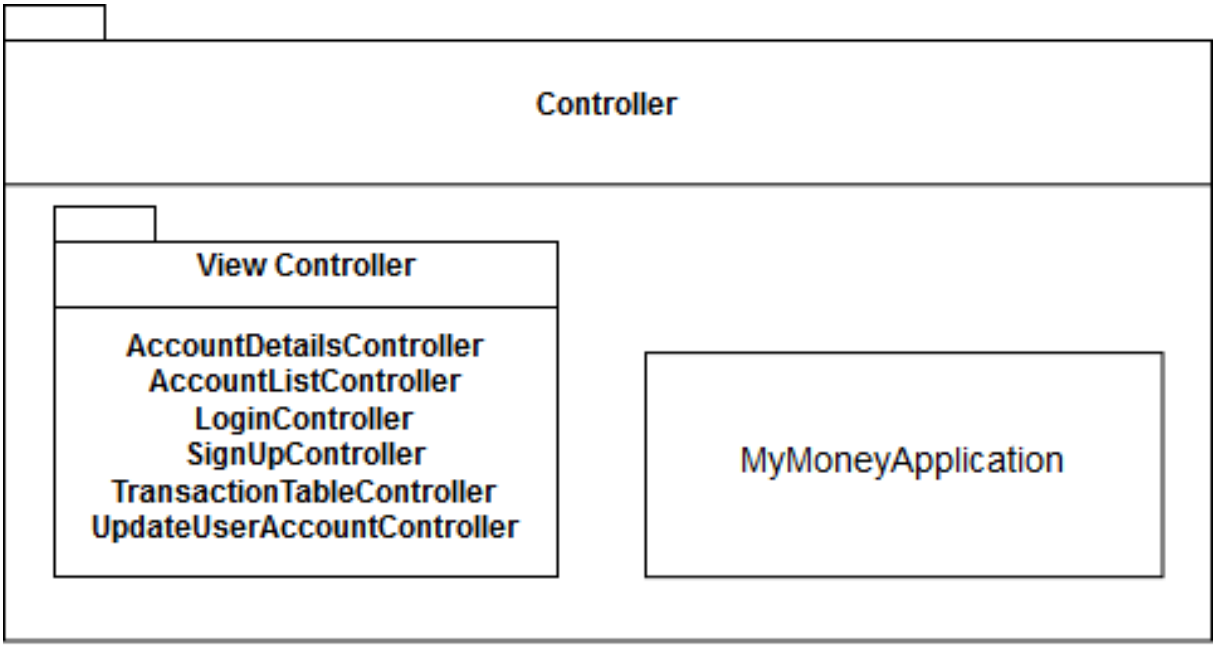
3.2 Subsystem Interface Specifications

View Subsystem



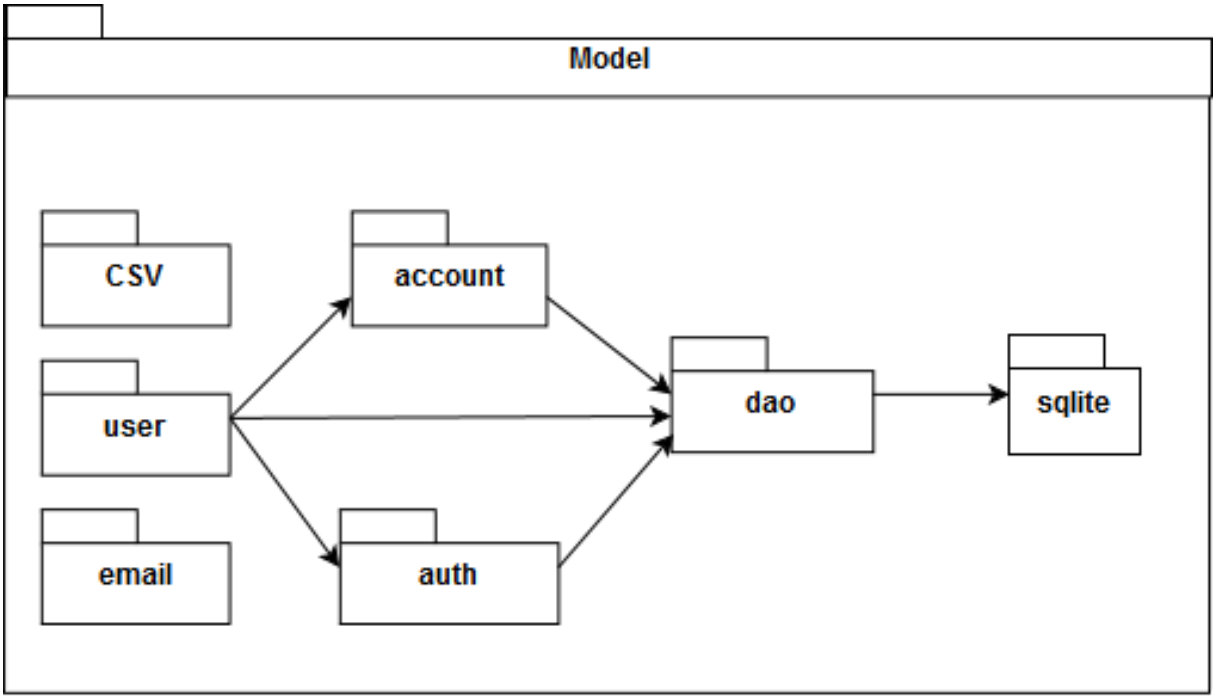
Our resources/fxml folder contains the view subsystem. The view is implemented, as mentioned before, through the JavaFX framework. It is a collection of FXML files, containing a structure contained in an AnchorPane. If the view model is simple menu (Login.fxml, SignUp.fxml, UpdateUserAccount.fxml), it will merely contain a few labeled text fields and buttons with 'onAction' parameters tying the buttons to a function in the view controllers. If it is a table view, (AccountDetails.fxml, AccountList.fxml, AllTransactions.fxml), then a TableView will be used with labelled columns, which are then recognised by the controllers in order to populate them with the model data. As nothing in our view contains actual code, no functions are mentioned here.

Controller Subsystem



The com.github.comp354project.viewController package contains a number of controllers for each different view. Each view is handled by its own controller, which contains each column and/or label as a private data member, and each button 'onAction' as a function. All controllers implement the Initializable interface, and hence contain an initialize(URL location, ResourceBundle resources) function which is called when the view is first displayed to the user. This function serves to populate the views with the domain data, if needed. The controllers may receive user input from the view and catch user events such as button presses or table entry selections. The functions catching these inputs pass these calls to the services in the model, described below. They are also the ones who pass any errors from the services to the views, mostly to be used for testing and alerting the user of any problems that might have occured. A more precise description of each controller is available in 4.2

Model Subsystem



The com.github.comp354project.model package contains both our application logic and our database. As mentioned in the previous section, it is organised in a layered manner, wherein each layer handles its own services and use the services of the layer below it within worrying about that layer's implementation. Data validation and processing is offered by each service: The account service, for example, validate calls to add or delete bank accounts, edit a transaction's category, or query for specific accounts. It does this by querying the database using an account DAO, and ensuring data integrity and validity (with regards to the business rules). It does not, however, worry about user authorisation, and simply assumes the layer above it (The user service) will have handled it. The com.github.comp354project.viewController calls services within this package to update the view and the model.

- The user module is a facade to the whole subsystem, and is called upon by the MVC controllers to pass the calls to the lower level layers.
- The account module handles account validation and verification, then passes the modification calls to the DAOs. Through the user package, it calls upon validation of authentication and authorisation of account modification calls.

- The auth module is the model which handles user authentication and session management, and is called upon by the user facade when other layers require a validation of user authentication. It also uses the DAOs to fetch the users from the database to validate the password and username during a login.
- The DAO module uses the SQLite module to create and return library DAOs to be used by the upper layers.
- The SQLite module provides connections to the underlying database.

We note that the com.github.comp354project.service package.account.remote package is a module to our model which mocks an API call to remote servers of banks or credit card companies. In our case however, we don't actually have access to such systems. For this reason, the remote data exists in an SQLite database like our local one. Hence, we don't truly consider it part of the architecture, instead it is merely an artifact of the implementation of this system.

4 Detailed Design

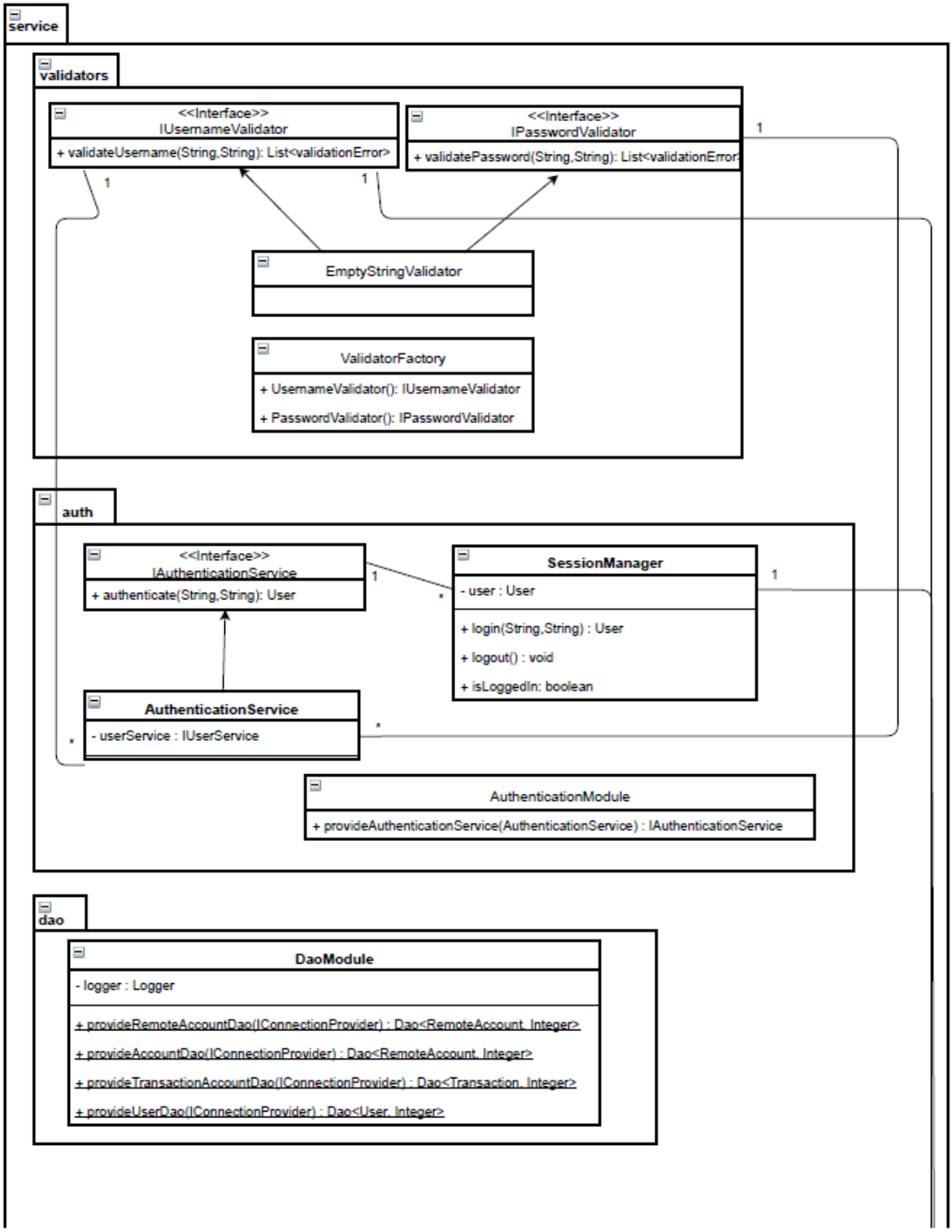
The myMoney system architecture is designed to be easily modified because of the low coupling between the modules. This was done with interfaces and auto injection of dependencies in classes. Each service package has a Module class designed to bind and provide an implementation to an interface. This way, classes are never instantiated directly into each other, but injected. This design pattern is useful because a change in implementation is as simple as creating a new class and change the module binding. The classes that use it and the tests should in no way be changed. Mocking classes for test purposes is also much easier.

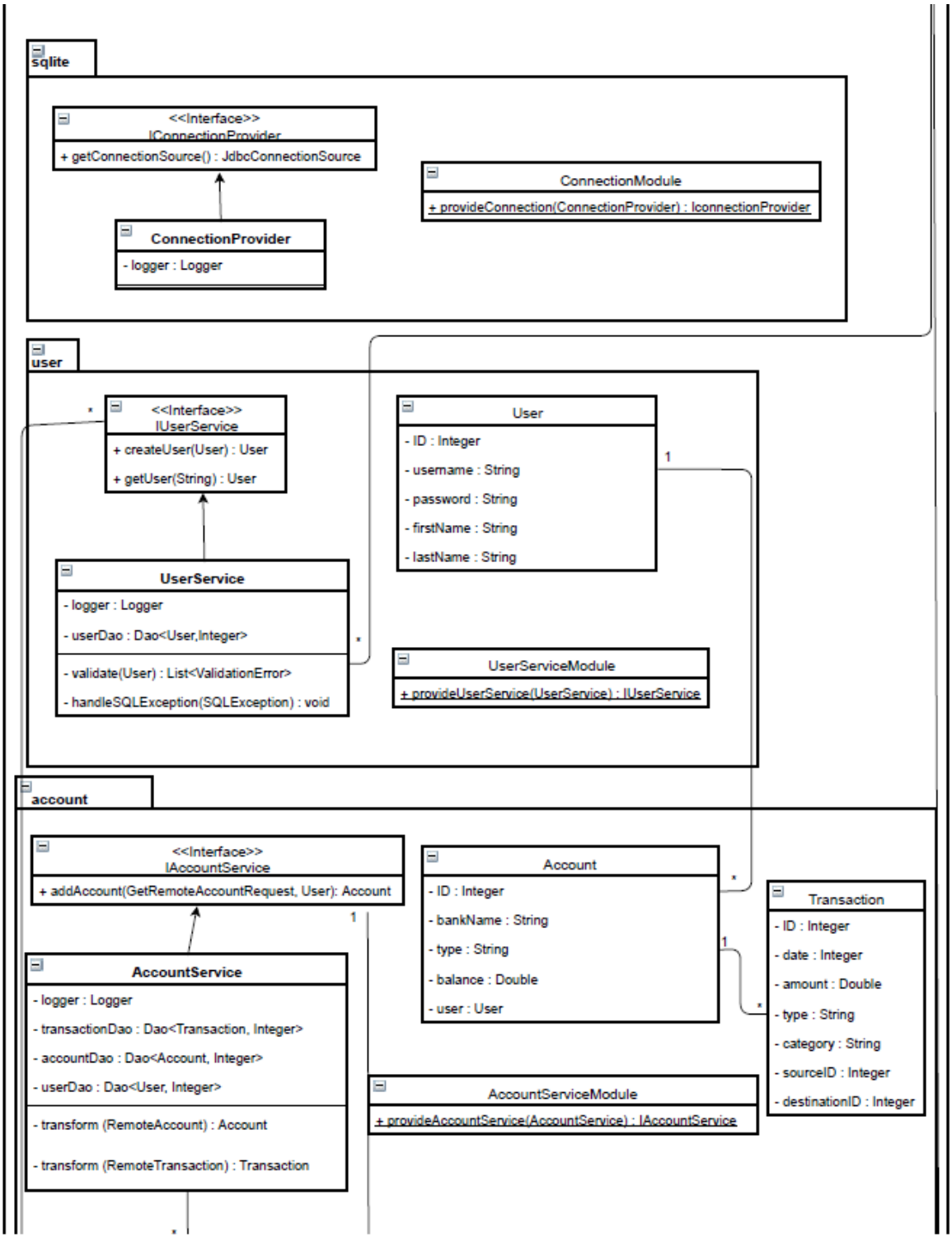
As a side note, we noticed that merge conflicts using git were much less likely to happen because we can each work on different parts of the system without modifying another module.

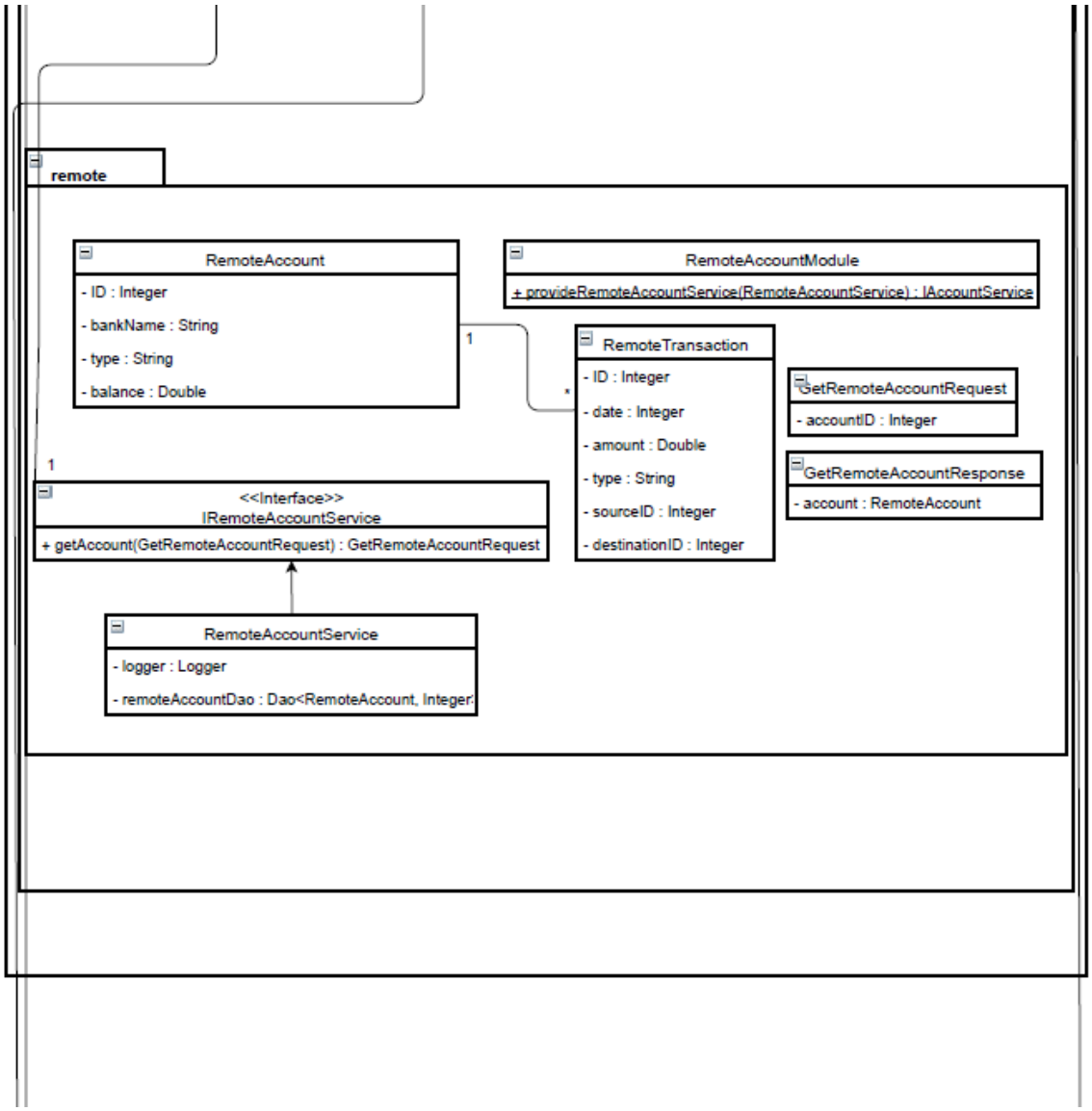
The tool used for this purpose is Dagger version 2.

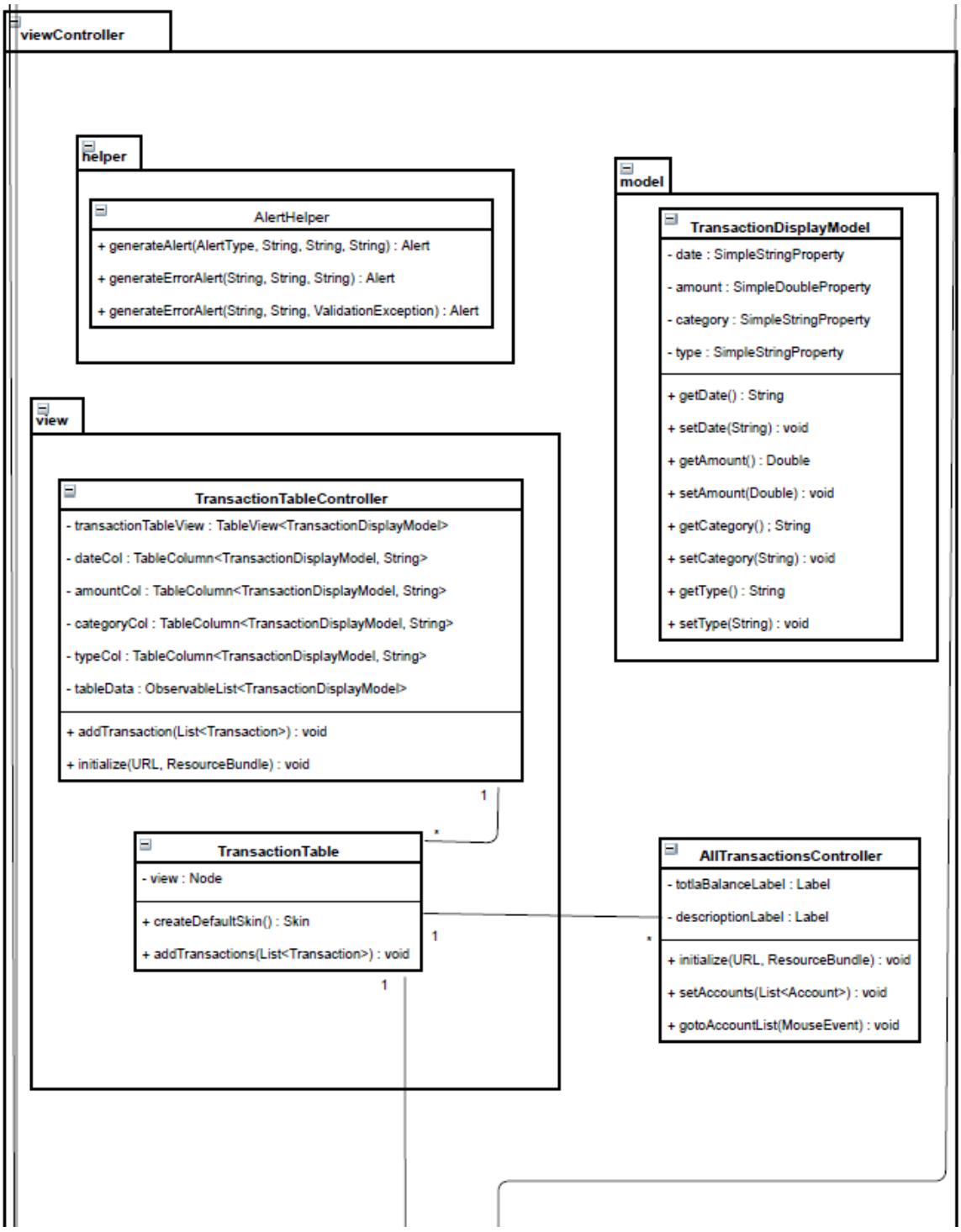
4.1 Class Diagram

In this section we provide the class diagram of our system, useful for the system developers and testers.This is an in depth look at all of the classes within our system see figure 1 below If a term is unclear, view section 4.3 for the glossary.


service
validators
<<Interface>>
IUsernameValidator
+ validateUsername(String,String): List<validationError>
<<Interface>>
IPasswordValidator
+ validatePassword(String,String): List<validationError>
1
1
1
EmptyStringValidator
ValidatorFactory
+ UsernameValidator(): IUsernameValidator
+ PasswordValidator(): IPasswordValidator
auth
<<Interface>>
IAuthenticationService
+ authenticate(String,String): User
1
*
SessionManager
- user : User
+ login(String,String) : User
+ logout() : void
+ isLoggedIn: boolean
1
AuthenticationService
- userService : IUserService
*
*
AuthenticationModule
+ provideAuthenticationService(AuthenticationService) : IAuthenticationService
dao
DaoModule
- logger : Logger
+ provideRemoteAccountDao(IConnectionProvider) : Dao<RemoteAccount, Integer>
+ provideAccountDao(IConnectionProvider) : Dao<RemoteAccount, Integer>
+ provideTransactionAccountDao(IConnectionProvider) : Dao<Transaction, Integer>
+ provideUserDao(IConnectionProvider) : Dao<User, Integer>


sqlite
<<Interface>>
IConnectionProvider
+ getConnectionSource() : JdbcConnectionSource
ConnectionProvider
- logger : Logger
ConnectionModule
+ provideConnection(ConnectionProvider) : IconnectionProvider
user
<<Interface>>
IUserService
+ createUser(User) : User
+ getUser(String) : User
UserService
- logger : Logger
- userDao : Dao<User,Integer>
- validate(User) : List<ValidationError>
- handleSQLException(SQLException) : void
User
- ID : Integer
- username : String
- password : String
- firstName : String
- lastName : String
UserServiceModule
+ provideUserService(UserService) : IUserService
account
<<Interface>>
IAccountService
+ addAccount(GetRemoteAccountRequest, User): Account
AccountService
- logger : Logger
- transactionDao : Dao<Transaction, Integer>
- accountDao : Dao<Account, Integer>
- userDao : Dao<User, Integer>
- transform (RemoteAccount) : Account
- transform (RemoteTransaction) : Transaction
Account
- ID : Integer
- bankName : String
- type : String
- balance : Double
- user : User
Transaction
- ID : Integer
- date : Integer
- amount : Double
- type : String
- category : String
- sourceID : Integer
- destinationID : Integer
AccountServiceModule
+ provideAccountService(AccountService) : IAccountService


remote
RemoteAccount
- ID : Integer
- bankName : String
- type : String
- balance : Double
RemoteAccountModule
+ provideRemoteAccountService(RemoteAccountService) : IAccountService
1
*
RemoteTransaction
- ID : Integer
- date : Integer
- amount : Double
- type : String
- sourceID : Integer
- destinationID : Integer
GetRemoteAccountRequest
- accountID : Integer
GetRemoteAccountResponse
- account : RemoteAccount
1
<<Interface>>
IRemoteAccountService
+ getAccount(GetRemoteAccountRequest) : GetRemoteAccountRequest
RemoteAccountService
- logger : Logger
- remoteAccountDao : Dao<RemoteAccount, Integer>


viewController
helper
AlertHelper
+ generateAlert(AlertType, String, String, String) : Alert
+ generateErrorAlert(String, String, String) : Alert
+ generateErrorAlert(String, String, ValidationException) : Alert
model
TransactionDisplayModel
- date : SimpleStringProperty
- amount : SimpleDoubleProperty
- category : SimpleStringProperty
- type : SimpleStringProperty
+ getDate() : String
+ setDate(String) : void
+ getAmount() : Double
+ setAmount(Double) : void
+ getCategory() ; String
+ setCategory(String) : void
+ getType() : String
+ setType(String) : void
view
TransactionTableController
- transactionTableView : TableView<TransactionDisplayModel>
- dateCol : TableColumn<TransactionDisplayModel, String>
- amountCol : TableColumn<TransactionDisplayModel, String>
- categoryCol : TableColumn<TransactionDisplayModel, String>
- typeCol : TableColumn<TransactionDisplayModel, String>
- tableData : ObservableList<TransactionDisplayModel>
+ addTransaction(List<Transaction>) : void
+ initialize(URL, ResourceBundle) : void
1
*
TransactionTable
- view : Node
+ createDefaultSkin() : Skin
+ addTransactions(List<Transaction>) : void
1
1
AllTransactionsController
- totlaBalanceLabel : Label
- descrioptionLabel : Label
+ initialize(URL, ResourceBundle) : void
+ setAccounts(List<Account>) : void
+ gotoAccountList(MouseEvent) : void
*

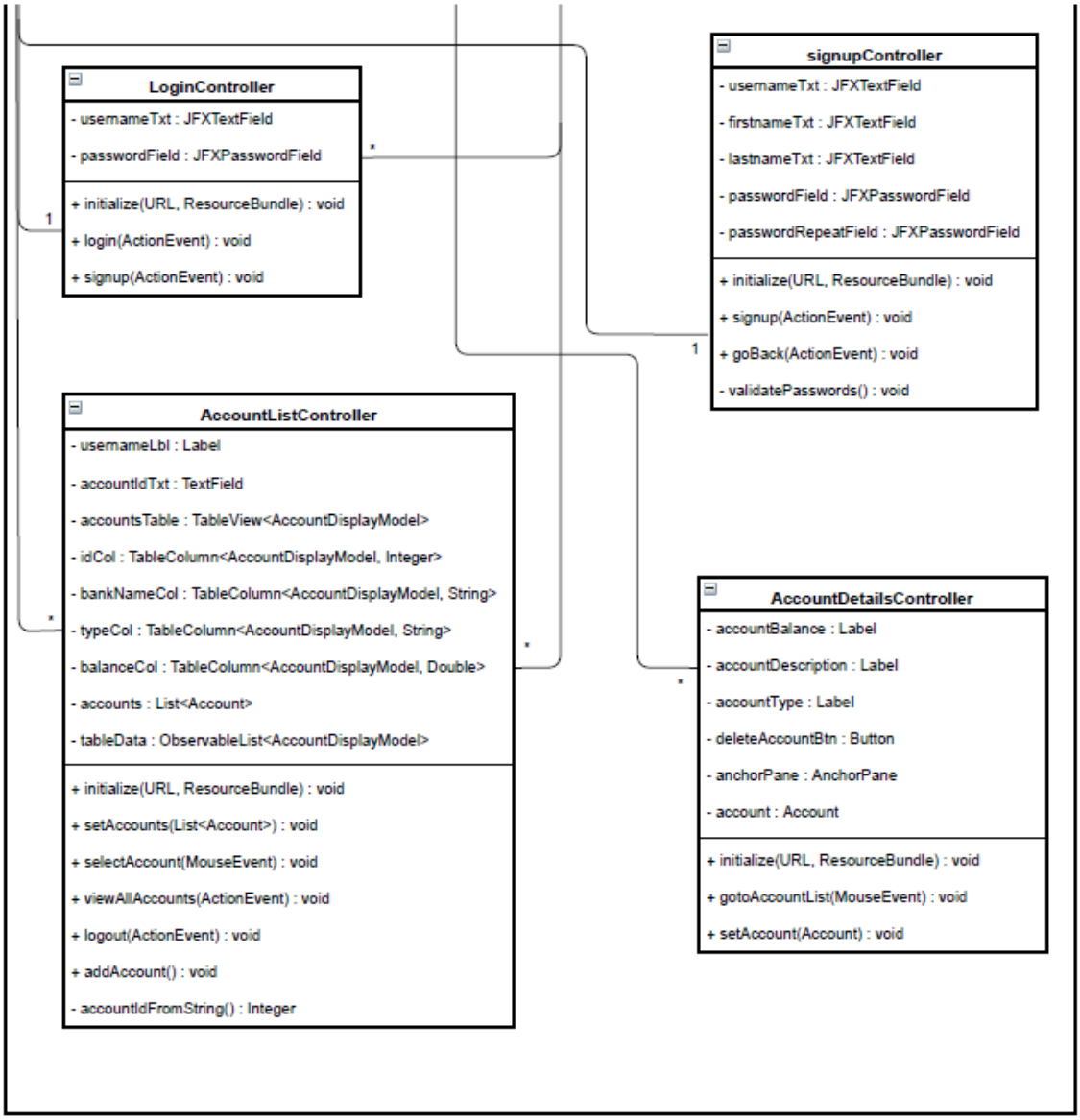


Figure 1: Class Diagram

4.2 Classes

Table 2: Interface ApplicationComponent

Class Name	com.github.comp354project.ApplicationComponent			
Visibility	Public			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Description	Class dependencies can be injected into the classes defined in the inject methods. This class is used for the Dagger2 injection framework			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	inject(MyMoneyApplication myMoneyApplication)	void	Injector for MyMoneyApplication class
	public	inject(LoginController loginController)	void	Injector for the LoginController class
	public	inject(AccountListController accountListController)	void	Injector for the AccountListController class
	public	inject(SignUpController signUpController)	void	Injector for the SignUpController class
	public	inject(TransactionTableController tableController)	void	Injector for the TransactionTableController class
	public	inject(UpdateUserAccountController updateUserAccountController)	void	Injector for the UpdateUserAccountController class

Table 3: Class BusinessRulesConstants

Class Name	com.github.comp354project.BusinessRulesConstants			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Contains business rules configuration for validators			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	public	Integer	USERNAME_MIN_LENGTH	The minimum length of a username
	public	Integer	USERNAME_MAX_LENGTH	The maximum length of a username
	public	Integer	PASSWORD_MIN_LENGTH	The minimum length of a password
	public	Integer	PASSWORD_MAX_LENGTH	The maximum length of a password
	public	Integer	CATEGORY_MIN_LENGTH	The minimum length of a category
	public	Integer	CATEGORY_MAX_LENGTH	The maximum length of a category
Methods	**Visibility**	**Name**	**Returns**	**Description**
None				

Table 4: Class Main

Class Name	com.github.comp354project.Main			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Launches the application			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	main(String[] args)	void	The entry point of the application

Table 5: Class MyMoneyApplication

Class Name	com.github.comp354project.MyMoneyApplication			
Visibility	Public			
Type	Class			
Inherits	Application			
Implements	N/A			
Description	Entry point for the GUI of the application			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	logger	Logs event information
	public	MyMoneyApplication	application	The GUI entry point variable
	protected	SessionManager	sessionManager	Manages user sessions
	private	ApplicationComponent	component	Used to instantiate and inject classes
	private	Stage	primaryStage	Used to display the GUI
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	MyMoneyApplication	MyMoneyApplication	Constructs the class. Initializes an ApplicationComponent for depedency injection
	public	getScene()	Scene	Returns the current scene
	public	start(Stage primaryStage)	void	Displays the first GUI when the application launches
	private	updateStage(String fxml, String title, int width, int height)	T	Updates the current view
	private	setStageTitle(String title)	void	Sets the view's title
	public	displayLogin()	void	Displays the login view
	public	displaySignUp()	void	Displays the sign up view
	public	displayAccounts()	void	Displays the user accounts view
	public	displayUpdateUser()	void	Displays the update user view
	public	displayAccountDetails(Account account)	void	Displays the account details view
	public	displayAllTransactions(List accounts)	void	Displays all transactions details view

Table 6: Interface IUserService

Class Name	com.github.comp345project.model.user.IUserService			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	no modifier	createUser(User user)	User	User to create a new user
	no modifier	deleteBankAccount(Account account)	void	Deletes a bank account belonging to user
	no modifier	updateUser(User user)	User	Updates the user's info
	no modifier	deleteUser(User user)	void	Deletes a user

Table 7: Class User

Class Name	com.github.comp345project.model.user.User			
Type	Class			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	private	Integer	ID	User identification number
	private	String	username	Username of user
	private	String	password	User's password
	private	String	firstName	User's first name
	private	String	lastName	User's last name
	private	String	email	User's email
	private	String	address	User's address
	private	String	phone	User's phone number
	private	ForeignCollection	accounts	Accounts belonging to user
Method	Visibility	Name	Returns	Description
	N/A			

Table 8: Class UserService

Class Name	com.github.comp345project.model.user.UserService			
Type	Class			
Inherits	N/A			
Implements	IUserService			
Attributes	Visibility	Data Type	Name	Description
	Private	Logger	logger	Logger used to Log things such as errors
	Private	Dao	userDao	User data access object used to interact with the user data stored in the database
	Private	Dao	accountDao	Account data access object used to interact with the account data stored in the database
	Private	IUsernameValidator	usernameValidator	Used to validate a username
	Private	IUserValidator	userValidator	Used to validate a user
	Private	SessionManager	sessionManager	Used to keep track of the currently logged in user
	Private	AccountService	accountService	Used to interact with the account service layer
Method	Visibility	Name	Returns	Description
	public	UserService(Dao userDao,Dao accountDao, SessionManager sessionManager, AccountService accountService)	N/A	constructor used to create UserService
	public	createUser(User user)	User	User to create a new user
	public	getUser(String username)	User	Get's a user by their username
	public	deleteBankAccount(Account account)	void	Deletes a bank account belonging to user
	public	updateUser(User user)	User	Updates the user's info
	public	deleteUser(User user)	void	Deletes a user

Table 9: Class UserServiceModule

Class Name	com.github.comp345project.model.user.UserServiceModule			
Type	Class			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	provideUserService(UserService userService)	IUserService	Provides a UserService

Table 10: Interface ICategoryNameValidator

Class Name	com.github.comp345project.model.validators.ICategoryNameValidator			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	validateCategory(String category, String message)	List	Validates transaction categories

Table 11: Interface INameValidator

Class Name	com.github.comp345project.model.validators.INameValidator			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	validateName(String name, String message)	List	Validates first and last name of user

Table 12: Interface INameValidator

Class Name	com.github.comp345project.model.validators.INameValidator			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	validatePassword(String password, String message)	List	Validates that password meets required criteria

Table 13: Interface INameValidator

Class Name	com.github.comp345project.model.validators.INameValidator			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	validateUsername(String username, String message)	List	Validates that username meets required criteria

Table 14: Interface IUserValidator

Class Name	com.github.comp345project.model.validators.IUserValidator			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	validateUser(User user)	List	Validates the user data

Table 15: Class StringLengthValidator

Class Name	com.github.comp345project.model.validators.StringLengthValidator			
Type	Class			
Inherits	ICategoryNameValidator, IUsernameValidator, IPasswordValidator, INameValidator			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	private	Integer	minLength	Minimum length of String
	private	Integer	maxLength	Maximum length of String
Method	Visibility	Name	Returns	Description
	public	StringLengthValidator(intminLength, int maxLength)	N/A	constructor
	public	validateName(String name, String message)	List	validates first and last name of user
	public	validateCategory(String category, String message)	List	validates the category of a transaction
	public	validatePassword(String password, String message)	List	validates that password meets criteria
	public	validateUsername(String username, String message)	List	validates that username meets criteria
	public	validate(String string, String paramName, String message)	List	validates the length of a string

Table 16: Class UserValidator

Class Name	com.github.comp345project.model.validators.UserValidator			
Type	Class			
Inherits	IUserValidator			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	private	IUsernameValidator	usernameValidator	used to validate the user
	private	IPasswordValidator	passwordValidator	used to validate the user's password
	private	INameValidator	nameValidator	used to validate the first and last name of user
Method	Visibility	Name	Returns	Description
	public	UserValidator(IUsernameValidatorusernameValidator, IPasswordValidator passwordValidator, INameValidator nameValidator)	N/A	Constructor
	public	validateUser(Useruser)	List	validates user attributes

Table 17: Class ValidatorFactory

Class Name	com.github.comp345project.model.validators.ValidatorFactory			
Type	Class			
Inherits	N/A			
Implements	N/A			
Attributes	Visibility	Data Type	Name	Description
	N/A			
Method	Visibility	Name	Returns	Description
	public	usernameValidator()	IUsernameValidator	creates a UsernameValidator
	public	passwordValidator()	IPasswordValidator	createsa PasswordValidator
	public	categoryNameValidator()	ICategoryNameValidator	creates a CategoryNameValidato
	public	userValidator()	IUserValidator	creates a UserValidator

Table 18: Class Account

Class Name	com.github.comp354project.model.account.Account			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Used to hold the account information of the user			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Integer	ID	bank account identification number
	private	String	type	type of bank account (chequing, savings, ect)
	private	Double	balance	Amount inside the account
	private	User	user	name of the user
	private	ForeignCollection<Transaction>	transactions	transaction object
Methods	**Visibility**	**Name**	**Returns**	**Description**
	none	none	none	none

Table 19: Class AccountService

Class Name	com.github.comp354project.model.account.AccountService			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	IAccountService			
Description	Class used to request information from the bank database in order to add or delete an account to myMoney application			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	logger	logger object attribute used to keep track of events
	private	Dao<Transaction,Integer>	transactionDao	Dao object used to query transactions
	private	Dao<User,Integer>	userDao	Dao object used to query users
	private	IRemoteAccountService	remoteAccountService	Dao object used to query remote accounts
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	addAccount	Account	method to request bank information from the database
	public	deleteAccount	void	method to delete a particular account from myMoney application
	public	transform	Account	method to create the appropriate banking info to display for the myMoney app based on the retrieved banking info
	public	Transaction	transform	method to create the appropriate transaction info to display for the myMoney app based on the retrieved banking info

Table 20: Class AccountServiceModule

Class Name	com.github.comp354project.model.account.AccountServiceModule			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	used to return need objects for account and transaction needs			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	None	none	none	none
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	provideTransactionService	transactionService	return transactionService Object
	public	provideAccountService	accountService	returns accountService Object

Table 21: Interface IAccountService

Class Name	com.github.comp354project.model.account.IAccountService			
Visibility	Public			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Description	interface class for adding and deleting an account			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None	None	None	none	none
Methods	**Visibility**	**Name**	**Returns**	**Description**
	N/A	addAccount	N/A	none
	N/A	deleteAccount	N/A	none

Table 22: Interface ITransactionService

Class Name	com.github.comp354project.model.account.ITransactionService			
Visibility	Public			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Description	interface class to updating transactions based on categories			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None	None	None	None	None
Methods	**Visibility**	**Name**	**Returns**	**Description**
	N/A	updateTransactionCategory	Transaction	N/A

Table 23: Class Transaction

Class Name	com.github.comp354project.model.account.Transaction			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Class used to contain the attributes needed to hold a transaction's details			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Integer	date	date of a transaction
	private	Double	amount	dollar amount of a transaction
	private	String	type	the type of a transaction
	private	String	category	the category of a transaction
	private	Integer	sourceID	ID number
	private	Integer	destinationID	ID number
	private	Account	account	name of the account
Methods	**Visibility**	**Name**	**Returns**	**Description**
	None	None	None	None

Table 24: Class TransactionService

Class Name	com.github.comp354project.model.account.TransactionService			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	ITransactionService			
Description	class used to help with transaction changes			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	logger	object used to interact with TransactionService class
	private	Dao<Transaction,Integer>	transactionDao	object used to perform methods related to transactions
	private	ICategoryNameValidator	categoryValidator	object used to validate if a category is correct
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	TransactionService	N/A	constructor
	public	updateTransactionCategory	Transaction	used to update a specific transation

Table 25: Class AccountDoesNotExistException

Class Name	com.github.comp354project.model.account.exceptions.AccountDoesNotExistException			
Visibility	Public			
Type	class			
Inherits	RuntimeException			
Implements	N/A			
Description	Exception thrown when a remote account does not exist			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	GetRemoteAccountRequest	request	The request that was sent
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	AccountDoesNotExistException	AccountDoesNotExistException(String message, Throwable cause, GetRemoteAccountRequest request)	Constructs the exception

Table 27: Class GetRemoteAccountRequest

Class Name	com.github.comp354project.model.account.remote.GetRemoteAccountRequest				
Visibility	Public				
Type	Public				
Inherits	N/A				
Implements	N/A				
Description	Retrieve the remote account request				
Attributes	**Visibility**	**Data Type**	**Name**	**Description**	
	Private	Integer	accountID	Identification of an account	
Methods	**Visibility**	**Name**	**Returns**	**Description**	**Throws**
None					

Table 28: Class GetRemoteAccountResponse

Class Name	com.github.comp354project.model.account.remote.GetRemoteAccountResponse				
Visibility	Public				
Type	Public				
Inherits	N/A				
Implements	N/A				
Description	Retrieve the response for the remote account request				
Attributes	**Visibility**	**Data Type**	**Name**	**Description**	
	Private	RemoteAccount	account	The remote account	
Methods	**Visibility**	**Name**	**Returns**	**Description**	**Throws**
None					

Table 26: Class AccountExistsException

Class Name	com.github.comp354project.model.account.exceptions.AccountExistsException			
Visibility	Public			
Type	class			
Inherits	RuntimeException			
Implements	N/A			
Description	Exception thrown when an account is already in use			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Account	account	The account that was sent
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	AccountExistsException	AccountExistsException(String message, Throwable cause, Account account)	Constructs the exception

Table 34: Class AuthenticationModule

Class Name	com.github.comp354project.model.auth.AuthenticationModule			
Type	class			
Inherits	N/A			
Implements	N/A			
Description	The authentication module to provide the dependencies			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	ValidationException(String message, Throwable cause, @Singular List errors)	IAuthenticationService	Provides the authentication service implementation

Table 29: Interface IRemoteAccountService

Name	com.github.comp354project.model.account.remote.IRemoteAccountService			
Visibility	Public			
Type	Public			
Inherits	N/A			
Implements	N/A			
Description	The interface for the remote account service (request and response)			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Throws**	**Description**
	Public	IRemoteAccountService	ValidationException	Remote account service

Table 30: Class RemoteAccount

Class Name	com.github.comp354project.model.account.remote.RemoteAccount				
Visibility	Public				
Type	Public				
Inherits	N/A				
Implements	N/A				
Description	The remote account with details				
Attributes	**Visibility**	**Data Type**	**Name**	**Description**	
	Private	Integer	ID	ID of the account	
	Private	String	bankName	Name of the bank	
	Private	String	Type	Type of the account	
	Private	Double	balance	Balance of the account	
	Private	ForeignCollection	transactions	Transactions of the account	
Methods	**Visibility**	**Name**	**Returns**	**Description**	**Throws**
None					

Table 31: Class RemoteAccountModule

Class Name	com.github.comp354project.model.account.remote.RemoteAccountModule			
Visibility	Public			
Type	Public			
Inherits	N/A			
Implements	N/A			
Description	The module for remote account class			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	Default	provideRemoteAccountService	remoteAccountService	Module provide the remote account service

Table 32: Class RemoteAccountService

Class Name	com.github.comp354project.model.account.remote.RemoteAccountService			
Visibility	Public			
Type	Public			
Inherits	N/A			
Implements	IRemoteAccountService			
Description	The services that the remote account can provide			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	Logger	logger	Gets the log of the Remote AccountService.class
	Private	Dao <Remote Account, Integer >	Remote AccountDao	RemoteAccountDoa
Methods	**Visibility**	**Name**	**Throws**	**Description**
	Public	RemoteAccountService(Dao <RemoteAccount,Integer >remoteAccountDao)	N/A	The constructor class for Remote AccountService
	Public	getAccount(GetRemoteAccountRequest request)	Validation Exception	Return the account information if there is a request for it and if it exists

Table 33: Class RemoteTransaction

Class Name	com.github.comp354project.model.account.remote.RemoteTransaction			
Visibility	Public			
Type	Public			
Inherits	N/A			
Implements	N/A			
Description	The remote transaction class			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	Integer	ID	Identification of the remote transaction
	Private	Integer	date	Date of the transaction
	Private	Double	amount	Amount of money transitioned
	Private	String	type	Type of transaction
	Private	Integer	SourceID	Identification of the source where the money was originally resided
	Private	Integer	destinationID	Identification of the destination where the money will be transitioned
	Private	Remote	account	The main account of the user
Methods	**Visibility**	**Name**	**Returns**	**Description**
None				

Table 35: Class AuthenticationService

Class Name	com.github.comp354project.model.auth.AuthenticationService			
Type	class			
Inherits	N/A			
Implements	IAuthenticationService			
Description	Service to authenticate users			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	Logger	Logger
	private	Dao<User, Integer>	userDao	The service Dao to interact with the db
	private	IUsernameValidator	usernameValidator	Used to validate a username
	private	IPasswordValidator	passwordValidator	Used to validate a password
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	AuthenticationService(Dao userDao)	AuthenticationService	Constructs an AuthenticationService
	public	authenticate(String username, String password)	User	Authenticates a user
	private	getUser(String username)	User	Finds a user by username

Table 36: Class IAuthenticationService

Class Name	com.github.comp354project.model.auth.IAuthenticationService			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Description	Service to authenticate users			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	authenticate(String username, String password)	User	Authenticates a user

Table 37: Class SessionManager

Class Name	com.github.comp354project.model.auth.SessionManager			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Service to manage user sessions			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	IAuthenticationService	authenticationService	The authentication service
	private	User	user	The current user logged in
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	SessionManager(IAuthenticationService authenticationService)	SessionManager	Constructs the SessionManager
	public	login(String username, String password)	User	Authenticates a user
	public	logout()	void	Disconnects a user
	public	isLoggedIn()	boolean	Checks if a user is logged in

Table 38: Class AuthenticationException

Class Name	com.github.comp354project.model.auth.exceptions.AuthenticationException			
Visibility	Public			
Type	class			
Inherits	Exception			
Implements	N/A			
Description	Exception thrown when a user did not authenticate properly			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	String	username	The username used
	private	String	password	The password used
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	AccountExistsException	AuthenticationException(String message, Throwable cause, String username, String password)	Constructs the exception

Table 39: Class AuthorisationException

Class Name	com.github.comp354project.model.auth.exceptions.AuthorisationException			
Visibility	Public			
Type	class			
Inherits	Exception			
Implements	N/A			
Description	Exception thrown when a user is not authorised to execute/read			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	User	user	The user used
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	AuthorisationException	AuthorisationException(String message, Throwable cause, User user)	Constructs the exception

Table 40: Class UserLoggedInException

Class Name	com.github.comp354project.model.auth.exceptions.UserLoggedInException			
Visibility	Public			
Type	class			
Inherits	Exception			
Implements	N/A			
Description	Exception thrown when a user needs to be logged in			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	User	user	The user used
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	UserLoggedInException	UserLoggedInException(String message, Throwable cause, User user)	Constructs the exception

Table 41: Class DaoModule

Class Name	com.github.comp354project.model.dao.DaoModule			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	DAO module to bind interfaces to their interfaces and provide them to the classes that require them			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	logger	Logs event information
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	provideRemoteAccountDao(IConnectionProvider connectionProvider)	Dao<RemoteAccount, Integer>	Returns the implementation of a RemoteAccountDao
	public	provideAccountDao(IConnectionProvider connectionProvider)	Dao<Account, Integer>	Returns the implementation of an AccountDao
	public	provideTransactionDao(IConnectionProvider connectionProvider)	Dao<Transaction, Integer>	Returns the implementation of a TransactionDao
	public	provideUserDao(IConnectionProvider connectionProvider)	Dao<User, Integer>	Returns the implementation of a UserDao

Table 42: Class UserLoggedInException

Class Name	com.github.comp354project.model.exceptions.DatabaseException			
Visibility	Public			
Type	class			
Inherits	Exception			
Implements	N/A			
Description	Exception thrown when a the database has an exception			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	User	user	The user used
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	DatabaseException(String message)	DatabaseException	Constructs the exception
	public	DatabaseException(Throwable inner)	DatabaseException	Constructs the exception
	public	DatabaseException(String message, Throwable inner)	DatabaseException	Constructs the exception

Table 43: Class ValidationError

Class Name	com.github.comp354project.model.exceptions.ValidationError			
Visibility	Public			
Type	class			
Inherits	N/A			
Implements	N/A			
Description	Class used to build exceptions			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	String	message	The error message
	private	String	parameterName	The parameter in error
	private	String	parameterValue	The parameter in error value
Methods	**Visibility**	**Name**	**Returns**	**Description**
None				

Table 44: Class ValidationException

Class Name	com.github.comp354project.model.exceptions.ValidationException			
Visibility	Public			
Type	class			
Inherits	Exception			
Implements	N/A			
Description	Exception used to hold multiple exceptions			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	List<ValidationException>	errors	The exception list
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	ValidationException(String message, Throwable cause, @Singular List errors)	ValidationException	Constructs the exception

Table 45: Class ConnectionModule

Class Name	com.github.comp354project.model.sqlite.ConnectionModule			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Module that creates a connection to the database			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	protected	provideConnection(ConnectionProvider connectionProvider)	IConnectionProvider	Returns the implementation of a ConnectionProvider

Table 46: Class ConnectionProvider

Class Name	com.github.comp354project.model.sqlite.ConnectionProvider			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	IConnectionProvider			
Description	Instatiates a connection to an SQLite database			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Logger	logger	Logs events
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	ConnectionProvider()	ConnectionProvider	Constructs the class
	public	getConnectionSource()	JdbcConnectionSource	Returns a database connection source

Table 47: Interface IConnectionProvider

Class Name	com.github.comp354project.model.sqlite.IConnectionProvider			
Visibility	Public			
Type	Interface			
Inherits	N/A			
Implements	N/A			
Description	Instatiates a connection to a database			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	getConnectionSource()	JdbcConnectionSource	Returns a database connection source

Table 48: Account Details Controller

Class Name	com.github.comp354project.viewController.AccountDetailsController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable			
Description	Populates the views with details and transactions of the specified account			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	TransactionTable	transactionTable	table of transactions
	private	Label	accountBalance	balance of the specified account
	private	Label	accountDescription	account ID and name of the bank
	private	Label	accountType	type of the account
	private	Account	account	specified acount
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	required by interface, but not actually used
	public	setAccount(Account account)	void	sets the account to be displayed
	public	goToAccountList()	void	returns to the account list view

Table 49: Account List Controller

Class Name	com.github.comp354project.viewController.AccountListController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable			
Description	Populates the views with the list of all accounts associated to the user			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	TextField	accountIdTxt	text field for account ID
	private	TableView<AccountDisplayModel>	accountsTable	display model for the accounts
	private	TableColumn<AccountDisplayModel, Integer>	idCol	display model for the acccount ids
	private	TableColumn<AccountDisplayModel, String>	bankNameCol	display model for the bank names
	private	TableColumn<AccountDisplayModel, String>	typeCol	display model for the account type
	private	TableColumn<AccountDisplayModel, Double>	balanceCol	display model for the account balances
	private	List<Account>	accounts	list of the accounts
	private	ObservableList<AccountDisplayModel>	tableData	the data used to feed to the table
	protected	SessionManager	sessionManager	manager for the current logged in user(s)
	protected	IAccountService	accountService	an instance of the AccountService object for model access
	protected	IUserService	userService	an instance of the UserService object for model access
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	required by interface, populates the columns
	public	setAccounts(List¡Account¿ accounts)	void	sets the accounts to be displayed
	public	selectAccount(MouseEvent event)	void	called when a user clicks an account, sets the selected account.
	public	displayAllTransactions()	void	goes to the display All Transactions view
	public	logout()	void	logs out the user and returns to login menu
	public	updateUserInfo()	void	goes to update User info view
	public	addAccount()	void	associates a new account to the user account
	public	removeAccount()	void	sets the accounts to be displayed
	public	setAccounts(List¡Account¿ accounts)	void	sets the accounts to be displayed

Table 50: Update User Account Controller

Class Name	com.github.comp354project.viewController.UpdateUserAccountController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable			
Description	Allows to update user account information and delete the account			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	usernameTxt	username of the logged-in user
	private	JFXTextField	addressTxt	address input field
	private	JFXTextField	emailTxt	email input field
	private	JFXTextField	lastnameTxt	last name input field
	private	JFXTextField	firstnameTxt	first name input field
	private	JFXTextField	phoneNumberTxt	phone number input field
	private	JFXPasswordField	passwordField	password input field
	private	JFXPasswordField	passwordRepeatField	password repeat input field
	protected	IUserService	userService	injected user service object
	protected	SessionManager	sessionManager	injected session manager object
	private	User	user	the logged-in user
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	populates the user and input fields
	public	goBackToAccountList(ActionEvent)	void	returns to the account list view
	public	setUserInfo()	void	Updates the input fields
	public	updateUser(ActionEvent)	void	Calls the userService to update the user
	public	deleteUser(ActionEvent)	void	Calls the userService to delete the user

Table 51: All Transactions Controller

Class Name	com.github.comp354project.viewController.AllTransactionsController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable			
Description	Populates the views with all transactions			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	totalBalanceLabel	balance of the specified account
	private	Label	descriptionLabel	account ID and name of the bank
	private	Label	accountType	type of the account
	private	TransactionTable	transactionTable	table of transactions
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	required by interface, but not actually used
	public	setAccounts(List¡Account¿ accounts)	void	sets the accounts to be displayed
	public	gotoAccountList(MouseEvent event)	void	returns to the account list view

Table 52: Transaction Table Controller

Class Name	com.github.comp354project.viewController.TransactionTableController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable			
Description	Allows to display and edit transactions			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	TableColumn<TransactionDisplayModel, Integer>	idCol	the transaction id column
	private	TableColumn<TransactionDisplayModel, String>	dateCol	the transaction date column
	private	TableColumn<TransactionDisplayModel, String>	amountCol	the transaction amount column
	private	TableColumn<TransactionDisplayModel, String>	categoryCol	the transaction category column
	private	TableColumn<TransactionDisplayModel, String>	typeCol	the transaction type column
	private	TableView<TransactionDisplayModel>	transactionTableView	the transaction table view
	private	ObservableList<TransactionDisplayModel>	tableData	the display models used to feed to the table
	private	List< *Transaction* >	transactions	the data used to construct dipslay models from
	private	ITransactionService	transactionService	a transaction service object
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	populates the table view
	public	hideAccountIDColumn()	void	hides the ID column of the table view
	public	setTransactions(List<Transaction>)	void	updates the transactions and table view data

Table 53: Sign up controller

Class Name	com.github.comp354project.viewController.SignUpController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable, EventHandler<KeyEvent>			
Description	Allows to sign up a new user			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	JFXTextField	usernameTxt	username input field
	private	JFXTextField	lastnameTxt	last name input field
	private	JFXTextField	firstnameTxt	first name input field
	private	JFXPasswordField	passwordField	password input field
	private	JFXPasswordField	passwordRepeatField	password repeat input field
	protected	IUserService	userService	injected user service object
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	required by the interface, empty body
	public	signUp(ActionEvent)	void	calls the signUp() method
	public	handle(KeyEvent)	void	handles user keyboard input
	private	signUp()	void	Calls the user service to create a new user
	private	validatePasswords()	void	ensures that the contents of the password fields match

Table 54: Login controller

Class Name	com.github.comp354project.viewController.LoginController			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	Initializable, EventHandler<KeyEvent>			
Description	Allows the users to log in			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	JFXTextField	usernameTxt	username input field
	private	JFXPasswordField	passwordField	password input field
	protected	SessionManager	userService	injected user service object
	protected	IUserService	userService	injected user service object
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	initialize()	void	required by the interface, empty body
	public	login(ActionEvent)	void	calls the login() method
	public	handle(KeyEvent)	void	handles user keyboard input
	public	signUp(ActionEvent)	void	calls the user service to create a new user
	private	login()	void	calls the sessionManager to login the user

Table 55: AlertHelper

Class Name	com.github.comp354project.viewController.helper.AlertHelper			
Visibility	Public			
Type	Class			
Inherits	N/A			
Implements	N/A			
Description	Creates an alert to notify the user of an error			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
None				
Methods	**Visibility**	**Name**	**Returns**	**Description**
	public	generateAlert(Alert.AlertType alertType, String title, String header, String content)	Alert	Returns a new alert composed of the passed content
	public	generateErrorAlert(String title, String header, String content)	Alert	return a new alert of type ERROR composed of the passed content
	public	generateErrorAlert(String title, String header, ValidationException exception)	Alert	return a new alert of type ERROR, specifically for our custom validation exception.

4.3 Glossary

Table 56: Glossary

Expression	Definition
Object-Oriented Programming	A programming paradigm which separates entities into objects, and uses the concept of inheritance of properties, polymorphism of objects, encapsulation of objects. We use this paradigm for its maintainability and structural benefits.
MVC - Model-View-Controller Architecture	An architectural pattern which strictly separates components into the model (manages the data and logic), the view (output of the model), and the controller (handling input and passing it to the model or view).
Interface	A component of a system by which other entities (be it humans or other systems) may engage in an exchange of data with the system in question.
API - Application Programming Interface	A protocol or set of functions which serve as a method of communication to a software system. It is a type of interface, and the one by which our system will communicate with the banking institutions' databases.
DAO - Data access object	An object that provides an abstract interface to some type of database or other persistence mechanism.
Dependency Injection (DI)	Software development technique where one object supplies the dependency of another object.
Module	Used in Dagger2 for the injection of dependencies into their classes and their submodules. Modules make the code less coupled.

4.4 Subsystem X

4.4.1 Detailed Design Diagram

UML class diagram depicting the internal structure of the subsystem, accompanied by a paragraph of text describing the rationale of this design.

*Note: The above is a description of what to provide. Need to edit into our own

4.4.2 Units Description

List each class in this subsystem and write a short description of its purpose, as well as notes or reminders useful for the programmers who will implement them. List all attributes and functions of the class.

*Note: The above is a description of what to provide. Need to edit into our own

5 Dynamic Design Scenarios

Describe some (at least two) important execution scenarios of the system using UML sequence diagrams. These scenarios must demonstrate how the various subsystems and units are interacting to achieve a system-level service. Units and subsystems depicted here must be compatible with the descriptions provided in section 3 and 4.

*Note: The above is a description of what to provide. Need to edit into our own

5.1 Dynamic Models of System Interface

We have chosen 3 major functionalities of the system (also known as use cases) in order to portray the interactions between the classes of the system. By using a sequence diagram, this will display the dynamics visually by showcasing the sequences of method calls when a particular use case begins functioning.

5.1.1 Use Case 1: Create User Account

The following scenario describes the actions that occur when the user clicks on the sign up button

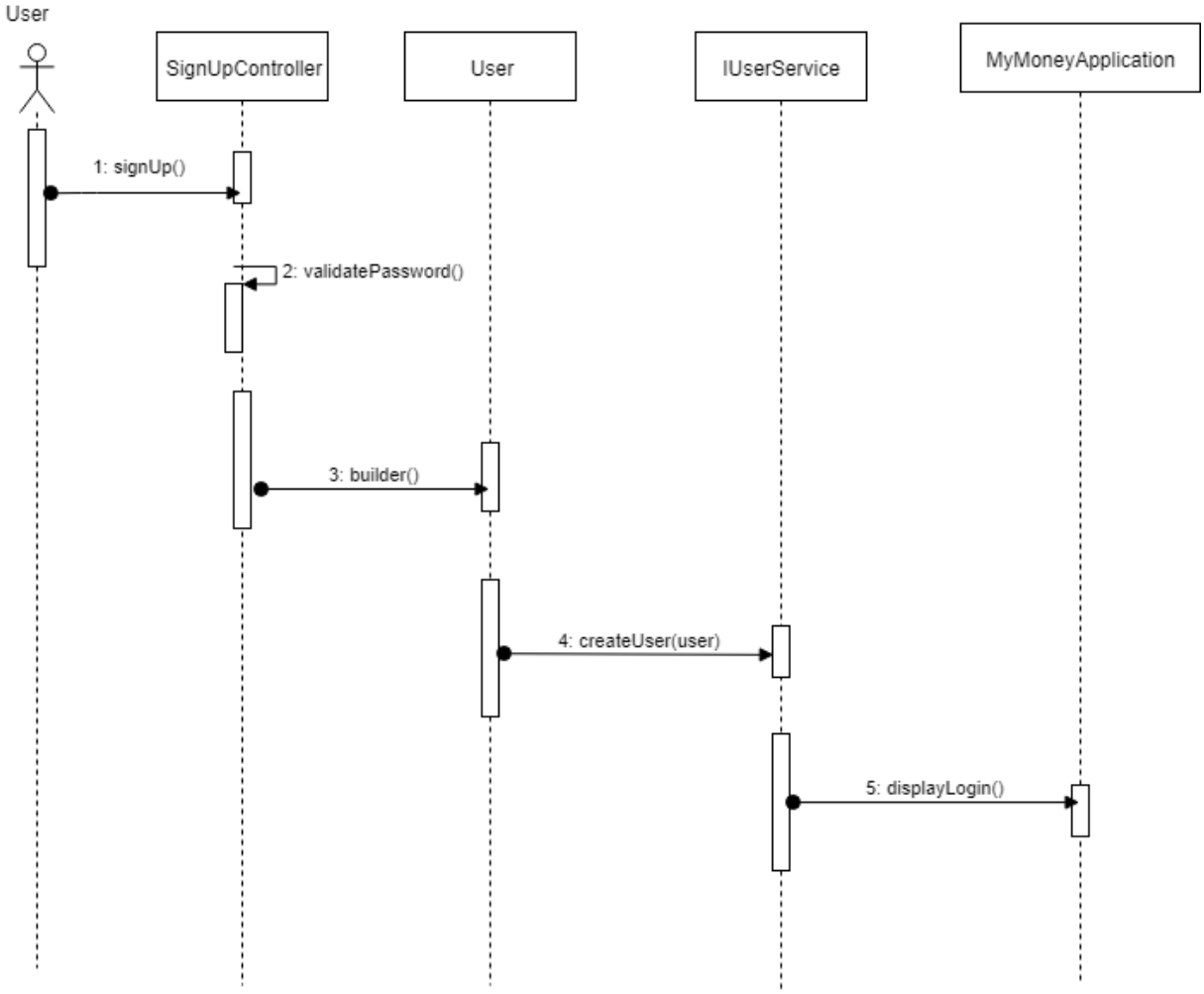


Figure 2: Use case 1 Sequence Diagram

5.1.2 Use Case 3: Add Bank Account to a User Account

The following scenario describes the actions that occur when a user clicks the add button in the account list view.

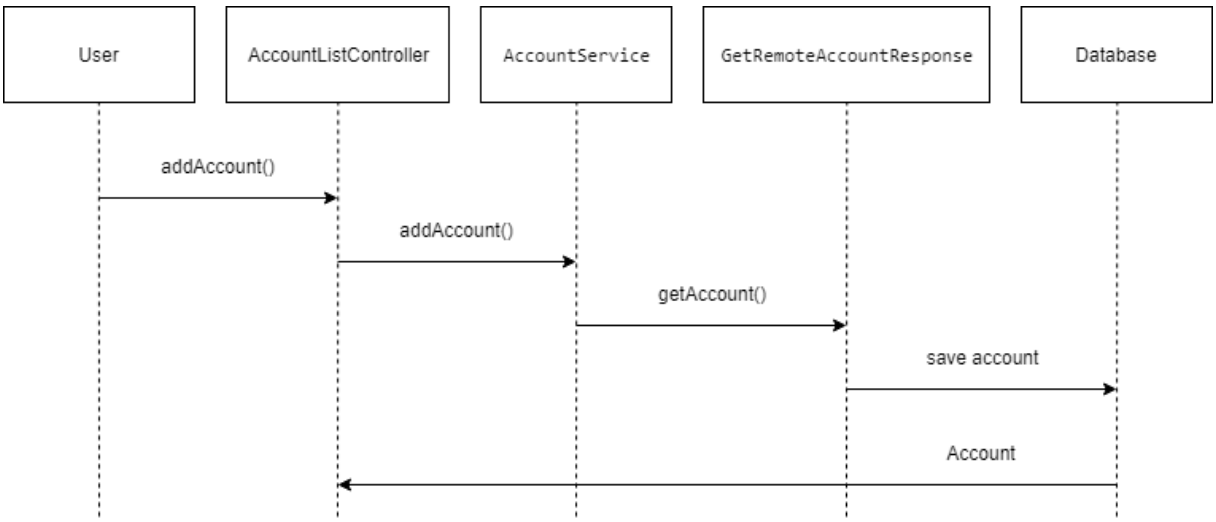


Figure 3: Use case 3 Sequence Diagram

5.1.3 Use Case 5: View Transactions for Specific Bank Account

The following scenario describes the actions that occur when the user clicks the button¡view transactions¿ for a specific bank account.

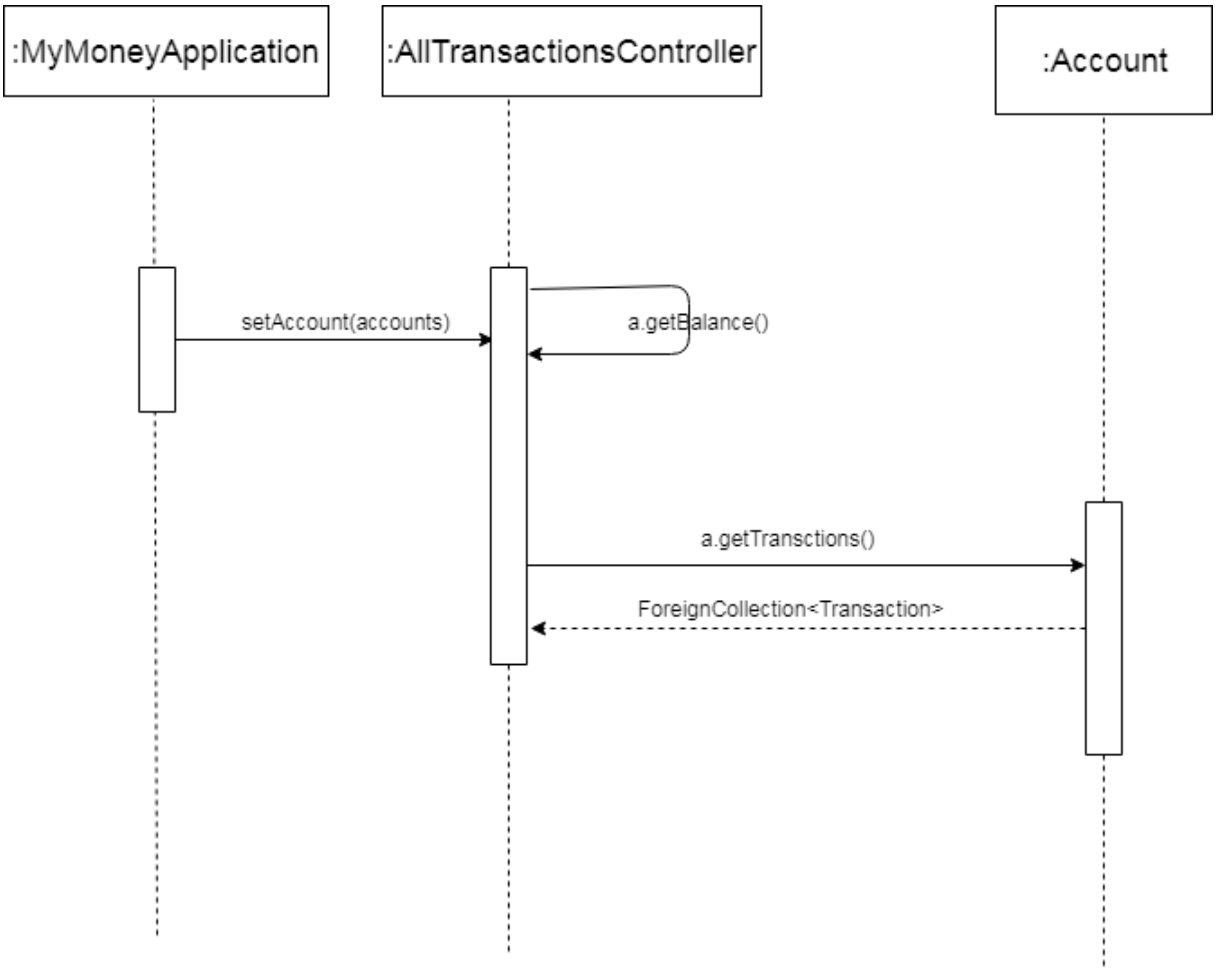


Figure 4: Use case 5 Sequence Diagram

5.1.4 Use Case 6: View All Transactions from all Bank Accounts

The following scenario describes the actions that occur when the user click the button "view all transactions" for viewing all transactions from all bank accounts.

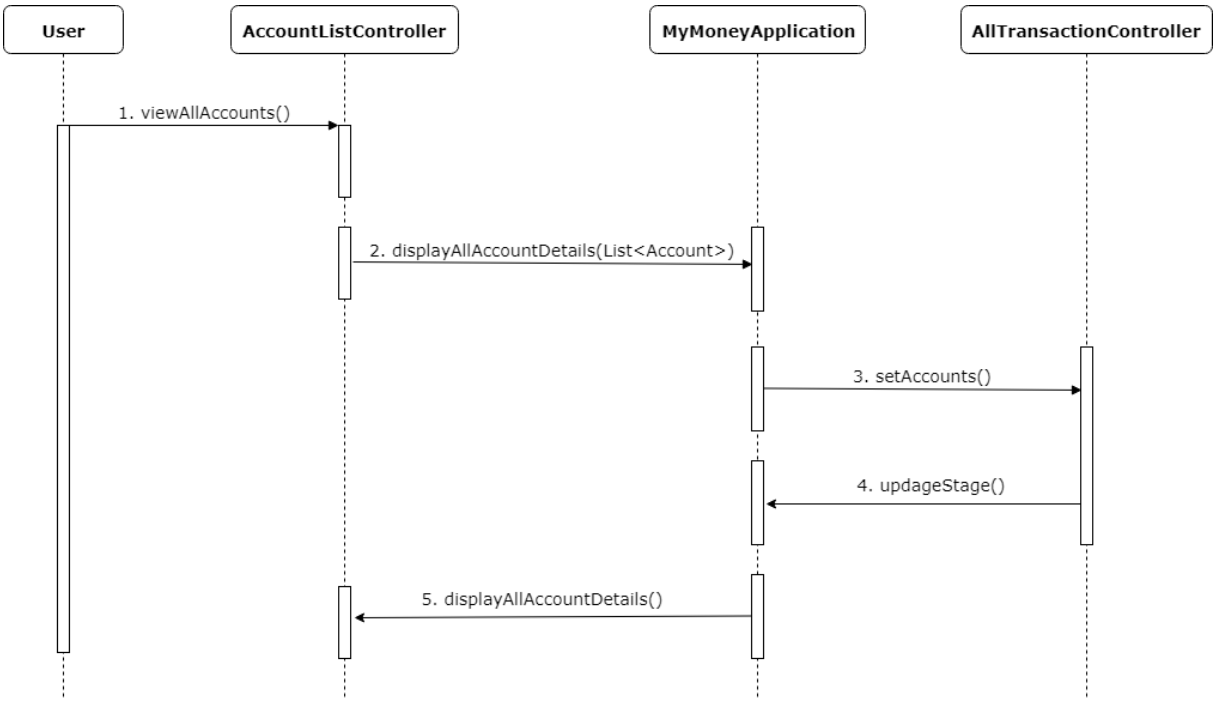


Figure 5: UseCase 6 Sequence Diagram

6 Reference

- User information: As our user and use-cases was based on feedback provided by our developers, our references lie mainly within our own team.
- Craig Larman - Applying UML and Patterns
- Greg Butler's course COMP 354 content
- MIT Curricular Information System Software Requirements Document
- Carnegie Mellon Business Goals
- Use-Case: Oracle
- Google Dagger Github