Software Engineering Program ENSF614–Fall 2021

Term Project: Design Phase Document

Group Members:

Zach Frena Ahmad Elkholy Kelten Falez Evan Cooksley

Table Of Contents

System requirements:	3
System Use Case Diagram	4
System Activity Diagram	5
State Transition Diagram	6
Scenarios	7
Unregister Scenario:	7
Register Scenario:	7
Buy Ticket Scenario:	7
Cancel Ticket Scenario:	7
System Interaction Diagrams	8
Design Level Class Specification	12
Class Diagram - No Relationships	13
Model	13
View	14
Controller	15
Package Diagram	16
Deployment Diagram	17

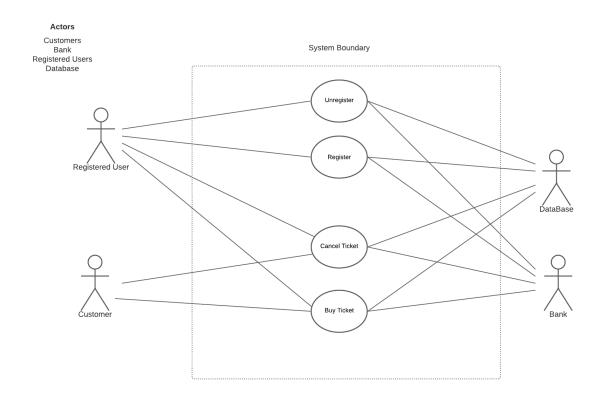
System requirements:

Our task is to analyze, design, and develop a system that can be used by two group of users:

- 1. Ordinary users that can search for a movie, select an specific theater, view available show times, view graphically available seats, select the desired seat, make payment by credit card, receive a copy of ticket and the receipt, via email. Users also should be able to cancel their ticket only up to 72 hours prior to show and receive a credit with %15 administration fee for future purchase up to a maximum of one-year expiration date.
- 2. Registered Users (RUs) whose information such as name, address, credit and/or debit card account must be saved on the system's database. RUs must pay a \$20.00 annual account fee, but they don't have to pay a 15% admin fee for cancelling their tickets and will receive the movie news before public announcement. There is also one more constraint: Only 10% of the seats can be purchased by RUs on a first come first serve policy prior to public announcements.

The implementation of this system/application will be limited to a single theatre development.

System Use Case Diagram



System Activity Diagram

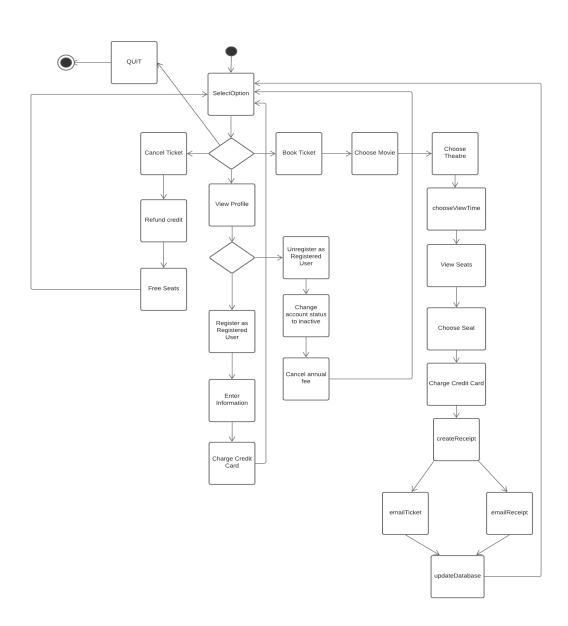
Register

Enter Details Register (Check if already registered) Save details to db Charge annual fee Update permissions Buy Ticket

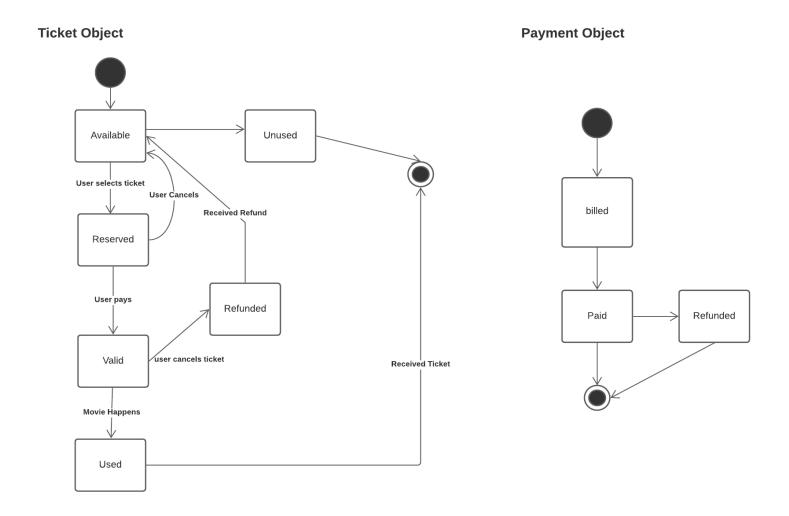
Search for movie Select Theatre Select Showtime View Seats Select Seats Make Payment Receive Ticket and Receipt Cancel Ticket

Choose ticket to cancel from active tickets Check if registered user Refund credit (charge admin fee if not RU) Delete reservation Unregister

Log in UnRegister Save details to db Update permissions



State Transition Diagram



Scenarios

*Assume that the names (Zach, Kelten, etc.) represent users of our system.

Unregister Scenario:

Kelten no longer wants to be a <u>registered user</u> in the system. Kelten selects **unregister** on his <u>profile</u>. System **confirms** with Kelten that he no longer wants to be a <u>registered user</u>. Kelten **confirms**. System **sets Kelten's status to inactive and notifies the billing system** to no longer charge his <u>account</u> for annual fees.

Register Scenario:

Zach wants to **register** in the system. Once this option is selected, the system prompts Zach to **enter** their <u>personal information</u> (email, billing address, <u>payment</u> details and name). Once valid information is entered, Zach will be **registered** into the system. His <u>account</u> will be **charged** its annual dues immediately and then again once every year. Zach can now **book** <u>movies</u> before they are **released** to the public and get full credit back when they **refund** a <u>ticket</u>.

Buy Ticket Scenario:

Evan wants to **buy** a ticket in the theatre registration app. Evan will be able to **view** the available <u>movies</u>. If Evan is a <u>Registered User</u> he can **view and book** available movies before the general public, as long as less than 10% of the <u>tickets</u> are **sold** before general release. After Evan **selects** the <u>movie</u>, he will **select** the <u>showtime</u>, view his <u>seat</u> via a GUI, **select** his <u>seat</u>, and make the **payment** through his credit card. Once completed, he will be **sent** a copy of the <u>ticket</u> and the receipt in his email.

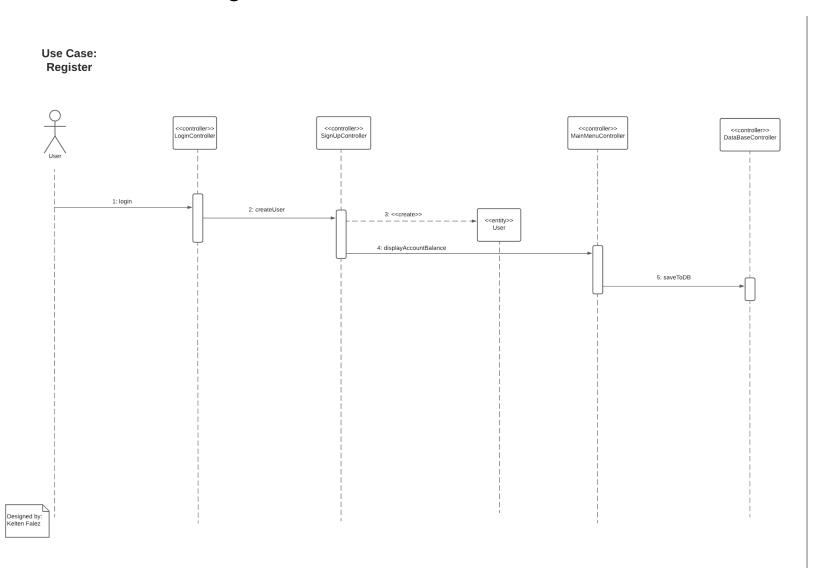
Cancel Ticket Scenario:

Ahmad wants to **cancel** a <u>ticket</u> from the app. Ahmad will be able to **view** the <u>ticket(s)</u> he **bought**. He then selects which <u>ticket(s)</u> to be **canceled**. Ahmad will be **refunded** a credit for the <u>ticket</u> he bought. If Ahmad is a <u>RU</u> he will **receive** the full value of the <u>ticket</u>. Otherwise he will be **charged** a 15% admin fee for cancellation.

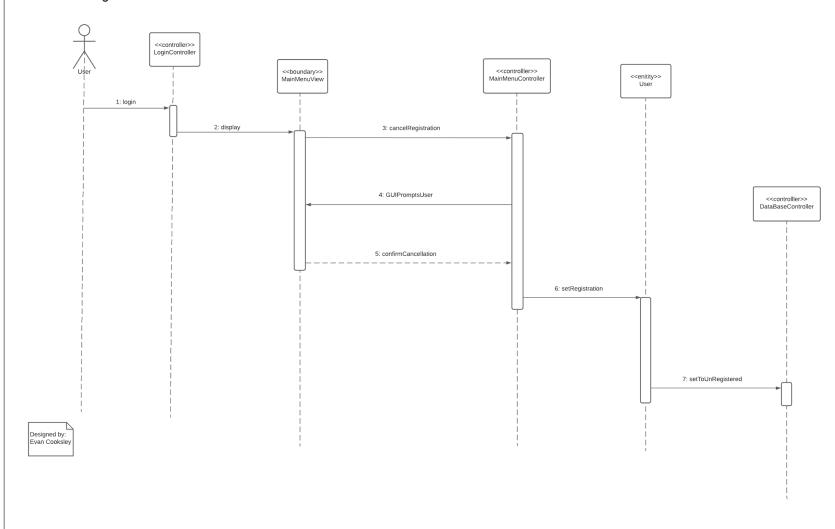
After the <u>ticket</u> has been **cancelled**, the chosen <u>seats</u> will be **released** and become available for others to choose/**reserve**.

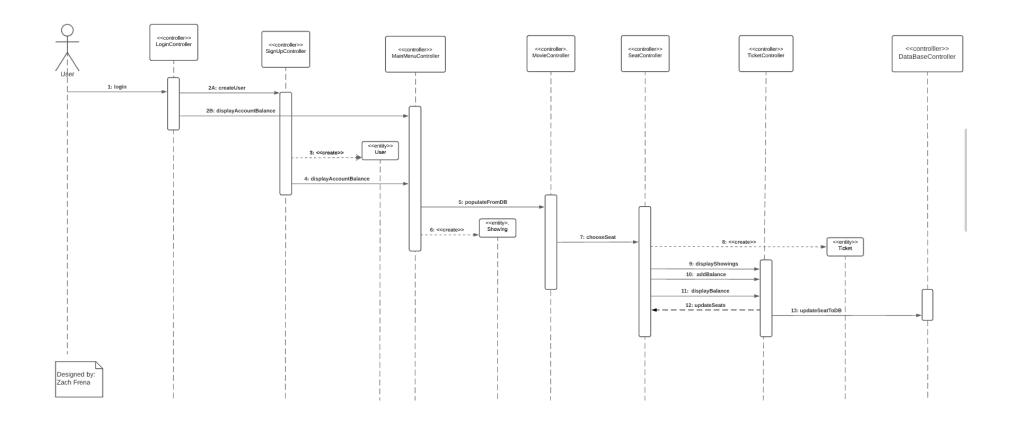
^{**}We used single-underline for objects and **BOLDED TEXT** for operations.

System Interaction Diagrams

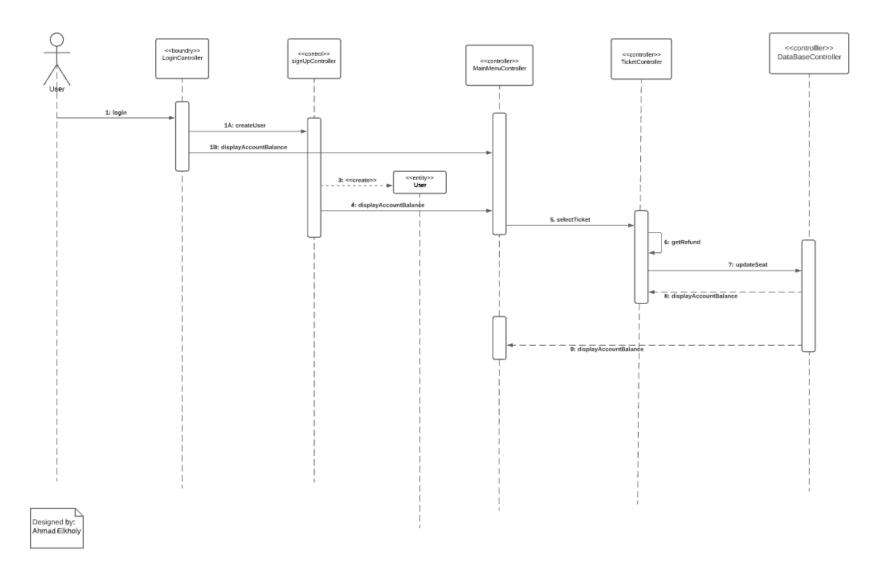


Use Case: Unregister

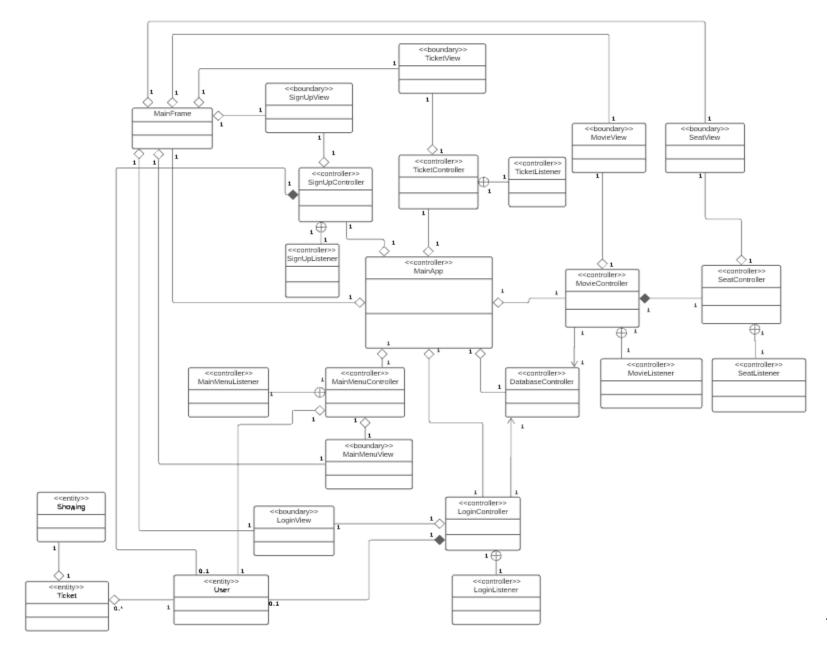




Use Case: Cancel Ticket



Design Level Class Specification



Class Diagram - No Relationships

Model

< <controller>> LoginListener</controller>	< <entity>> User</entity>	< <entity>> Ticket</entity>
+ actionPerformed(e: ActionEvent): void	- firstName: String - lastName: String - email: String - isRegistered: boolean - ticketList: ArrayList <ticket> - accountBalance: int</ticket>	-ticketID: int -showing: Showing -user: User +emailTicket():void
< <controller>> MainMenuListener</controller>	+setRegistration(status: boolean):void	< <entity>></entity>
	_	Showing
+ actionPerformed(e: ActionEvent): void		-theatre: String -seat: String -movie: String
< <controller>> MovieListener</controller>	< <controller>> SignUpListener</controller>	-showingTime: Date +displaySeats(): void +displayShowTlme(): void
+ actionPerformed(e: ActionEvent): void	+ actionPerformed(e: ActionEvent): void	
< <controller>> SeatListener</controller>	< <controller>> TicketListener</controller>	
+ actionPerformed(e: ActionEvent): void	+ actionPerformed(e: ActionEvent): void	

View

<<bod><
MainFrame

- mainMenuView: MainMenuView
- loginView: LoginView
- movieView: MovieView
- -seatView: SeatView
- -ticketView: TicketView
- -signUpView: SignUpView
- cards: CardLayout
- cardPanel: JPanel
- + setCard(int i): void

<
boundary>>
MainMenuView

- -nameCard: JLabel-movieButton: JButton
- ticketButton: JButton
- -addButton: JButton
- -accountBalance: JTextField
- -addAmount: JTextField
- -cancelRegistration: JButton
- +addActionListener(listener: ActionListener): void

<<body>
SignUpView</br/>

-fName: JTextField -lName: JTextField -email: JTextField -userName: JTextField -creditNum: JTextField -registered: JCheckBox -confirm: JButton

+addConfirmListener(listener: ActionListener): void

<
boundary>>
LoginView

-userNameBox: JTextField-loginButton: JButton

+addLoginListener(listener: ActionListener): void

<
boundary>>
MovieView

-movie1: JButton -movie2: JButton -theatre1: JButton -theatre2: JButton -time1: JButton -time2: JButton -back: JButton -movie: JLabel -theatre: JLabel

+addActionListener(listener: ActionListener): void +addBackListener(listener: ActionLlstener): void

<<bod><
SeatView

-seat1: JButton -seat2: JButton -seat3: JButton -seat4: JButton -seat 5: JButton -seat6: JButton -confirm: JButton -back: JButton

+addActionListener(listener: ActionListener): void +addBackListener(listener: ActionListener): void

<<bod><
TicketView

- showingList: Showing[]
- showing: String
- cancelButton: JButton
- backButton: JButton
- ticketSelector: JComboBox
- accountBalance: JLabel
- +addActionListener(listener: ActionListener): void +addBackListener(listener: ActionListener): void

Controller

<<controller>> LoginController

-loggedInUser: User -loginView: LoginView

+login(userName:String): void -isLoggedIn():boolean

<<controller>> MovieController

-movieList: ArrayList<Showing>
-movieView: MovieView

+populateFromDB(): void

<<controller>> SeatController

availableSeats: ArrayList<String>-seatView: SeatView

+chooseSeat(): void -checkOut(): void

<<controller>> TicketController

-ticketView: TicketView

-user: User

+selectTicket(): Ticket

+getRefund(int ticketID): void

+ displayShowings(showingList): void

+ displayAccountBalance(): void

+addBalance(user: User, amount: int): void

+ updateSeats(): void

<<controller>> DatabaseController

+loadDB(): void

+saveToDB(user: User): void +updateSeat(ticket: Ticket):void +setToNonRegistered(user: User): void

<<controller>> SignUpController

-signUpView: SignUpView

+createUser(): void +payAnnualFee(): void

<<controller>> MainApp

-frame: MainFrame

-ticketController: TicketController

-databaseController: DatabaseController -signUpController: SignUpController -mainController: MainMenuController

+run(): void

-buildMainFrame(): void -buildTicket(): void -buildDataBase(): void -buildSignUP(): void -buildMainMenu(): void

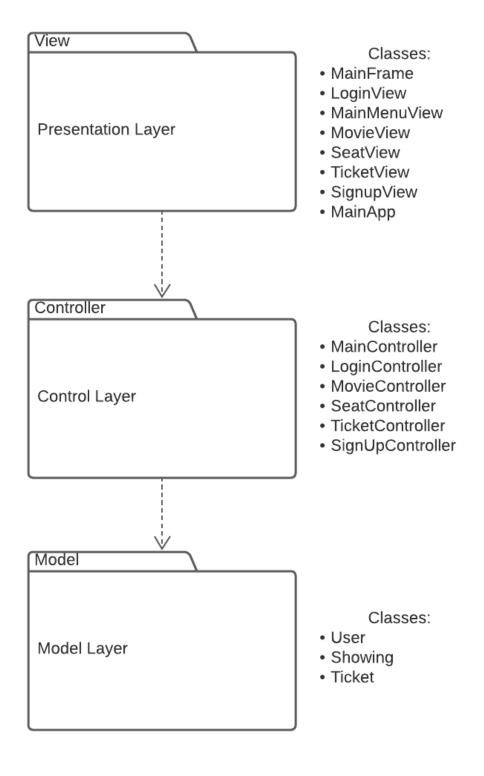
<<controller>> MainMenuController

-user: User

-mainMenuView : MainMenuView

+checkBalance(user: User): boolean +addBalance(user: User, amount: int): void +cancelRegistration(user: User): void +displayAccountBalance():void

Package Diagram



Deployment Diagram

