DESIGN: UNIVERSAL RAFFLE MANAGEMENT SYSTEM

Version 1.0

Team #1

Katie MacEachern

Max Jennings

Logan Brown

Samuel McKinnon

Ethan Rouse

Dylan Coakley

CSCI 485 Software Design

Dr. Man Lin

March 5th, 2018

TABLE OF CONTENTS

INTRODUCTION

1.1 Overview

The purpose of our proposed software, currently referred to as the Universal Raffle Management System, also known as **URMS**, is to provide a system that an organization may use to facilitate the sale of tickets and simplify management for a raffle. The pursuit for this type of system stemmed from the troubles faced by our client while managing their raffle. The current procedure used by our client is to extract buyer information from a paper ticket, and manually input this information into a spreadsheet. This manual inputting of information leads to information inconsistency. Presently, our client stores ticket and seller information across various spreadsheets. This variety of storage locations causes unnecessary complexity within their current system. The sale of their tickets spans a large area, and some communities in this area are fairly rural and distant. This makes it difficult for our client to deliver paper tickets to these communities. It was for these reasons, as well as others, that our client sought to find a new system that allows the sale of digital tickets in addition to the conventional physical tickets, and that provides better organization and analysis of their raffle.

1.2 Scope

The Universal Raffle Management System is composed of two primary components: a client-side web application that receives user input and requests, and a server-side application that responds to requests and maintains a centralized database. The system features can be broken up into two groups as well: core features, which are essential to the function of the application, and additional features, which are meant to add extra functionality. The following list includes all of the features currently designated for inclusion in the final release of the **URMS**:

Core Features:

1. Welcome & General Information Page
   Provides greeting and general information about the raffle and the associated sponsors.
2. Help Requests
   Enables buyers to request aid from an administrator if something goes wrong, such as not receiving a confirmation email for their ticket purchase. Sellers may also request to increase/reduce their current amount of tickets, to which an administrator can respond.
3. Seller Registration
   Allows a user to register to be able to sell tickets, this user must first be approved by an administrator before they are able to sell.

4. Sale of Raffle Tickets
   Allows sellers to sell tickets by inputting the information of a ticket purchaser.
5. Sale Email Confirmation Notification
   All buyers are sent a confirmation of their ticket purchases by email
6. Seller Transaction Tracking
   All seller transactions will be logged by the system, allowing the seller to see how many tickets they have available.
7. Administrator Ticket Allocation
   Allows administrator to distribute tickets between all of their sellers. Modification of distributed tickets, by requesting or reducing tickets, is also supported.
8. Administrator Raffle Statistics Reports
   Generate reports on the number of tickets sold daily, the highest performing sellers, etc.

Additional Features:

1. SMS Ticket Confirmation Notifications

   Add SMS ticket confirmations in addition to the e-mail ticket confirmations.

2. User Account Information Modification

   Allows users to edit the information associated with their account, such as e-mail address or phone number.

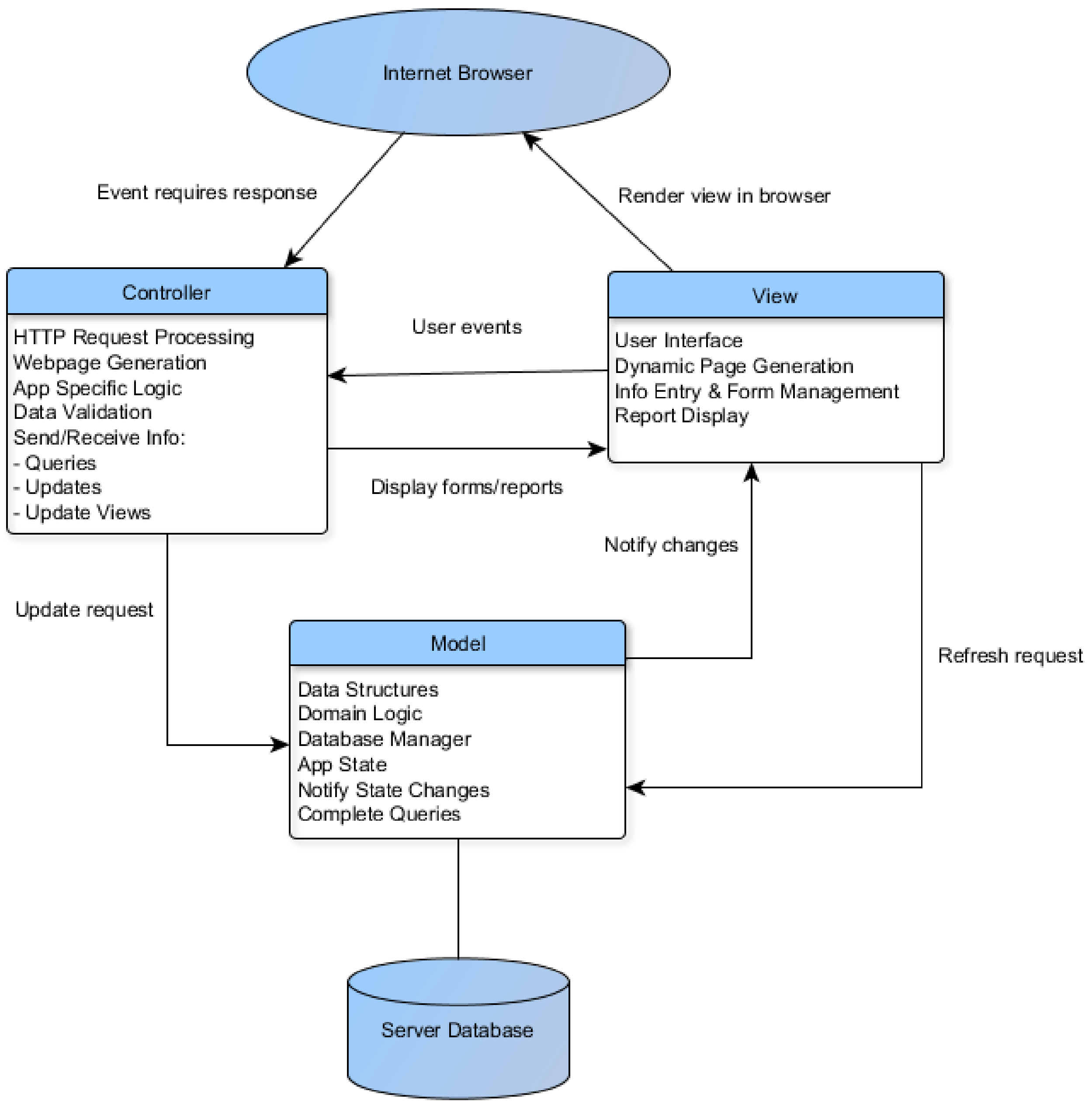
1.3 Constraints

The greatest constraint for the Universal Raffle Management System project is time. There is roughly one month allocated to the entire development, testing, and documentation of this project between six (6) team members. This development process includes both the front-end website and the back-end server application and associated database. Collectively, the development team has little experience with web development in the past, so a significant portion of this time will be dedicated to learning how to approach web development efficiently. Consequently, this makes time an even larger constraint. This may result in fewer features in the initial release, however the core functionality of the system should not be affected.

SYSTEM ARCHITECTURE

2.1 Architectural Design

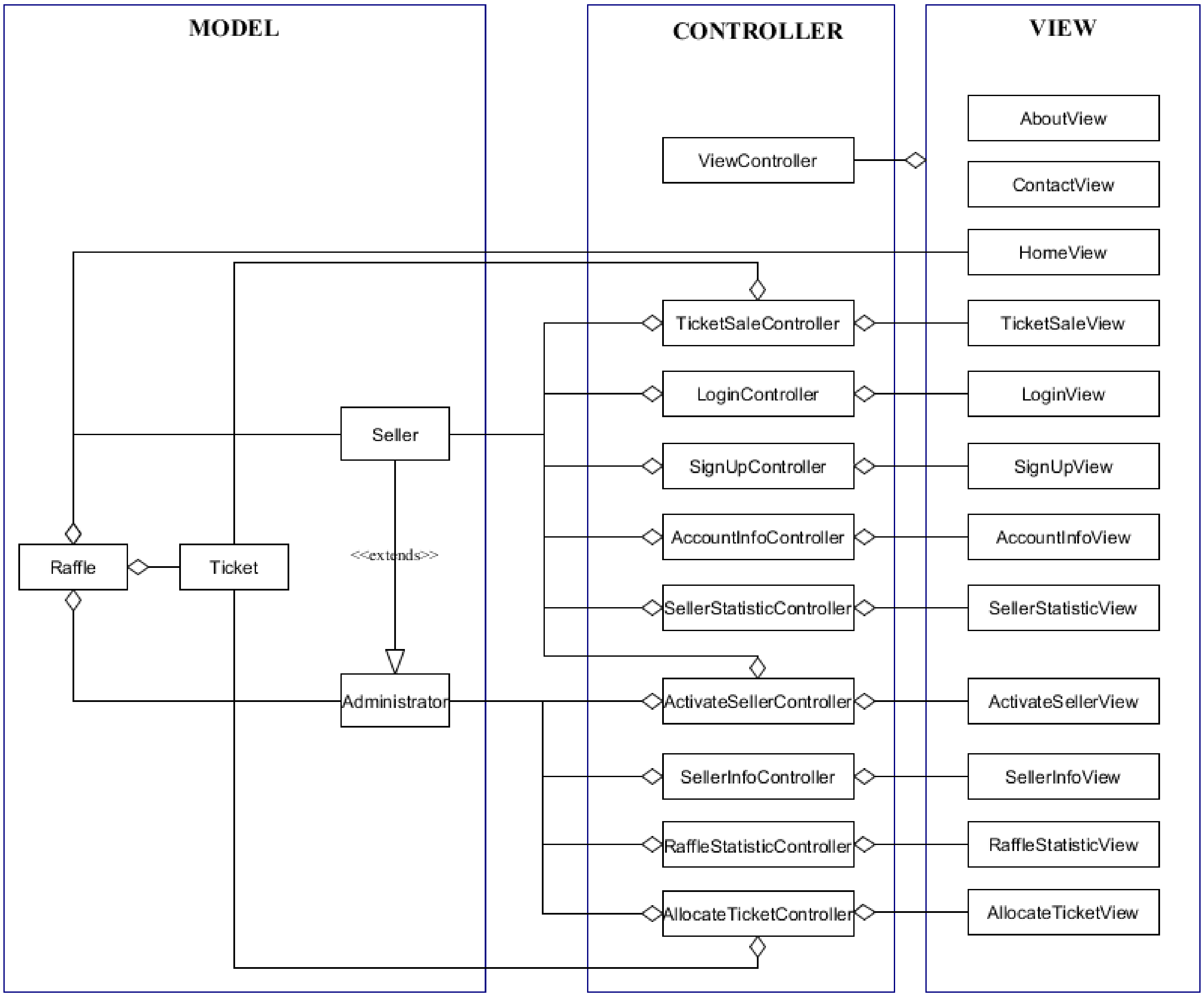
The **URMS** will use a Model-View-Controller (MVC) architecture as it is the most appropriate for the system. The system architecture will be structured in the following way:



2.2 Decomposition Description

The following class diagram demonstrates the relationships between the classes contained within the Model, View, and Controller portions of the architecture. Each of these classes will be discussed in depth in Section 3.3, which details the structural model of the system, by showing all attributes and methods of each class, and how they interact with other classes.

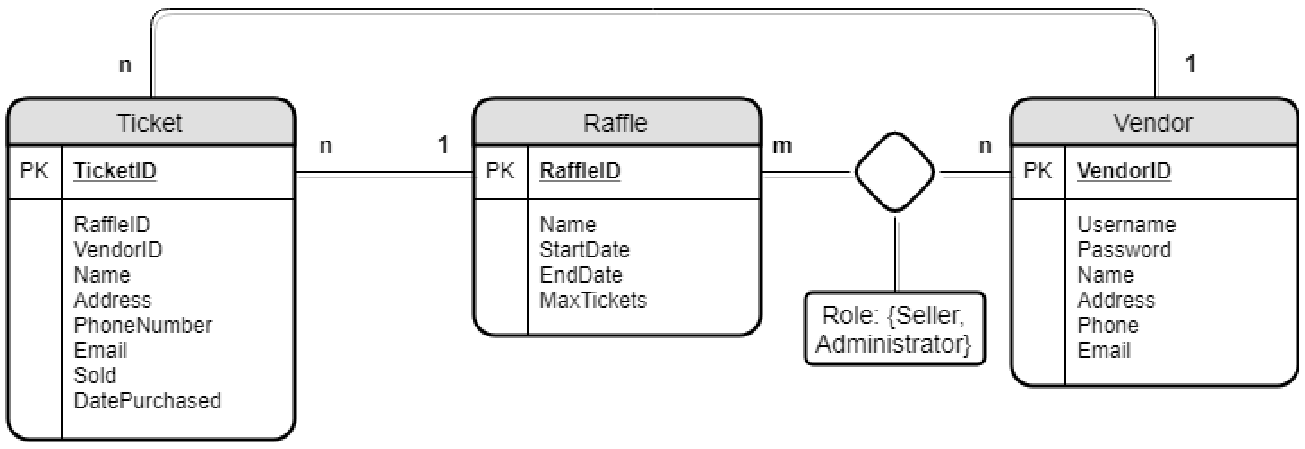
URMS CLASS DIAGRAM



All classes in the Model portion represents data structures that contain the data of the system, as well as methods to manipulate this data. No classes within the Model portion have any dependencies on any class within the Controller nor View portion. The classes contained within the Controller portion are responsible for providing model data to their associated view. They interpret user events such as button clicks as well. All Controller classes depend on their associated view, and some part of the model. Each class within the View portion can display model data, provides forms for data entry, and sends user events such as button clicks to their associated controller.

The following is an Entity-Relationship Diagram, which gives a detailed description of the structure of the tables within the server database, shown above in the MVC architecture of the system.

Server Database E/R Diagram



2.3 Architectural Alternatives

There was one other possible architecture that was considered for our system. The alternative architecture that was considered was the Client-Server architecture. In the Client-Server architecture, the functionality of the system is organized into individual services, with each of these services delivered from a separate server. The clients are the users of these services, and access these servers to make use of them. This proposed architecture came from the fact that the **URMS** is a web-based application. However, our system does not provide an abundant amount of services to clients. Therefore, we thought that it would not be appropriate for our system.

2.4 Design Rationale

Unnecessary complexity occurs frequently in software development. Complexity leads to software that is buggy, and difficult to understand. The easiest way to make code overly complex is to have dependencies located everywhere. On the other hand, removing unnecessary dependencies makes clean code which is less buggy and easier to maintain because it is reusable without modification. The purpose of the controller is to remove the view dependency from the model. By removing the view dependency from the model, the model code becomes much more clear and concise. This is the greatest advantage of the MVC architecture. Another big reason for choosing the MVC architecture for our system is to accommodate the CodeIgniter Application Development Framework. The CodeIgniter Framework allows teams to develop projects much faster than they could if they were writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. CodeIgniter itself is based on the MVC development pattern.

DESIGN MODEL

3.1 System Classes and Objects

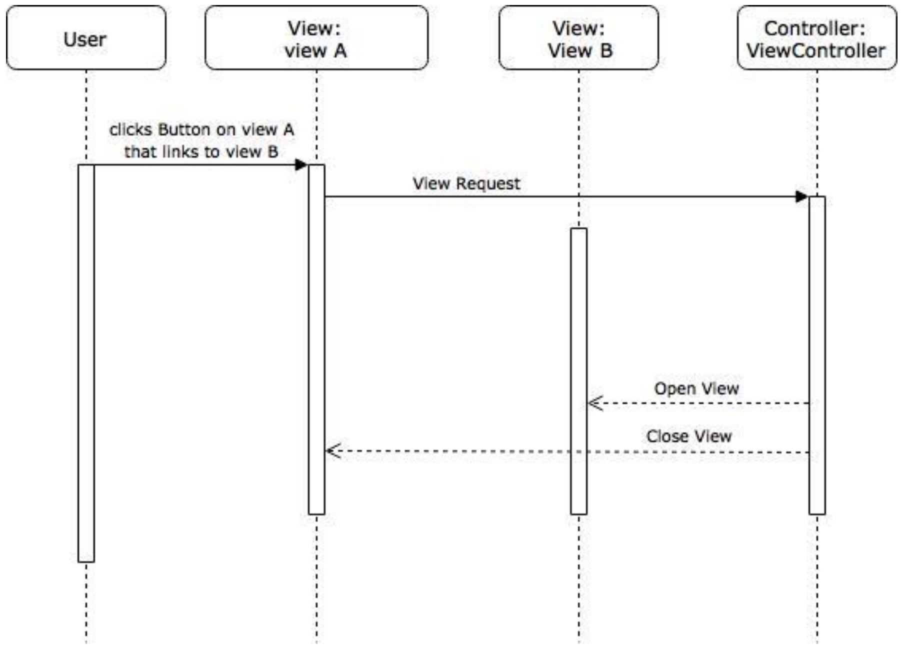
Model	**Controller**	**View**
Raffle	LoginController	HomeView
Ticket	SignUpController	AboutView
Seller	AccountInfoController	ContactView
Administrator	TicketSaleController	LoginView
DatabaseManager	SellerStatisticController	SignUpView
	RaffleStatisticController	SellerStatisticView
	SellerInfoController	RequestView
	ActivateSellerController	TicketSaleView
		TicketInfoView
		SellerInfoView
		ActivateSellerView
		AllocateTicketView

3.2 Dynamic Model – Sequence Diagrams

The following section describes how the user will interact with program objects using sequence diagrams. Each sequence diagram shows the sequence of interactions that take place during a particular use case; the use cases of these sequence diagrams refer to those use cases noted in the requirement specification deliverable. They are broken down into four categories: generalized, generic user, seller and administrator.

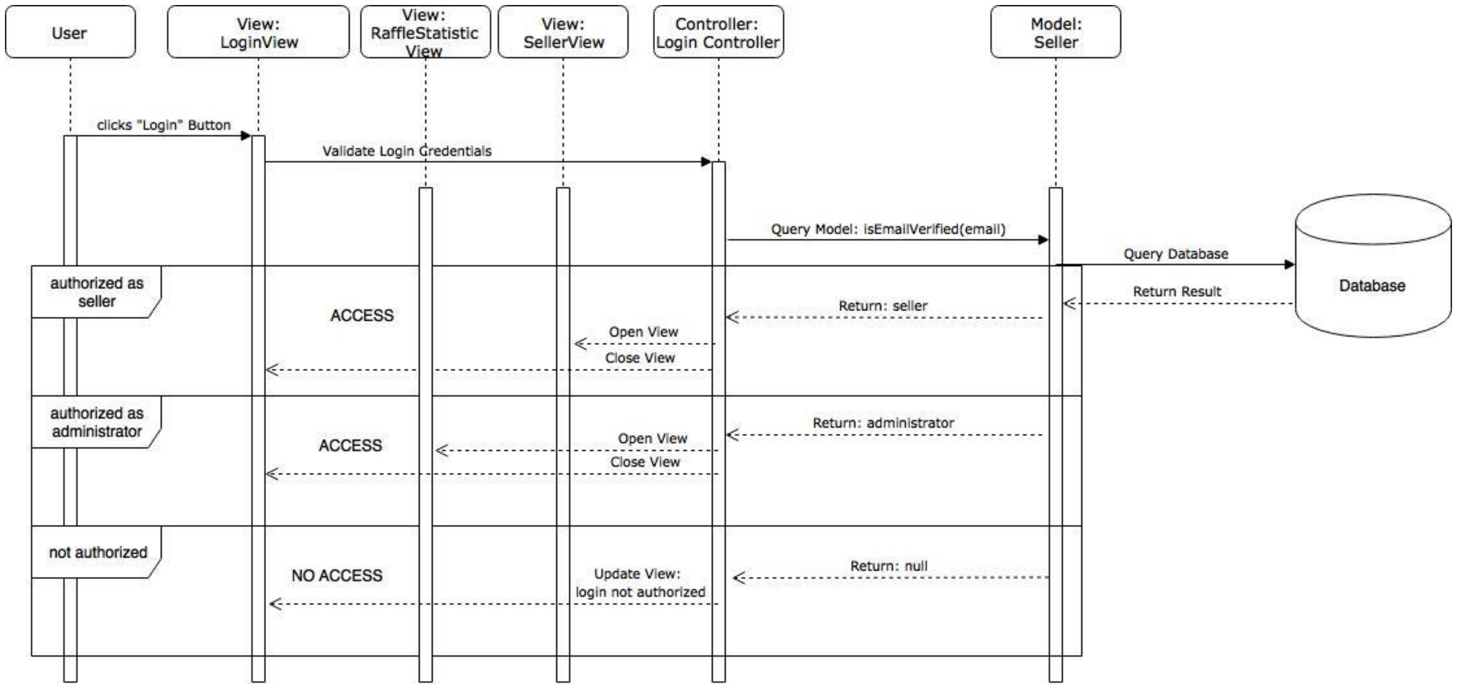
Generalized Sequence Diagrams

As seen in section 4.3 of this report, there are many view objects and actions. To draw sequence diagrams for each transition between views would be repetitive and time consuming. To be more efficient, a generic transition is defined from view ‘A’ to view ‘B’ to summarize each of these view transitions.


User
View: view A
View: View B
Controller: ViewController
clicks Button on view A that links to view B
View Request
Open View
Close View

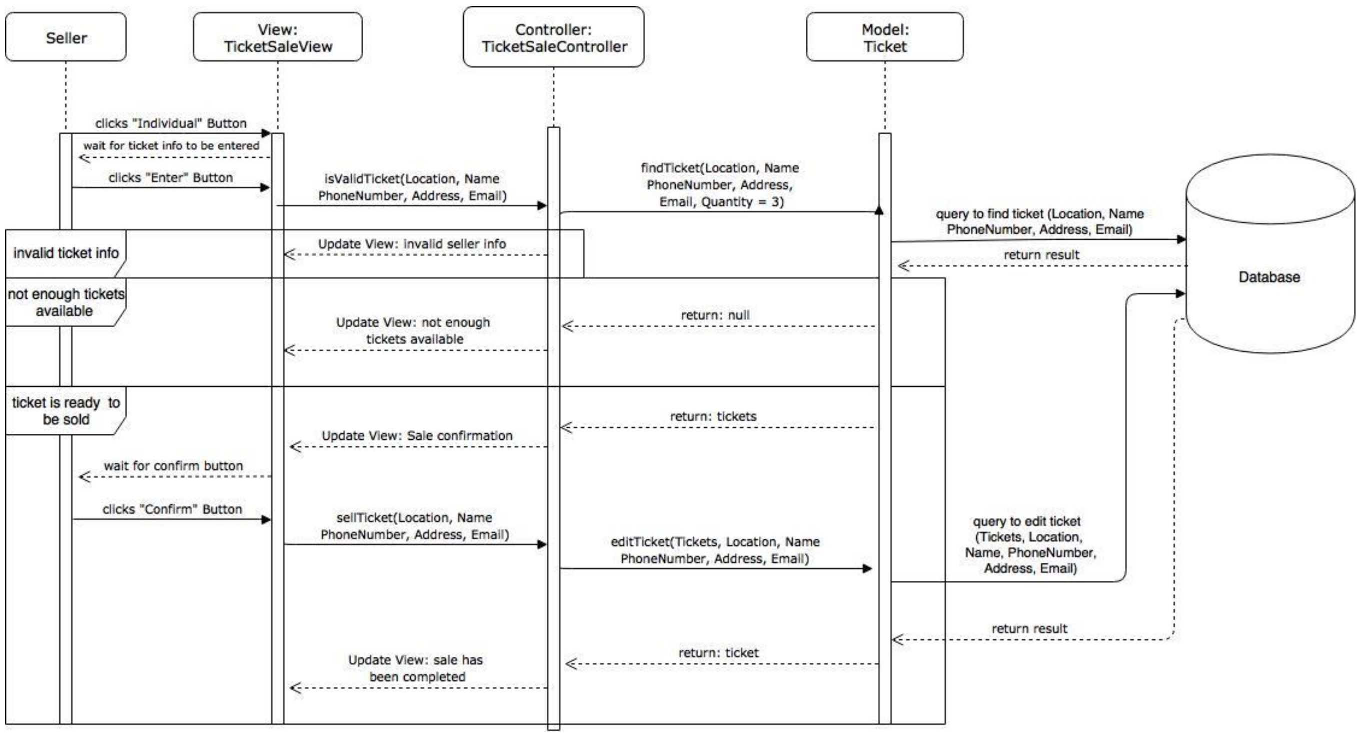
It can be seen that every view transition is processed through the view controller.

Generic User: Login

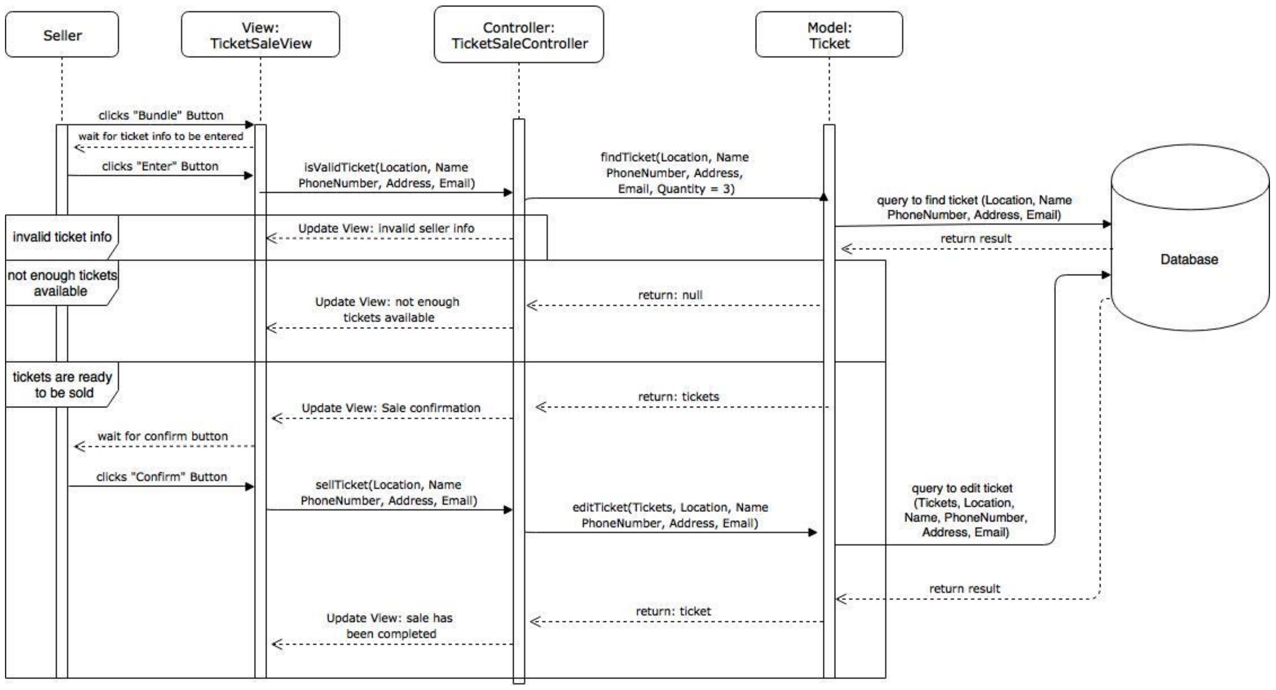


Seller

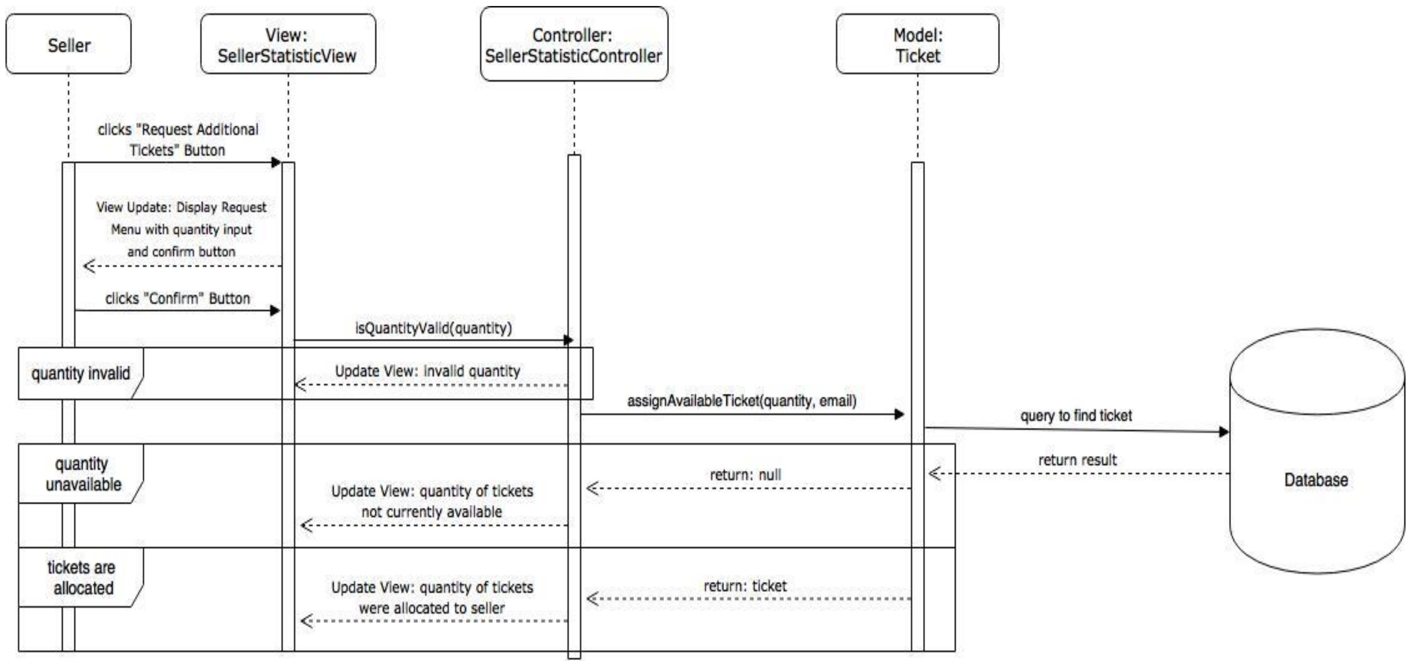
Seller: Sell Individual Ticket


Seller
View: TicketSaleView
Controller: TicketSaleController
Model: Ticket
clicks "Individual" Button
wait for ticket info to be entered
clicks "Enter" Button
isValidTicket(Location, Name PhoneNumber, Address, Email)
findTicket(Location, Name PhoneNumber, Address, Email, Quantity = 3)
query to find ticket (Location, Name PhoneNumber, Address, Email)
return result
Database
invalid ticket info
Update View: invalid seller info
not enough tickets available
return: null
Update View: not enough tickets available
ticket is ready to be sold
return: tickets
Update View: Sale confirmation
wait for confirm button
clicks "Confirm" Button
sellTicket(Location, Name PhoneNumber, Address, Email)
editTicket(Tickets, Location, Name PhoneNumber, Address, Email)
query to edit ticket (Tickets, Location, Name, PhoneNumber, Address, Email)
return result
return: ticket
Update View: sale has been completed

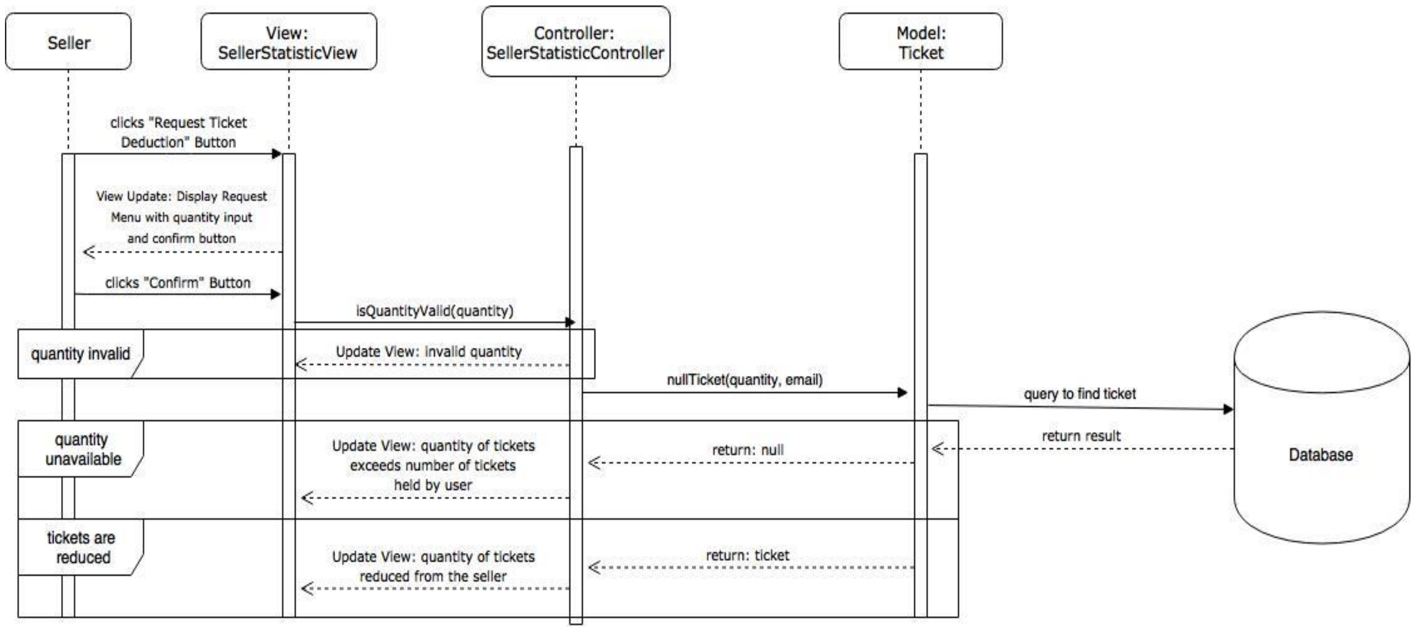
Seller: Sell Book of Tickets


Seller
View: TicketSaleView
Controller: TicketSaleController
Model: Ticket
clicks "Bundle" Button
wait for ticket info to be entered
clicks "Enter" Button
isValidTicket(Location, Name PhoneNumber, Address, Email)
findTicket(Location, Name PhoneNumber, Address, Email, Quantity = 3)
query to find ticket (Location, Name PhoneNumber, Address, Email)
return result
Database
invalid ticket info
Update View: invalid seller info
not enough tickets available
return: null
Update View: not enough tickets available
tickets are ready to be sold
return: tickets
Update View: Sale confirmation
wait for confirm button
clicks "Confirm" Button
sellTicket(Location, Name PhoneNumber, Address, Email)
editTicket(Tickets, Location, Name PhoneNumber, Address, Email)
query to edit ticket (Tickets, Location, Name, PhoneNumber, Address, Email)
return result
return: ticket
Update View: sale has been completed

Seller: Request Tickets

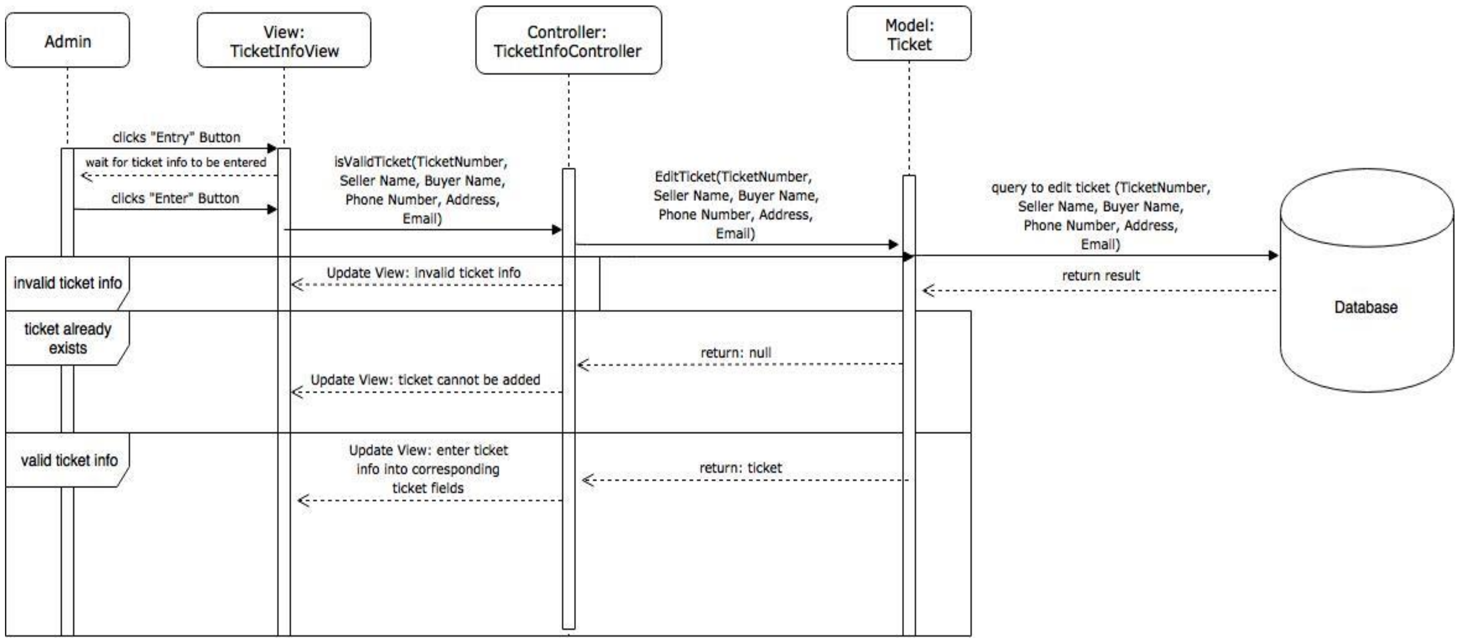

Seller
View: SellerStatisticView
Controller: SellerStatisticController
Model: Ticket
clicks "Request Additional Tickets" Button
View Update: Display Request Menu with quantity input and confirm button
clicks "Confirm" Button
isQuantityValid(quantity)
quantity invalid
Update View: invalid quantity
assignAvailableTicket(quantity, email)
query to find ticket
return result
Database
quantity unavailable
return: null
Update View: quantity of tickets not currently available
tickets are allocated
return: ticket
Update View: quantity of tickets were allocated to seller

Seller: Reduce Tickets


Seller
View: SellerStatisticView
Controller: SellerStatisticController
Model: Ticket
clicks "Request Ticket Deduction" Button
View Update: Display Request Menu with quantity input and confirm button
clicks "Confirm" Button
isQuantityValid(quantity)
quantity invalid
Update View: invalid quantity
nullTicket(quantity, email)
query to find ticket
return result
Database
quantity unavailable
Update View: quantity of tickets exceeds number of tickets held by user
return: null
tickets are reduced
return: ticket
Update View: quantity of tickets reduced from the seller

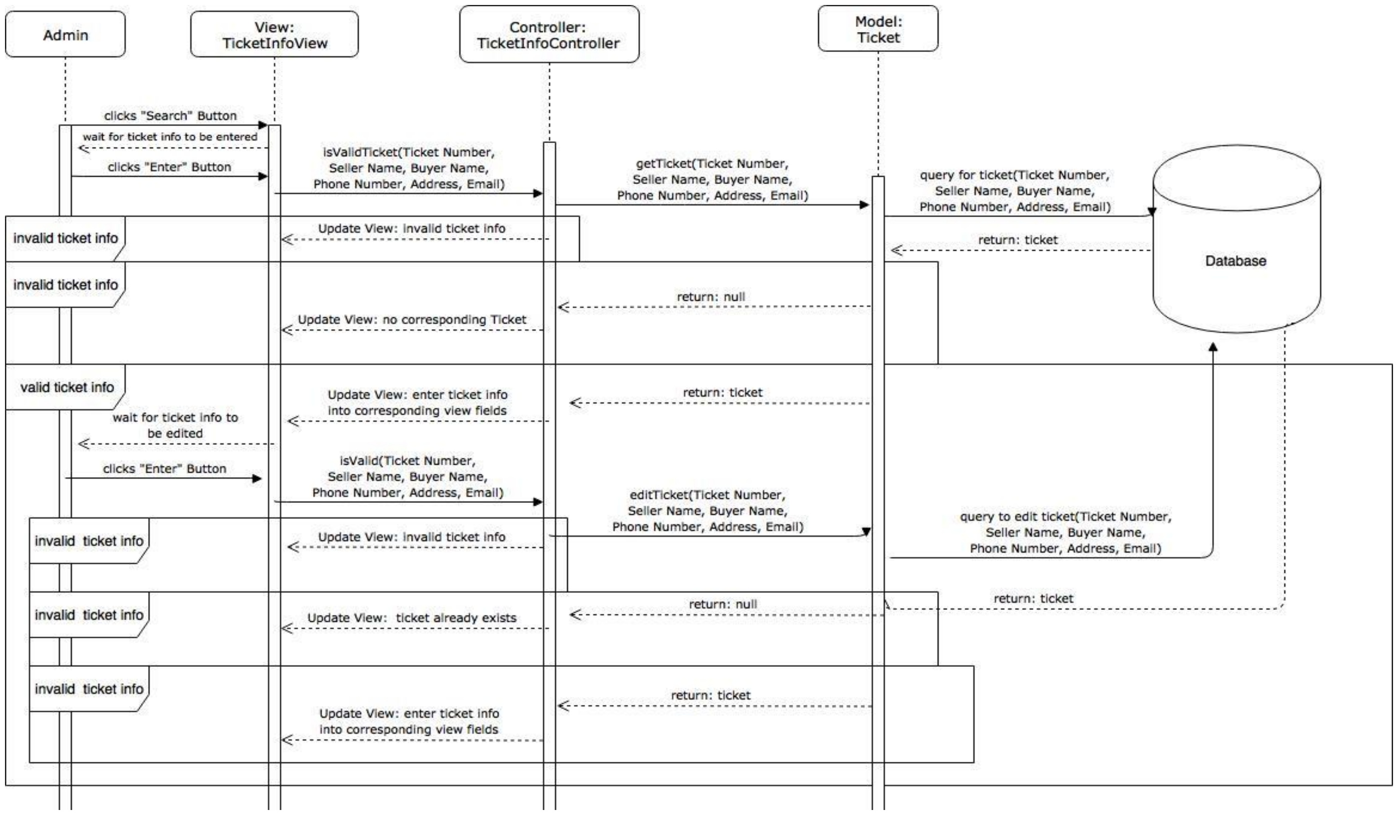
Administrator

Administrator: Enter Physical Ticket



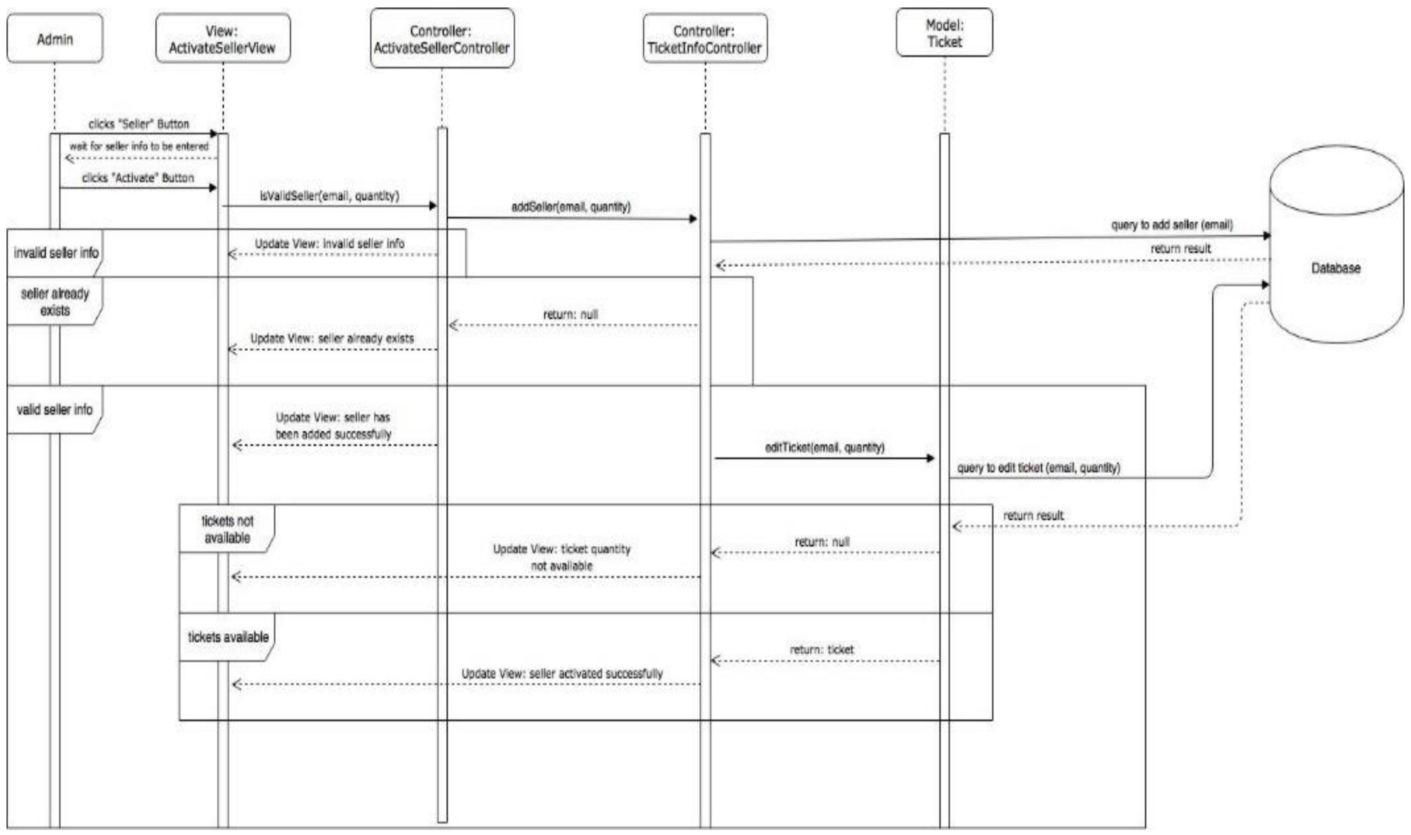
Admin
View: TicketInfoView
Controller: TicketInfoController
Model: Ticket
clicks "Entry" Button
wait for ticket info to be entered
clicks "Enter" Button
isValidTicket(TicketNumber, Seller Name, Buyer Name, Phone Number, Address, Email)
EditTicket(TicketNumber, Seller Name, Buyer Name, Phone Number, Address, Email)
query to edit ticket (TicketNumber, Seller Name, Buyer Name, Phone Number, Address, Email)
Database
invalid ticket info
Update View: invalid ticket info
return result
ticket already exists
return: null
Update View: ticket cannot be added
valid ticket info
Update View: enter ticket info into corresponding ticket fields
return: ticket

Administrator: Edit Ticket

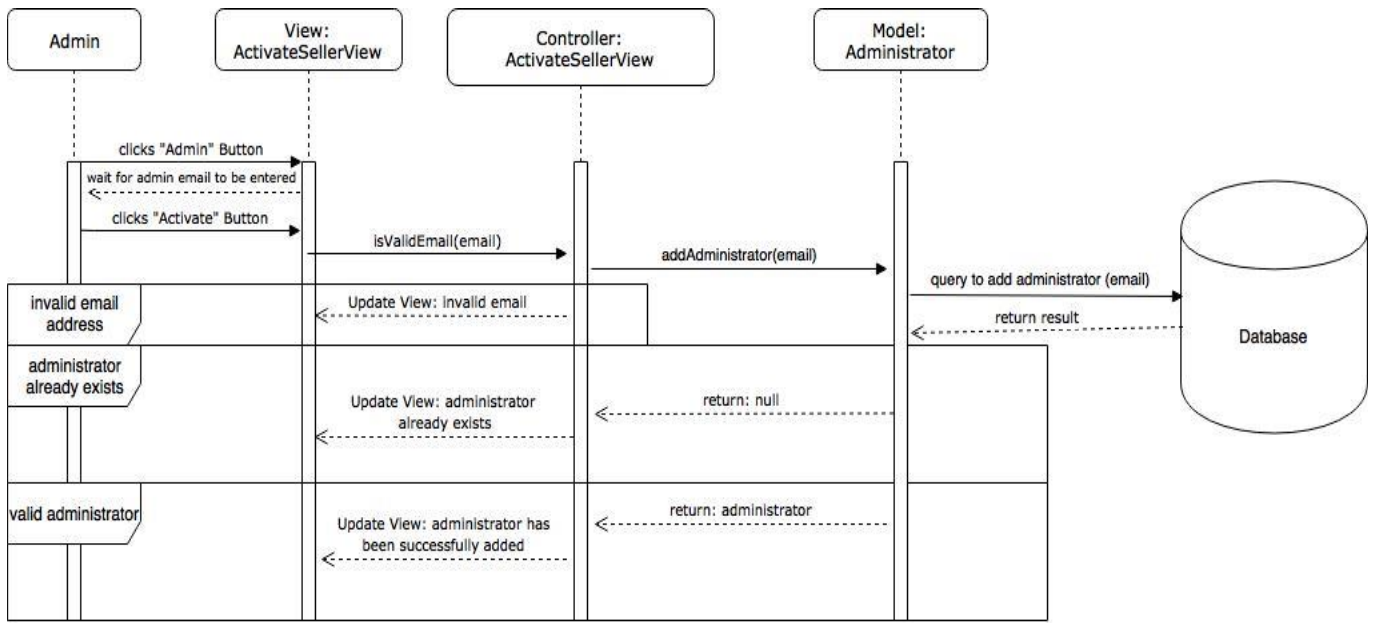


Admin
View: TicketInfoView
Controller: TicketInfoController
Model: Ticket
clicks "Search" Button
wait for ticket info to be entered
clicks "Enter" Button
isValidTicket(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
getTicket(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
query for ticket(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
Database
invalid ticket info
Update View: invalid ticket info
return: ticket
invalid ticket info
return: null
Update View: no corresponding Ticket
valid ticket info
Update View: enter ticket info into corresponding view fields
return: ticket
wait for ticket info to be edited
clicks "Enter" Button
isValid(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
editTicket(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
query to edit ticket(Ticket Number, Seller Name, Buyer Name, Phone Number, Address, Email)
invalid ticket info
Update View: invalid ticket info
invalid ticket info
return: null
return: ticket
Update View: ticket already exists
invalid ticket info
return: ticket
Update View: enter ticket info into corresponding view fields

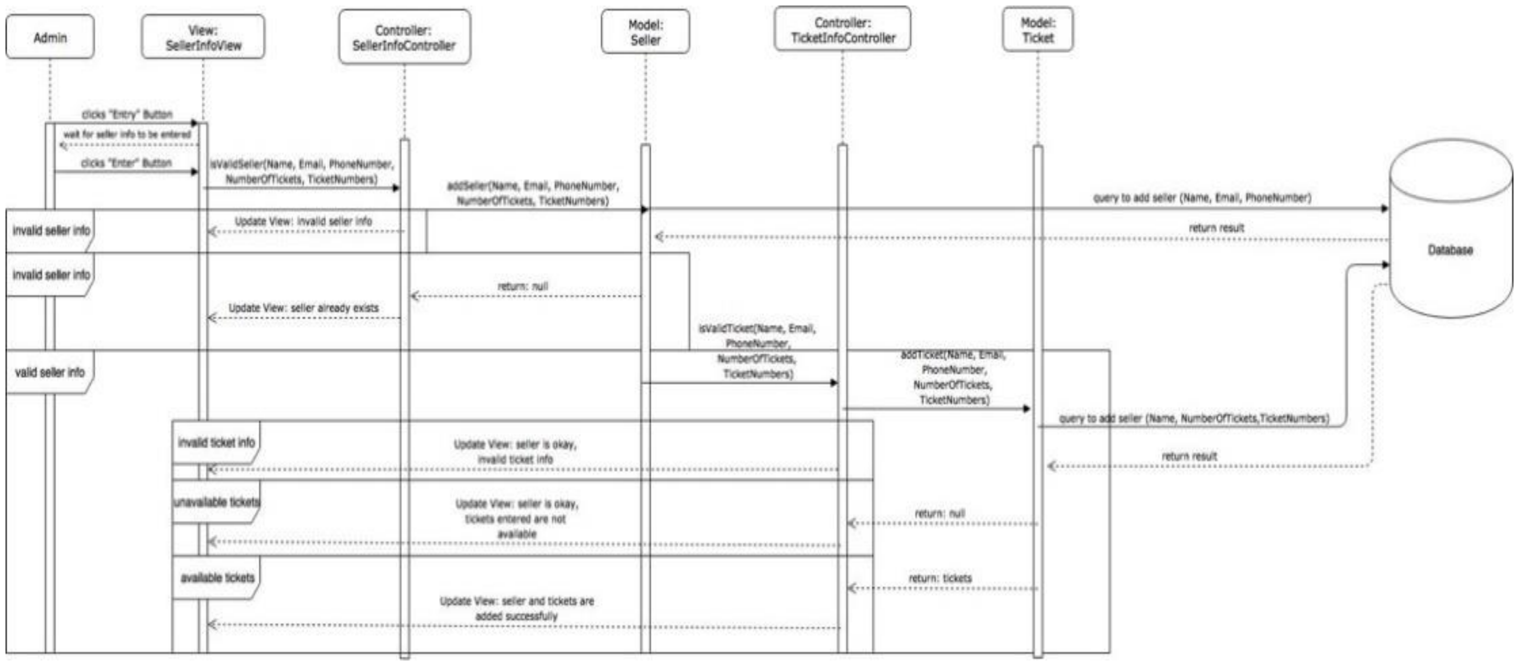
Administrator: Activate Seller


Admin
View: ActivateSellerView
Controller: ActivateSellerController
Controller: TicketInfoController
Model: Ticket
clicks "Seller" Button
wait for seller info to be entered
clicks "Activate" Button
isValidSeller(email, quantity)
addSeller(email, quantity)
query to add seller (email)
return result
Database
invalid seller info
Update View: invalid seller info
seller already exists
return: null
Update View: seller already exists
valid seller info
Update View: seller has been added successfully
editTicket(email, quantity)
query to edit ticket (email, quantity)
return result
tickets not available
return: null
Update View: ticket quantity not available
tickets available
return: ticket
Update View: seller activated successfully

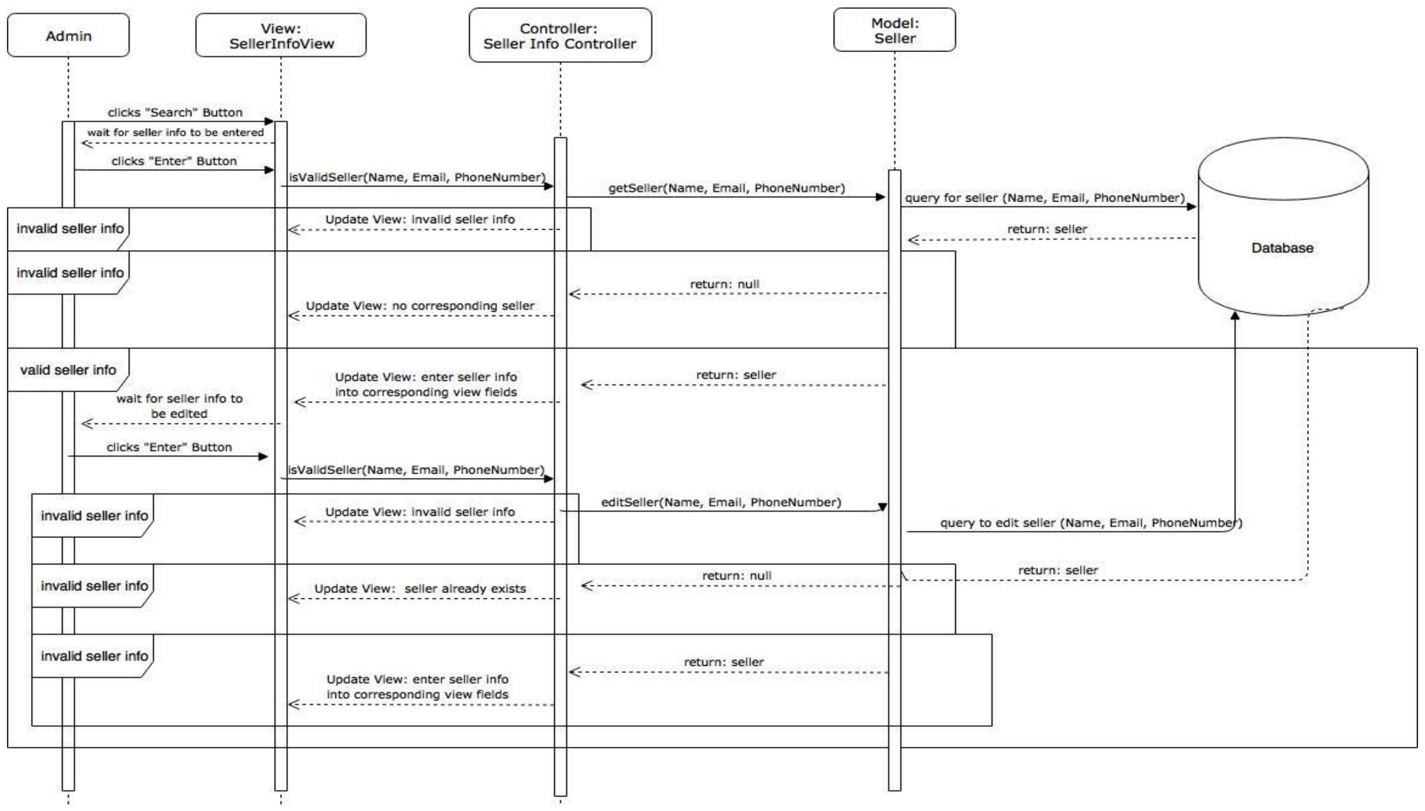
Administrator: Activate Administrator


Admin
View: ActivateSellerView
Controller: ActivateSellerView
Model: Administrator
clicks "Admin" Button
wait for admin email to be entered
clicks "Activate" Button
isValidEmail(email)
addAdministrator(email)
query to add administrator (email)
return result
Database
invalid email address
Update View: invalid email
administrator already exists
Update View: administrator already exists
return: null
valid administrator
Update View: administrator has been successfully added
return: administrator

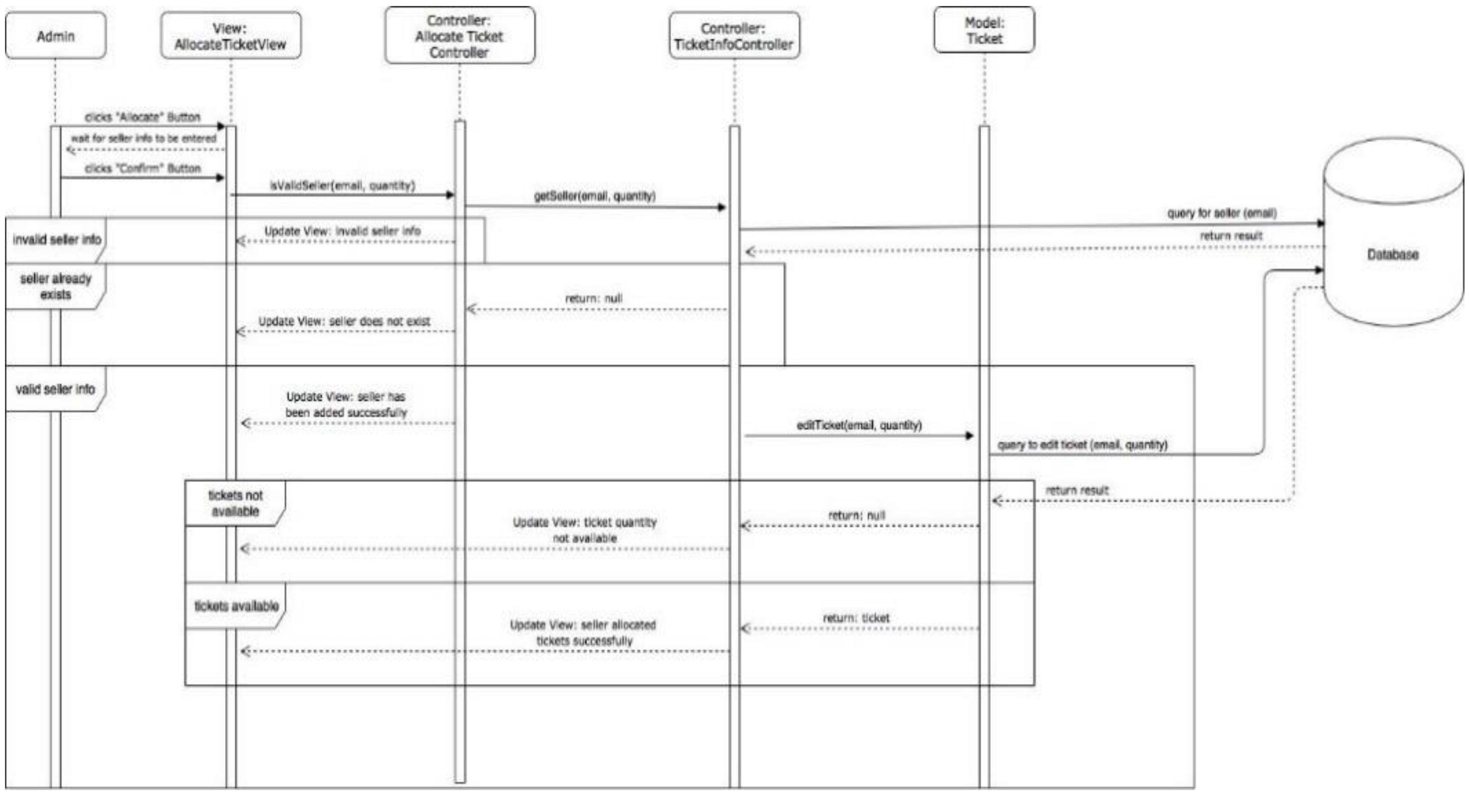
Administrator: Enter Seller


Admin
View: SellerInfoView
Controller: SellerInfoController
Model: Seller
Controller: TicketInfoController
Model: Ticket
Database
clicks "Entry" Button
wait for seller info to be entered
clicks "Enter" Button
isValidSeller(Name, Email, PhoneNumber, NumberOfTickets, TicketNumbers)
addSeller(Name, Email, PhoneNumber, NumberOfTickets, TicketNumbers)
query to add seller (Name, Email, PhoneNumber)
return result
invalid seller info
Update View: invalid seller info
invalid seller info
return: null
Update View: seller already exists
valid seller info
isValidTicket(Name, Email, PhoneNumber, NumberOfTickets, TicketNumbers)
addTicket(Name, Email, PhoneNumber, NumberOfTickets, TicketNumbers)
query to add seller (Name, NumberOfTickets,TicketNumbers)
return result
invalid ticket info
Update View: seller is okay, invalid ticket info
unavailable tickets
return: null
Update View: seller is okay, tickets entered are not available
available tickets
return: tickets
Update View: seller and tickets are added successfully

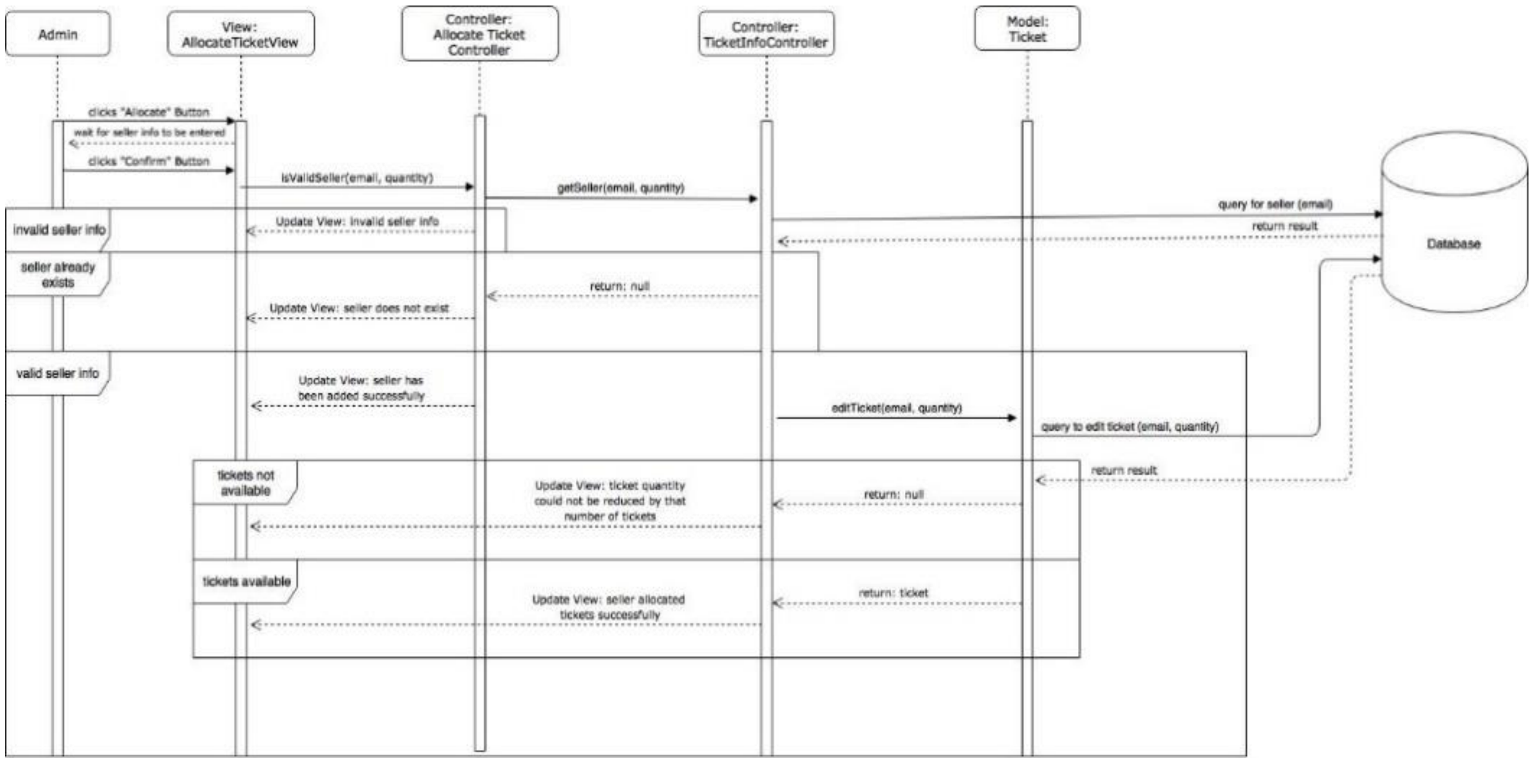
Administrator: Edit Seller


Admin
View: SellerInfoView
Controller: Seller Info Controller
Model: Seller
Database
clicks "Search" Button
wait for seller info to be entered
clicks "Enter" Button
isValidSeller(Name, Email, PhoneNumber)
getSeller(Name, Email, PhoneNumber)
query for seller (Name, Email, PhoneNumber)
return: seller
invalid seller info
Update View: invalid seller info
invalid seller info
return: null
Update View: no corresponding seller
valid seller info
return: seller
Update View: enter seller info into corresponding view fields
wait for seller info to be edited
clicks "Enter" Button
isValidSeller(Name, Email, PhoneNumber)
invalid seller info
Update View: invalid seller info
editSeller(Name, Email, PhoneNumber)
query to edit seller (Name, Email, PhoneNumber)
return: seller
invalid seller info
return: null
Update View: seller already exists
invalid seller info
return: seller
Update View: enter seller info into corresponding view fields

Administrator: Allocate Tickets


Admin
View: AllocateTicketView
Controller: Allocate Ticket Controller
Controller: TicketInfoController
Model: Ticket
Database
clicks "Allocate" Button
wait for seller info to be entered
clicks "Confirm" Button
isValidSeller(email, quantity)
getSeller(email, quantity)
query for seller (email)
return result
invalid seller info
Update View: invalid seller info
seller already exists
return: null
Update View: seller does not exist
valid seller info
Update View: seller has been added successfully
editTicket(email, quantity)
query to edit ticket (email, quantity)
return result
tickets not available
return: null
Update View: ticket quantity not available
tickets available
return: ticket
Update View: seller allocated tickets successfully

Administrator: Reduce Tickets


Admin
View: AllocateTicketView
Controller: Allocate Ticket Controller
Controller: TicketInfoController
Model: Ticket
Database
clicks "Allocate" Button
wait for seller info to be entered
clicks "Confirm" Button
isValidSeller(email, quantity)
getSeller(email, quantity)
query for seller (email)
return result
invalid seller info
Update View: invalid seller info
seller already exists
return: null
Update View: seller does not exist
valid seller info
Update View: seller has been added successfully
editTicket(email, quantity)
query to edit ticket (email, quantity)
return result
tickets not available
return: null
Update View: ticket quantity could not be reduced by that number of tickets
tickets available
return: ticket
Update View: seller allocated tickets successfully

3.3 Structural Model

All classes that fall within the Model class type have setter and getter methods for all of their attributes.

Class:	**Ticket**
Class Type:	Model
Attributes:	-ticketNumber : int
	-name : string
	-location : string
	-phoneNumber : string
	-email : string

Class:	**Seller**
Class Type:	Model
Attributes:	-name : string
	-location : string
	-phoneNumber : string
	-email : string
	-password : string
	-numberTicketsAllocated : int

Class:	**Administrator** <<extends>> Seller
Class Type:	Model
Attributes:	-name : string
	-location : string
	-phoneNumber : string
	-email : string
	-password : string
	-numberTicketsAllocated : int

Class:	**Raffle**
Class Type:	Model
Attributes:	-name : string

	-startDate : Date
	-endDate : Date
	-maxTickets : int
	-raffleTickets : List<Ticket>
	-raffleAdmins : List<Administrator>
	-raffleSellers : List<Seller>

For the following, an object of type T can be any one of Ticket, Seller, or Administrator.

Class:	**DatabaseManager**
Class Type:	Model
Attributes:	-database: DatabaseObject
Methods:	+insert(object : T) : bool
	Inserts the given object into its corresponding table in the database. If successful returns true, else returns false.
	+update(object : T) : bool
	Updates the given object in its corresponding table in the database. If successful returns true, else returns false.
	+exists(object : T) : bool
	Checks if the given object exists in its associated table. If successful returns true, else returns false.
	+remove(object : T) : bool
	Removes the given object from its corresponding table in the database. If successful returns true, else returns false.
	+query(searchQuery : string) : QueryResult
	Performs a query on the database using the specified searchQuery, and returns the result in the form of a QueryResult.

Class:	**LoginController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-seller : Seller

	-loginView : LoginView
Methods:	+login() : bool
	Attempts to log in the seller using information provided in the class's Seller object. If successful returns true, else returns false.

Class:	**SignUpController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-seller : Seller
	-signUpView : SignUpView
Methods:	+signUp() : bool
	Attempts to sign up the seller using information provided in the class's Seller object. If successful returns true, else returns false.

Class:	**AccountInfoController**
Class Type:	Controller
Attributes:	-seller : Seller
	-accountInfoView : AccountInfoView
Methods:	+display()
	Displays the account information for the given Seller in the AccountInfoView

Class:	**TicketSaleController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-ticket : Ticket
	-seller : Seller
	-ticketSaleView : TicketSaleView
Methods:	+sell() : bool
	Takes the information from the input fields of ticketSaleView and assigns it to ticket. The databaseManager then attempts to insert the ticket into the database. If successful returns true, else returns false.

Class:	**SellerStatisticController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-seller : Seller
	-sellerStatisticView : SellerStatisticView
Methods:	+obtainStats()
	Obtains the raffle information for the associated seller and displays it in sellerStatisticView.

Class:	**TicketSaleController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-ticket : Ticket
	-seller : Seller
	-ticketSaleView : TicketSaleView
Methods:	+sell() : bool
	Takes the information from the input fields of ticketSaleView and assigns it to ticket. The databaseManager then attempts to insert the ticket into the database. If successful returns true, else returns false.

Class:	**SellerStatisticController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-seller : Seller
	-sellerStatisticView : SellerStatisticView
Methods:	+obtainStats()
	Obtains the raffle statistics associated with the seller, and displays them in sellerStatisticView.

Class:	**RaffleStatisticController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager

	-administrator : Administrator
	-raffle : Raffle
	-raffleStatisticView : RaffleStatisticView
Methods:	+obtainStats()
	Obtains the raffle statistics associated with the Raffle, and displays them in raffleStatisticView.

Class:	**SellerInfoController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-sellerList : List<Seller>
	-administrator : Administrator
	-sellerInfoView : SellerInfoView
Methods:	+obtainSellers()
	Obtains all of the sellers associated with the raffle and displays them in sellerInfoView.

Class:	**ActivateSellerController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-administator : Administrator
	-seller : Seller
	-activateSellerView : ActivateSellerView
Methods:	+activate() : bool
	Administrator activates the account associated with the seller in the database given in the activateSellerView. If successful returns true, else returns false.

Class:	**AllocateTicketController**
Class Type:	Controller
Attributes:	-databaseManager : DatabaseManager
	-administator : Administrator
	-seller : Seller

	-allocateTicketView : ActivateSellerView
Methods:	+allocate(number : int) : bool
	Administrator allocates the given number of tickets to a seller. If successful returns true, else returns false.

Class:	**HomeView**
Class Type:	View
Attributes:	+contactUs: Button
	+about: Button
	+login: Button

Class:	**AboutView**
Class Type:	View
Attributes:	+home: Button

Class:	**ContactView**
Class Type:	View
Attributes:	+home: Button

Class:	**LoginView**
Class Type:	View
Attributes:	+username: Textfield
	+password: Textfield
	+signup: Button
	+home: Button
	+forgotPassword: Button

Class:	**SignUpView**
Class Type:	View
Attributes:	+username: Textfield
	+password: Textfield
	+home: Button
	+signIn: Button

Class:	**SellerStatisticView**
Class Type:	View
Attributes:	+home: Button
	+sell: Button
	+requestTickets: Button

	+reduceTickets: Button
Class:	**RequestView**
Class Type:	View
Attributes:	+quantity: Textfield +confirm: Button
Class:	**TicketSaleView**
Class Type:	View
Attributes:	+quantity: Textfield +BuyerInformation: Textfield +bookInd: Button +back: Button +error: Button
Class:	**SignUpView**
Class Type:	View
Attributes:	+sell: Button +home: Button +tickets: Button +entryRequest: Button +entrySeller: Button +activate: Button +allocate: Button
Class:	**TicketInfoView**
Class Type:	View
Attributes:	+ticketInfo: Textfield +entrySearch: Button +back: Button +enter: Button
Class:	**SellerInfoView**
Class Type:	View
Attributes:	+sellerInfo: Textfield +back: Button +enter: Button +entrySearch: Button
Class:	**ActivateSellerView**
Class Type:	View

Attributes:	+email: Textfield +quantity: Textfield +sellerAdmin: Button +activate: Button
Class:	**AllocateTicketView**
Class Type:	View
Attributes:	+seller: Textfield +quantity: Textfield +allocateReduce: Button +confirm: Button +back: Button

USER INTERFACE DESIGN

4.1 Overview of the User Interface

The web-based application is designed with four possible users in mind, the ticket seller (Seller), the fundraiser administrator (Admin) and the general public or buyer (referred to collectively as Public in this section because they interact with the user interface in the same manner). For the convenience of the development team these pages have been temporarily colour coded. Red for Public, green Seller, and blue Admin. Figure 4.1 depicts the main webpage (Home) that is accessible to all, and contains a description of the 12 Draws of Christmas, a prize list and a countdown to the first draw. The material on this page will be customized to meet the advertising needs of the client over time. Its main function is to serve as an information source for the Public. This page contains links to three other pages through the "About", "Contact Us" and "Login" buttons and these pages are depicted in Figures 4.2, 4.3, and 4.4 respectively.

Figure 4.2 and 4.3 show simple webpages that provide additional information to the general public about the draw and the sponsoring organization(s). These pages serve as sources of information only, and therefore will host material provided to the development team by the client. This information includes details about the client's organization (Figure 4.2) and contact information for the fundraisers main administrators (Figure 4.3). From the Home webpage, a user can navigate to the login page via the "Login" button, this page is shown in Figure 4.4. This page is where a Admin or Seller can login to their account to view details about their accounts and the fundraiser by entering their email and password, then clicking the "Login" button. If someone attempts to login with an unknown email and password combination the page will show a brief animation communicating the inability to login (not shown as animations cannot be shown in a "screen shot"). If a Seller or Admin is using their account for the first time they can click the "Sign Up" link and be directed to the Sign Up page. From the Login page a user or Admin can also navigate back to Home using the "Home" button or can select "Forgot Password". In order to select "Forgot Password", a recognized email address must be entered, if so, then a request is sent to the Admin page for account Reactivation (see Figures 4.11 and 4.12). If the email address is not recognized or not entered a brief animation will communicate the inability to send the reactivation request. Figure4. 5 shows the Sign Up page, this page allows a first time Seller or Admin to enter their information and then gain access to their account by clicking the "Sign Up" button. In order for anyone to have an account and Admin must have previously activated their email address through the Activate Page. This ensures that only people belonging to, or who are known by the organization may be granted an account. This page also allows a user to navigate back to Home via the "Home" button, or to go to the Login page if they already have an account. The data entered in this page automatically updates the database when a valid email is entered, data is entered and the "Sign Up" button is when a valid email is entered, data is entered and the "Sign Up" button is clicked.

If a Seller successfully logs into their account they are directed to their Seller Account page (Figure 4.6). This page shows a graphical representation of how many tickets they have sold out of their allotted amount. It also allows a Seller to navigate back to Home using the "Home" button, and lets them request more tickets, or to have their tickets reduced by clicking the "Request Tickets" and "Reduce Tickets" buttons respectively. Clicking the "Request Tickets" or "Reduce Tickets" brings up a corresponding popup as is shown in Figures 4.7 and 4.8. These forms all Sellers to enter how many tickets they would like to request to be allotted or to give back. Clicking the "Confirm" button sends an appropriate request to the Admin page (see Figure 4.11). Clicking the "Sell" button directs a seller to the Sell Page.

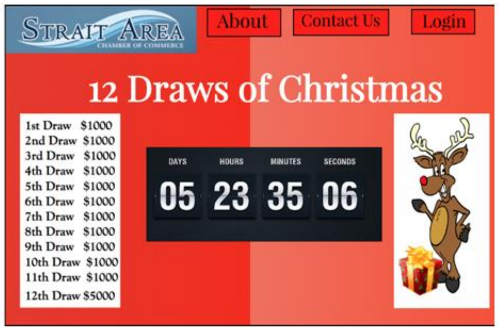
Figure 4.9 depicts the Sell page. This page is where a Seller can sell digital tickets to a buyer through a user-friendly form. This form allows the seller to select a book (group of 3 tickets) or individual tickets, and enter the quantity and other relevant data. When all the data is entered they will click the "Enter" button, which prompts a popup (Figure 4.10) which displays the information entered and a "Confirm" button. If the seller is happy with the displayed information they will click "Confirm" which automatically enters the ticket data into the database. If there was an error in a ticket entry a Seller can click the "Error" button and then re-enter the ticket data, fixing any mistakes. Clicking the error button sends a request to Admin with the relevant information (see Figure 4.11). Lastly this page has a "Back" button which returns a Seller to their Seller Account page.

When an Admin successfully logs into their account they are directed to their Admin Account page (Figure 4.11). This page shows a graphical representation of total ticket sales over time. It also shows a list of all the Sellers for this fundraiser and a list of received requests. If an Admin clicks on a seller name they are directed to the Seller Info page (Figure 4.14), if they click on a request they are taken to either the Seller Info Page, the Ticket Info Page (Figure 4.15), or the Activation page (Figure 4.12). They can also access the Ticket Info page by clicking the "Tickets" button and the Activation page by clicking the "Activate" button. An Admin may also sell Tickets by clicking the "Sell" button, this will direct them to the Seller Info page (Figure 4.7). From this page, an Admin can sell or request ticket number changes and see their seller statistics. An Admin may go back to the Home page (Figure 4.1) by clicking the "Home" button.

An Admin can activate seller accounts by clicking the "Activate" button, which will take them to the Activation page (Figure 4.12). This page allows an Admin to activate a new account by choosing whether they are creating an Admin or Seller account, entering an email address, the number of tickets to allocate to that seller and clicking the "Activate" button. This will automatically update the database with this information. The Admin can go back to their account page by clicking the "Back" button. An Admin can allocate new tickets to a seller or reduce the number of tickets a seller is allocated through the Allocation page (Figure 4.13). This page can be reached via the "Allocate" button on the Admin Account page (Figure 4.11). On the Allocate page the Admin can choose if they wish to allocate or reduce the number of tickets a seller has and enter the quantity they wish to add or take away. They can then press the "Confirm" button which updates the database. The Admin can go back to their Admin Account page by clicking the "Back" Button.

Figure 4.14 shows the Seller Info page, which can be used to view and update seller information. If any of the field in the Seller Info page are filled in suggestions of who the Admin may be searching for appears. If a suggestion is selected all the information for that Seller appears in the fields. This information can then be changed by editing the information then clicking the "Update" button. The Admin may return to the Admin Account Page (Figure 4.9) by clicking the "Back" button. Figure 4.11 shows the Ticket Info page, which can be used to view, edit and enter ticket information. As any of the fields are filled in ticket suggestions are displayed. If one of these suggestions are clicked all corresponding information appears in the fields. This information can be edited when the “Search” button is selected, the new data is entered and the “Enter” button is pressed, thus updating the database. Selecting the “Entry” button allows an Admin to enter new ticket data from physical tickets that have been sold. In that case the Admin can enter ticket data and click the “Enter” button to add this data to the database. The Admin can go back to the Admin Account page (Figure 4.9) by clicking the “Back” button.

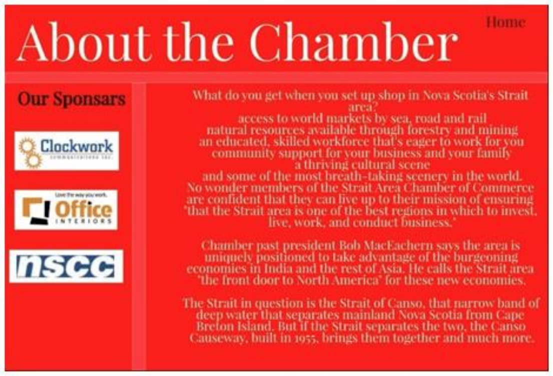
4.2 Screen Images



12 Draws of Christmas

1st Draw	$1000
2nd Draw	$1000
3rd Draw	$1000
4th Draw	$1000
5th Draw	$1000
6th Draw	$1000
7th Draw	$1000
8th Draw	$1000
9th Draw	$1000
10th Draw	$1000
11th Draw	$1000
12th Draw	$5000

Figure 4.1 Home Page



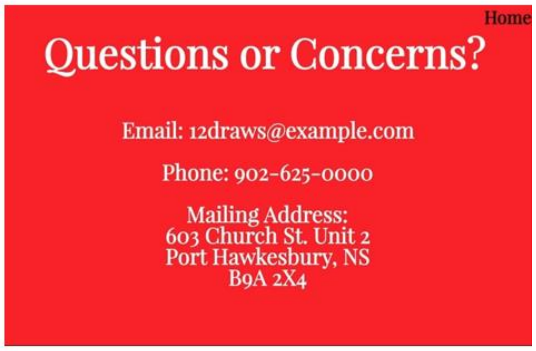
About the Chamber

What do you get when you set up shop in Nova Scotia's Strait area?
access to world markets by sea, road and rail
natural resources available through forestry and mining
an educated, skilled workforce that's eager to work for you
community support for your business and your family
a thriving cultural scene
and some of the most breath-taking scenery in the world.
No wonder members of the Strait Area Chamber of Commerce are confident that they can live up to their mission of ensuring "that the Strait area is one of the best regions in which to invest, live, work, and conduct business."

Chamber past president Bob MacEachern says the area is uniquely positioned to take advantage of the burgeoning economies in India and the rest of Asia. He calls the Strait area "the front door to North America" for these new economies.

The Strait in question is the Strait of Canso, that narrow band of deep water that separates mainland Nova Scotia from Cape Breton Island. But if the Strait separates the two, the Canso Causeway, built in 1955, brings them together and much more.

Figure 4.2 About Page



Questions or Concerns?

Email: 12draws@example.com

Phone: 902-625-0000

Mailing Address:
603 Church St. Unit 2
Port Hawkesbury, NS
B9A 2X4

Figure 4.3 Contact Page

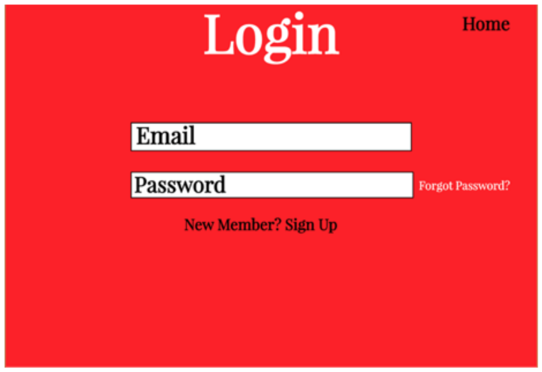


Figure 4.4 Login Page

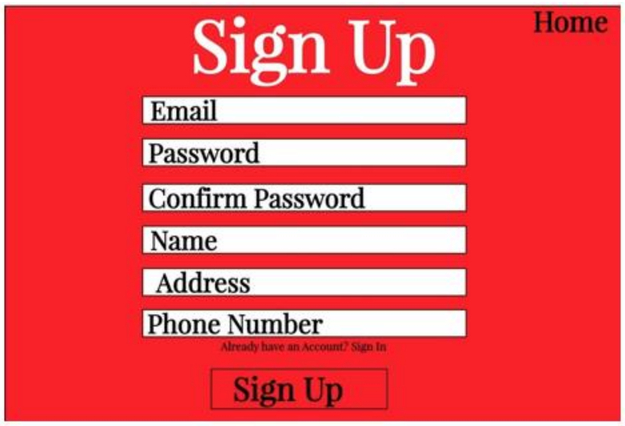


Figure 4.5 Sign Up Page

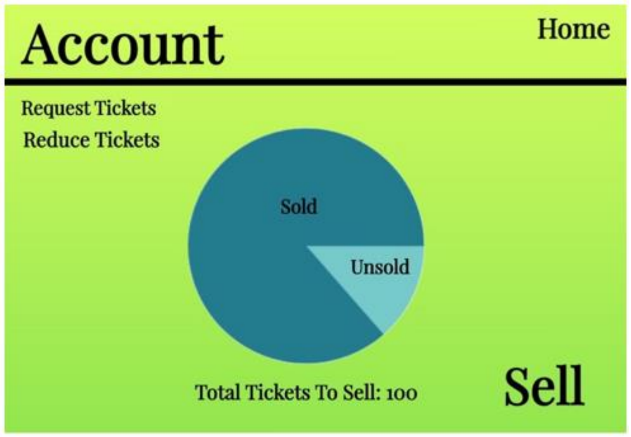


Figure 4.6 Seller Account Page

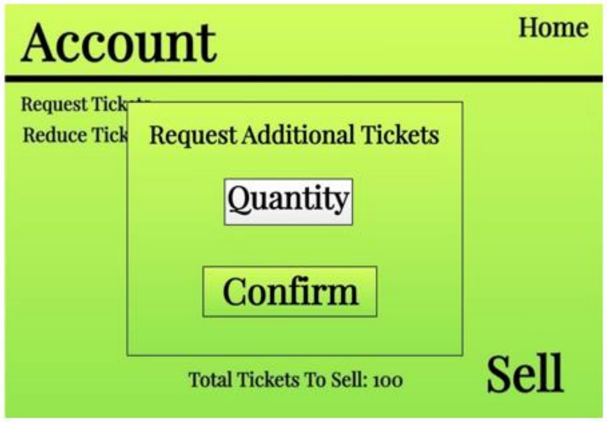


Figure 4.7 Request Additional Tickets Popup

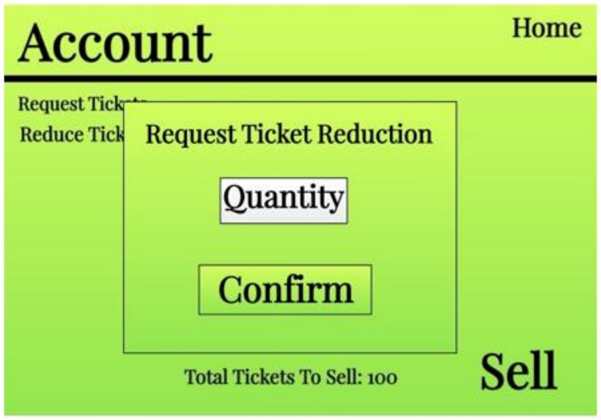


Figure 4.8 Request Reduction of Tickets Popup

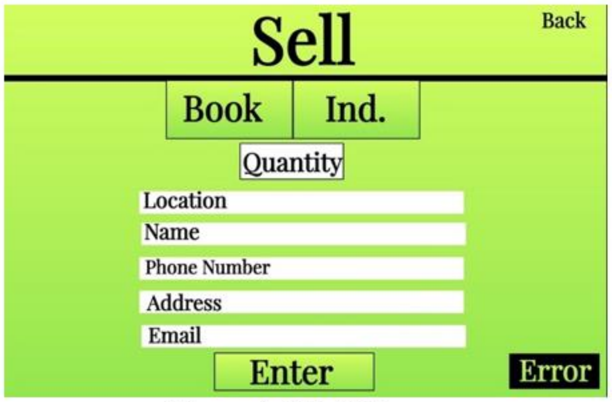


Figure 4.9 Sell Page

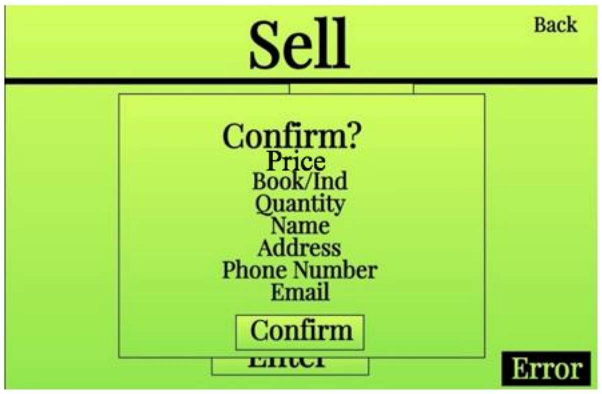


Figure 4.10 Sell Page with Popup

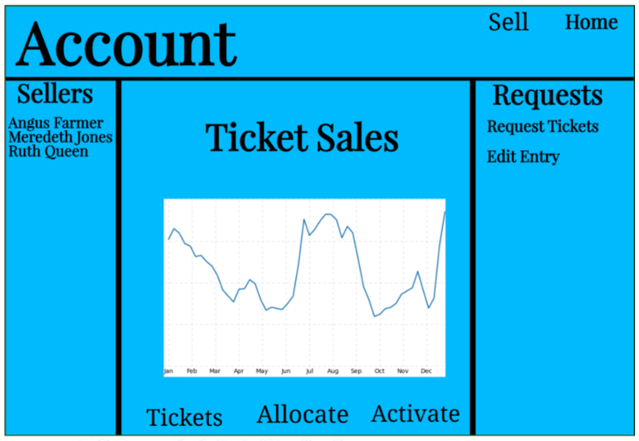


Figure 4.11 Admin Account Page

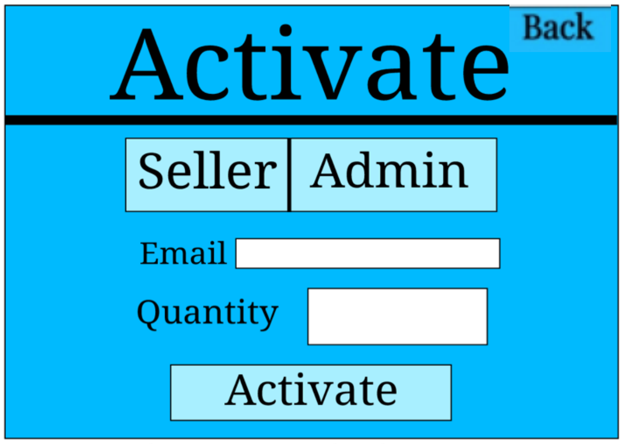


Figure 4.12 Activation Page

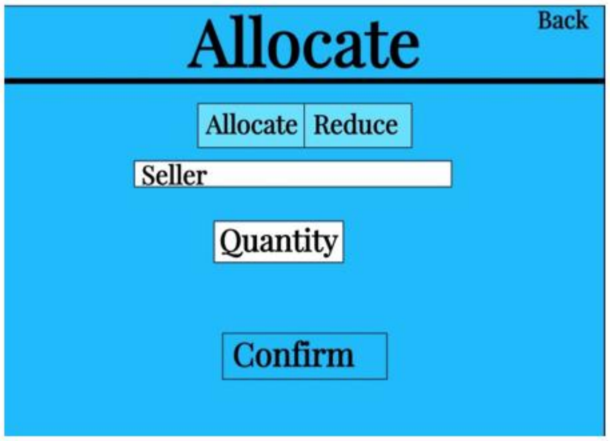


Figure 4.13 Allocation Page

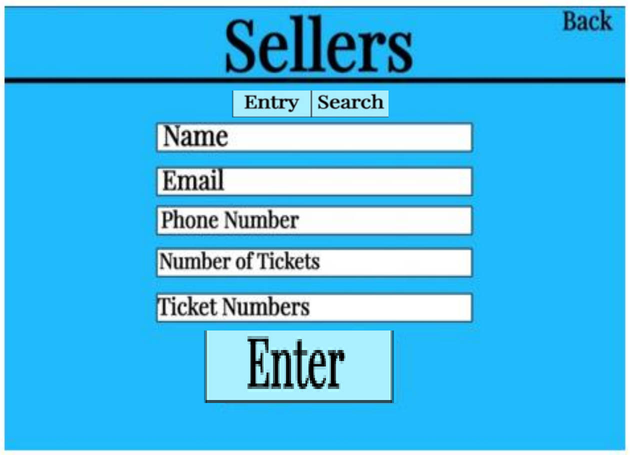


Figure 4.14 Seller Info Page

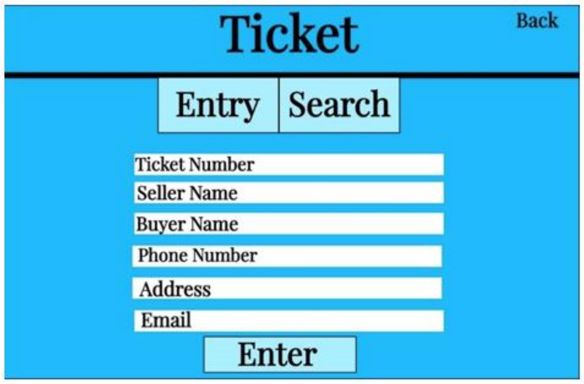
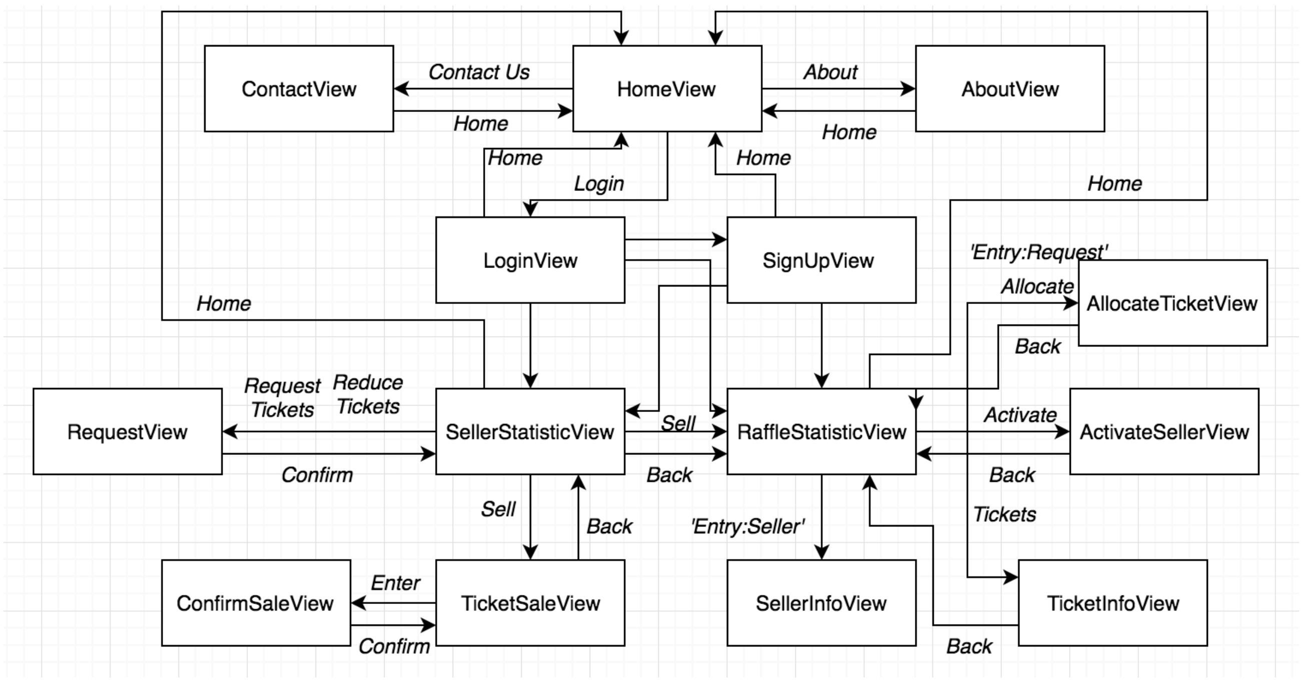


Figure 4.15 Ticket Info Page

4.3 Screen Objects and Actions



*For simplicity, the descriptions of the Views are omitted in this section and can instead be found in Section 3.

** For clarity, each screen object will be referred to by the corresponding View name

HomeView

Contact Us

Users are taken to the ContactView by clicking the 'Contact Us' button.

About

Users are taken to the AboutView by clicking the 'About' button

Login

Users are taken to the LoginView by clicking the 'Login' button

AboutView

Home

Users are taken to the HomeView by clicking the 'Home' button.

ContactView

Home

Users are taken to the HomeView by clicking the 'Home' button.

LoginView

Username

A string of characters corresponding to the email of an activated account.

Password

A string of characters corresponding to the username of the entered, activated account.

Forgot Password

A request for a password change is sent to the Admin account(s).

‘Enter’

If a correct username and password pair are entered the user is taken to either the SellerStatisticView or the RaffleStatisticView depending on the account type associated with the entered username.

Sign Up

Users are taken to the SignUpView by clicking the ‘Sign’ button

Home

Users are taken to the HomeView by clicking the ‘Home’ button.

SignUpView

Username

A string of characters corresponding to the email of an activated account.

Password

A string of characters which will be set to correspond to the username of the entered, activated account.

Home

Users are taken to the HomeView by clicking the ‘Home’ button.

Sign In

Users are taken to the LoginView by clicking the ‘Sign In’ button.

SellerStatisticView

Home

Users are taken to the HomeView by clicking the ‘Home’ button.

Sell

Users are taken to the TicketSaleView by clicking the ‘Sell’ button.

Request Tickets

Users are taken to the RequestView by clicking the ‘RequestTickets’ button.

Reduce Tickets

Users are taken to the RequestView by clicking the ‘ReduceTickets’ button.

RequestView

Quantity

A numerical value corresponding to the amount of tickets requested to be added/removed.

Confirm

Users are taken to the SellerStatisticView by clicking the ‘Confirm’ button.

‘Click’

Users are taken to the SellerStatisticView by clicking away from the popup.

TicketSaleView

Book/Ind

Users may select if they are selling individual tickets or a book by clicking the corresponding side of the ‘Book/Ind’ button.

Back

Users are taken to the SellerStatisticView by clicking the 'Back' button.

Error

Users may send a notification to the admin of a selling error by clicking the 'Error' button.

Quantity

A numerical value corresponding to the number of tickets/books to be sold.

BuyerInfomation

A series of characters corresponding to the buyer's personal information.

RaffleStatisticView

Sell

Users are taken to the SellerStatisticView by clicking the 'Sell' button.

Home

Users are taken to the HomeView by clicking the 'Home' button.

Tickets

Users are taken to the TicketInfoView by clicking the 'Ticket' button.

'Entry: Request'

Users are taken to the AllocateTicketView by clicking the an 'Entry' in the Seller column.

'Entry: Seller'

Users are taken to the SellerInfoView by clicking an 'Entry' in the Request column.

Activate

Users are taken to the ActivateSellerView by clicking the 'Activate' button.

Allocate

Users are taken to the AllocateTicketView by clicking the 'Allocate' button.

TicketInfoView

Back

Users are taken to the RaffleStatisticView by clicking the 'Back' button.

Entry/Search

Users can select either side of the 'Entry/Search' button to select whether they are searching for ticket info or entering new ticket info.

Enter

Users execute either ticket data entry or search for ticket data depending on their Entry/Search selection.

Ticket Info

A series of characters corresponding to ticket information.

SellerInfoView

Back

Users are taken to the RaffleStatisticView by clicking the 'Back' button

Entry/Search

Users can select either side of the 'Entry/Search' button to select whether they are searching for selling info or entering new seller info.

Enter

Users execute either seller data entry or search for seller data depending on their Entry/Search selection.

Seller Info

A series of characters corresponding to seller information.

ActivateSellerView

Seller/Admin

Users can select either side of the ‘Seller/Admin’ button to select whether they are activating a new seller or admin account.

<u>Quantity</u>

A numerical value corresponding to how many tickets that new account will be initially allocated.

<u>Email</u>

A string of characters corresponding to the new account to be created.

<u>Activate</u>

Executes the account activation.

AllocateTicketView

<u>Back</u>

Users are taken to the RaffleStatisticView by clicking the ‘Back’ button

<u>Allocate/Reduce</u>

Users can select either side of the ‘Allocate/Reduce’ button to select whether they are allocating or reducing ticket amounts.

<u>Seller</u>

A string of characters corresponding to a seller account.

<u>Quantity</u>

A numerical value corresponding to the amount of tickets requested to be allocated/removed.

<u>Confirm</u>

Executes the allocation/reduction

4.4 Report Formats

Figure 4.16 is a visual representation of a report that will be displayed by the HomeView. This report is a countdown to the end of the Raffle and can will be viewed by anyone who visits the Raffle webpage. The countdown begins when the raffle is created and the value of the countdown is generated based on the start date and end date for the raffle set by the admin at the start of the raffle. Figure 4.17 is a report that is generated and displayed by the SellerStatisticView. It is a visual chart that provides information to the seller about how many tickets they have sold of their allocated amount. The data for this report is generated by a query which determines how many tickets have been allocated to the seller who logged in and how many tickets they have sold. Figure 4.18 and 4.19 are reports which are displayed by the RaffleStatisticView which shows an Admin how many tickets have been sold by all the sellers either by day or by week. RaffleStatisticView will allow an admin to switch between weekly and daily reports and to change to dates between which the values are displayed. The values are generated using a query which returns the values for ticket sell dates. Figure 4.20 is a report displayed by the TicketInfoView. It is displayed when an Admin selects the search option, enters in partial data and then clicks the ‘Enter’ button. The report will contain the data entered and will fill all the other fields (see Figure 4.20) with the corresponding data. The data is returned by a query which uses the entered information to find the relevant ticket. Figure 4.21 is displayed by the SellerInfoView and is displayed when an Admin selects the search option, enters in partial data and then clicks the ‘Enter’ button. The report will contain the data entered and will fill all the other fields (see Figure 4.21) with the corresponding data. The data is returned by a query which uses the entered information to find the relevant seller. Figure 4.22 is displayed by the AllocateTicketView when an Admin clicks on a request notification. A query returns the information for that notification and it is displayed in the corresponding fields (see Figure 2.22). Figure 4.23 is displayed by ConfirmSaleView and is displayed when a Seller or Admin clicks the ‘Enter’ button on the SellerStatisticView. The values in the report are generated from the values entered into the form by the Seller or Admin or are calculated based on the price for each ticket which is set during the creation of the Raffle by the Admin.

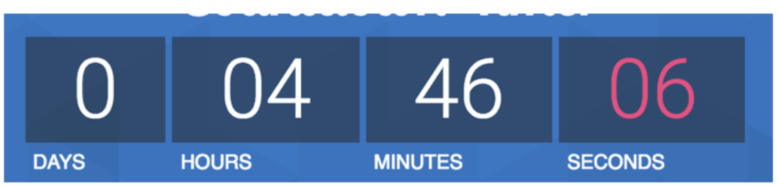


Figure 4.16

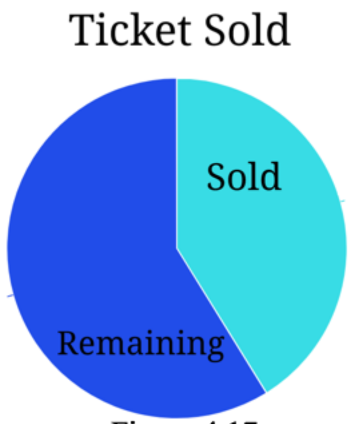


Figure 4.17

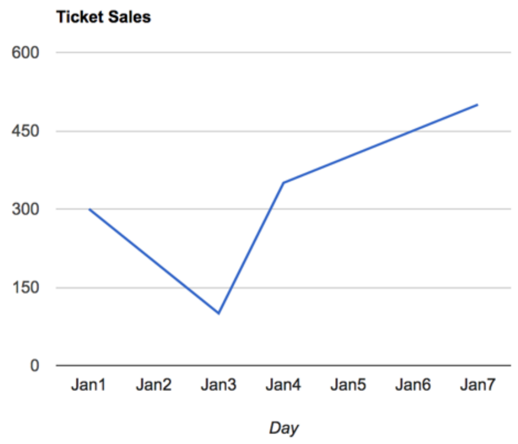


Figure 4.18

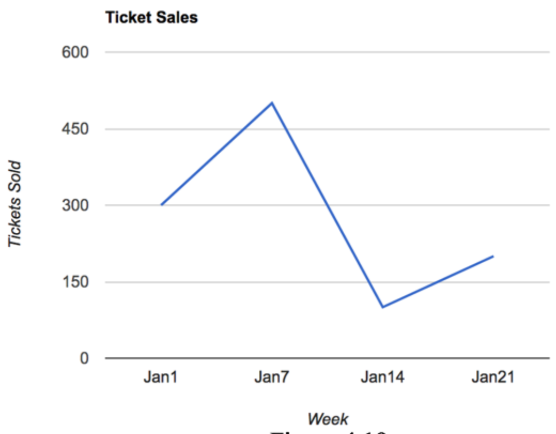


Figure 4.19

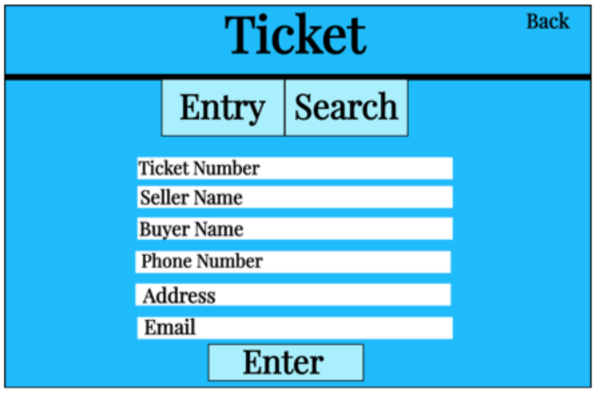


Figure 4.20

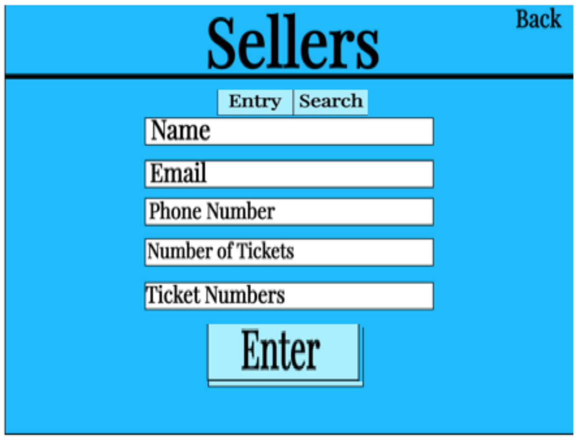


Figure 4.21

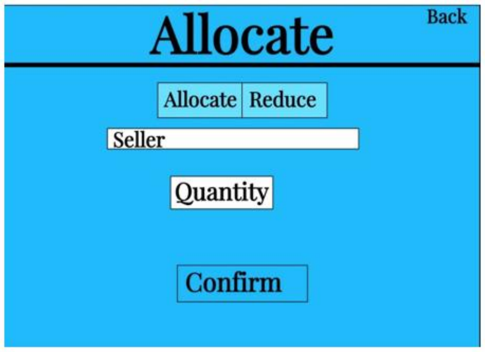


Figure 4.22

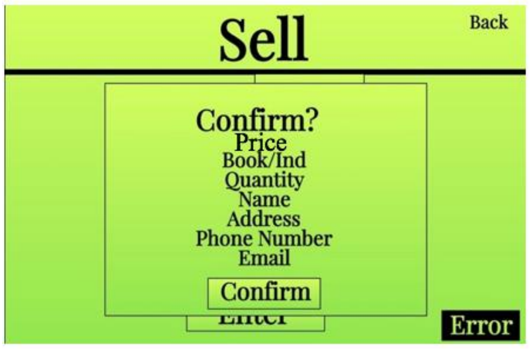


Figure 4.23

REQUIREMENTS MATRIX

Functional Requirement ID	Use Case Description	Priority	Actors	System Component(s)
F-01	Guest user accesses the "About" webpage	4	Guest	- HTTP Request Processing - Webpage Generation - AboutView
F-02	Guest user accesses the "Contact Us" webpage	4	Guest	- HTTP Request Processing - Webpage Generation - ContactView
F-03	Seller registration through sign up webpage	2	Seller User Database	- Seller - SignUpController - SignUpView

			Administrator	
F-04	Seller user login through login webpage	1	Seller	- Seller - LoginController - LoginView
F-05	Seller forgot account password	2	Seller Administrator	- LoginView - Forgot password request - Administrator - SellerInfoController
F-06	Seller sells a ticket to a customer	1	Seller Customer Ticket Database	- Seller - TicketSaleController - TicketSaleView
F-07	Seller account activation	1	Administrator	- Administrator - ActivateSellerController
F-08	Edit seller account details	1	Administrator	- Administrator - SellerInfoController
F-09	Administrator sells a ticket to a customer	1	Administrator	- Administrator - TicketSaleController - TicketSaleView - Ticket
F-10	Edit ticket information	1	Administrator	- Administrator - Ticket
F-11	Fulfill seller ticket request	1	Administrator	- Administrator - AllocateTicketController - AllocateTicketView - Ticket

RESOURCE ESTIMATES

The Universal Raffle Management System (URMS) will require a physical server that will host the database, which responds to requests from Seller and Administrator accounts through the Internet. The size of the server is dependent on the scope of the raffle itself. The larger the raffle, the higher the hardware requirements. This is also true for every additional year the raffle takes place. The client has limited the total amount of tickets to be made available for purchase, and therefore the storage requirements are low. The URMS implements a website to host the online client-side of the application, implying that the Sellers and Administrators will need devices capable of accessing the Internet. The demands of the web application are minimal so any device that can access the Internet should be capable of running the application.