ENSF614 Fall 2022

Final Project

Design Report

December 3, 2022

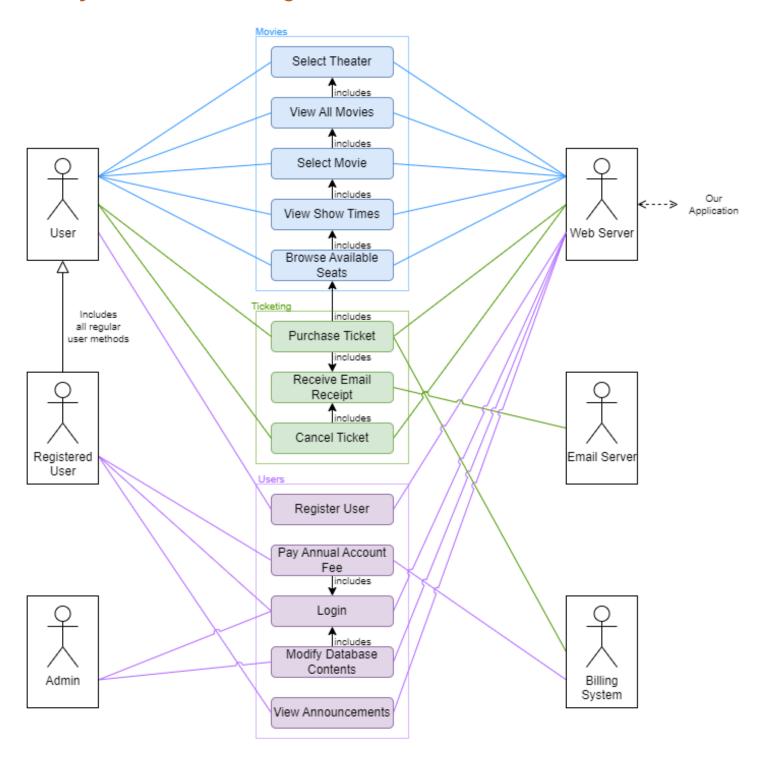
Group 8

Ryan Ahlstrom Andrew Bright
Dhruvi Dave Dele Oyelese

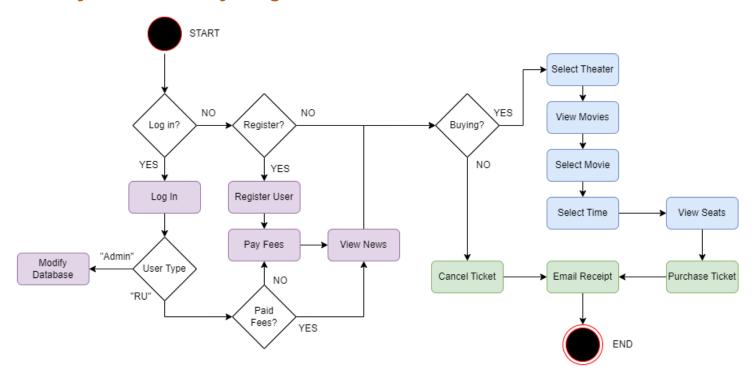
Table of Contents

Systems Use Case Diagram	3
Systems Activity Diagram	
Jse Case Scenarios	5
Select Theater	5
View All Movies	5
Select Movie	5
View Show Times	5
Browse Available Seats	5
Purchase Ticket	5
Cancel Ticket	6
Receive Email Receipt	6
Register User	6
Pay Annual Account Fee	6
Login	6
Modify Database Contents	7
View Announcements	7
Filtered Table	7
State Transition Diagrams	8
Ticket Object	8
Payment Object	g
Use Case — Login	10
Use Case — Pay Annual Fees	10
System Interaction Diagrams	11
Use Case — Login	11
Use Case — Purchase Ticket	12
Use Case — Cancel Ticket	13
Use Case – Browse Seats	14
Design Level Class Specification – Relationships	15
Design Level Class Specification - Attributes	16
Package Diagram	17
Conforment Diagram	10

Systems Use Case Diagram



Systems Activity Diagram



Use Case Scenarios

Select Theater

After the <u>user selects</u> the option to begin <u>ticket purchase</u>, a list of all available movie <u>theaters</u> is <u>displayed</u> on the page.

User selects a movie theater from the list.

View All Movies

After selecting a <u>theater</u>, a new page is <u>shown</u> to the user with a list of <u>movies</u> being played at the selected <u>theater</u>.

The user views all <u>movies</u> in an alphabetical list. The user decides they want to find their desired movie, so they enter a <u>title</u> into the search bar.

After entering the title into the search bar, only the movies which match the title are displayed to the user.

Select Movie

With the list of <u>movies</u> shown on the page, the user <u>clicks on</u> the <u>movie</u> they want to watch.

View Show Times

After selecting the movie, a new page is shown to the user.

The user views all available <u>show times</u> for the selected <u>movie</u> at the selected <u>theater</u>, and they select a <u>showtime</u> by clicking on it.

Browse Available Seats

After selecting a <u>show time</u>, the user<u>views a colored map</u> of all available <u>seats</u> for the selected <u>show time</u>. The <u>seats</u> are <u>displayed by number and availability</u>.

The <u>user</u> clicks on the <u>seat</u> they want, and the application verifies that the <u>seat</u> is available.

The <u>user</u> decides they want to buy a <u>seat</u> for their friend, so they <u>click on another</u> <u>seat</u>. The <u>application verifies</u> that the user's <u>seats</u> will not exceed the limit of 10% capacity for a reserved <u>showing</u>.

After the user has selected all of the seats they want, they click on a payment button.

Purchase Ticket

After the user <u>clicks on the payment button</u>, a new page is displayed with several fields for the user to <u>enter</u> information into.

The input fields displayed are "Card number", "Expiration date", "CSV", and "email address".

If the <u>user</u> is currently logged in as a <u>registered user</u>, these fields are retrieved from the <u>server</u> and filled automatically.

At the bottom of the page is a button labeled "Pay for tickets"

The <u>user presses the button</u>, and the <u>application contacts the billing system</u> to <u>determine if the payment is valid</u>.

If the payment is invalid, an error message is shown.

Cancel Ticket

After pressing the cancel ticket button, a new page is shown with several fields for the user to enter.

The displayed input fields are "Movie title", "Movie time", and "Email address".

The user <u>enters all information</u> into these fields, and <u>clicks a button</u> on the bottom of the page labelled "confirm cancellation"

The <u>application</u> contacts the <u>server</u> to <u>determine</u> if the <u>ticket</u> exists and the <u>movie show time</u> is <u>greater than</u> 72 hours from the current time.

If the <u>ticket</u> cancellation is not possible, an error message is shown.

The <u>application contacts</u> the <u>billing system</u> if the <u>ticket</u> cancellation is possible, and <u>distributes a refund</u> to the user depending on the user type. 85% for <u>nonregistered users</u>, and 100% for <u>registered users</u>.

If the <u>refund</u> is unsuccessful, an error message is shown.

Receive Email Receipt

After the <u>application confirms the validity</u> of a <u>purchase</u> or <u>refund</u>, a new page is shown.

This page lists the payment amount and the email address of the user.

The <u>user</u> is able to <u>press a button</u> labelled "email receipt" to <u>receive</u> an <u>emailed copy</u> of the <u>receipt</u>.

Register User

After <u>clicking on the "register" button</u>, the <u>user</u> is shown a new <u>registration</u> page

This page has several listed input fields <u>"First name"</u>, <u>"Last name"</u>, <u>"Address"</u>, <u>"Email address"</u> and <u>"Credit</u> card information".

Pay Annual Account Fee

After registering for the first time or <u>logging in as a registered user</u> with overdue <u>annual fees</u>, the <u>payment</u> page for <u>annual fees</u> is shown.

Since the <u>user</u> has their <u>credit card details</u> on file, they do not need to enter the <u>credit card details</u>.

There are two buttons displayed on the screen "Charge credit card" or "Continue as guest".

Since the user wants to remain a registered user, they press the button to charge their credit card.

Login

The user types in the <u>movie website address</u> into their web browser, and they are <u>redirected to</u> a <u>login</u> page.

The <u>login</u> page displays two input fields <u>"Username"</u> and <u>"Password"</u>

Several buttons are shown on the page "Log in", "Register", and "Continue as a guest"

If the user is a <u>registered user</u> or an <u>administrator</u>, they <u>enter their login details</u> into the two input fields and press the "Log in" button. The <u>application</u> validates the information.

The <u>user</u> is able to select any of the 3 options. Since the user is a guest, <u>they click the</u> "Continue as a guest" option.

After selecting the guest option, the user is given an option to purchase or cancel a ticket.

Modify Database Contents

If the user <u>pressed the "Log in" button</u> with valid <u>administrator credentials</u>, a new page is shown.

This is an <u>administrator</u> console where the <u>admin</u> user can enter <u>API calls</u> to modify <u>database</u> contents.

View Announcements

After <u>logging in or paying annual fees</u>, a <u>registered user</u> is sent to a new page.

On the new page are movie announcements not available to the public.

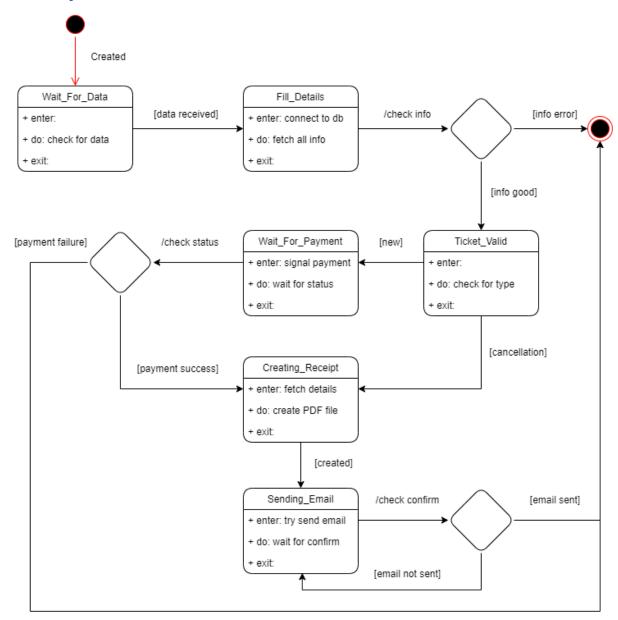
At the bottom of the page are two buttons, "Purchase a ticket" or "Cancel a ticket".

Filtered Table

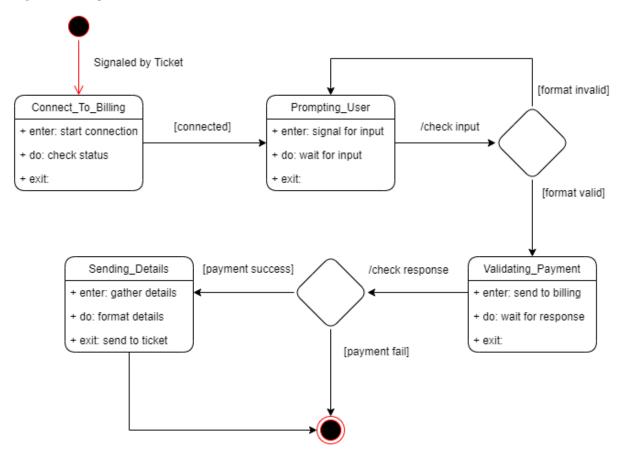
Noun	Filtering Decision
User	Candidate object
Theater	Candidate object
Movie	Candidate object
Showtime	Candidate object
Ticket	Candidate object
Seat	Candidate object
Registered User	Candidate object
Administrator	Candidate object
Guest	Filtered (not stored)
Application	Filtered (out of scope)
Server	Filtered (out of scope)
Billing System	Filtered (out of scope)
Card number, Expiration, CSV, email address	Filtered (attributes of object)
Movie time, Movie title, email address	Filtered (attributes of object)
Refund	Candidate object
Purchase	Candidate object
Payment	Candidate object
Receipt	Candidate object
Emailed copy	Filtered (duplicate of receipt)
First name, last name, address, email address, credit	Filtered (attributes of object)
card information	
Annual fees	Filtered (attribute or method)
Credit card details	Filtered (attribute of object)
Website address	Filtered (out of scope)
Username, password	Filtered (attributes of object)
Movie announcements	Candidate object
API calls	Filtered (out of scope)

State Transition Diagrams

Ticket Object

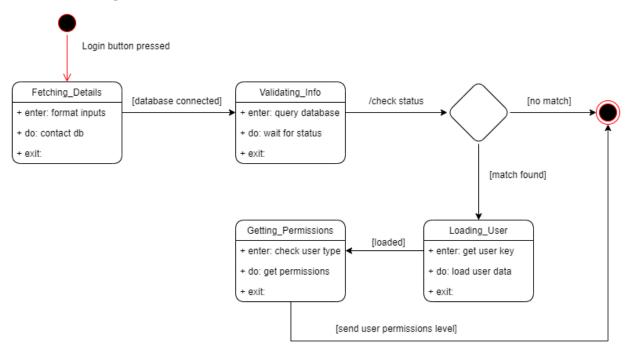


Payment Object

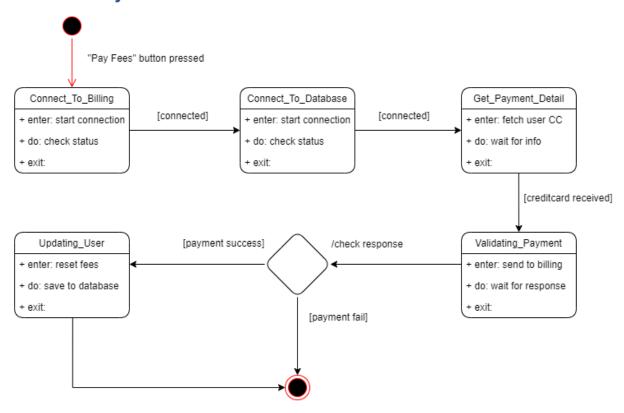


ENSF614 Final Project: Design Report

Use Case - Login

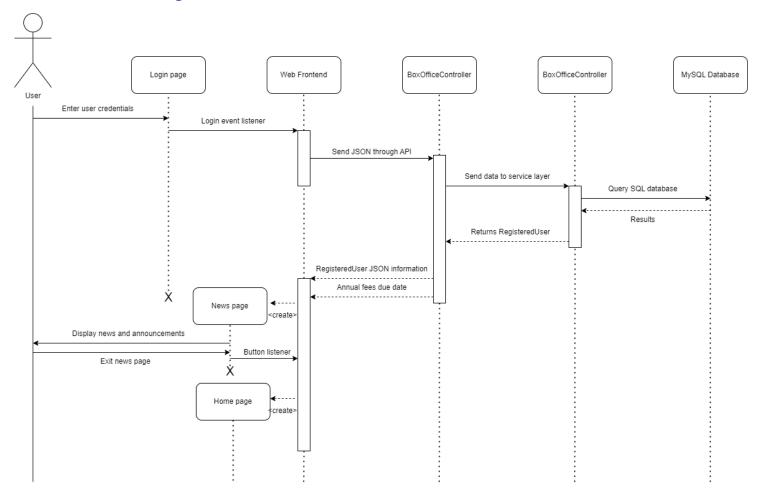


Use Case - Pay Annual Fees

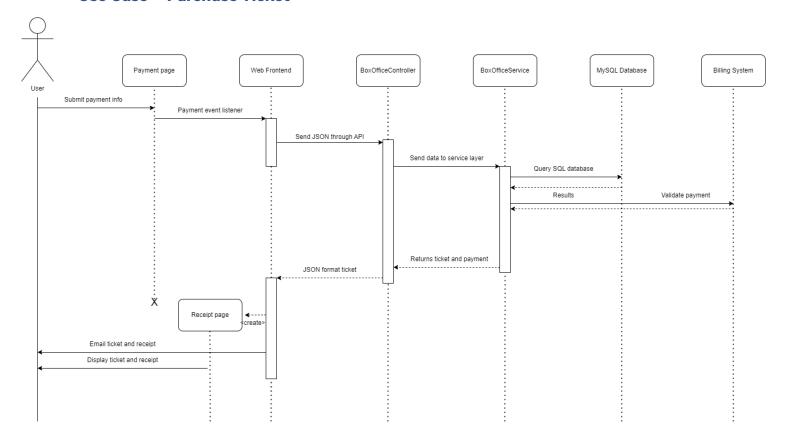


System Interaction Diagrams

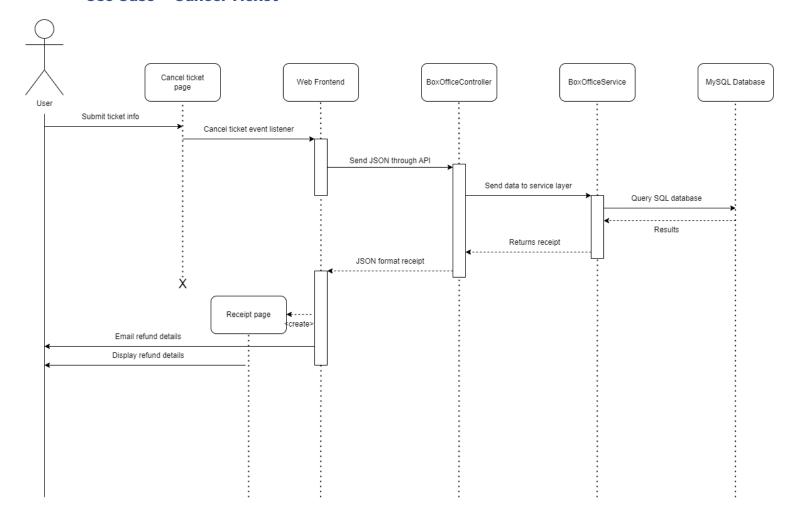
Use Case – Login



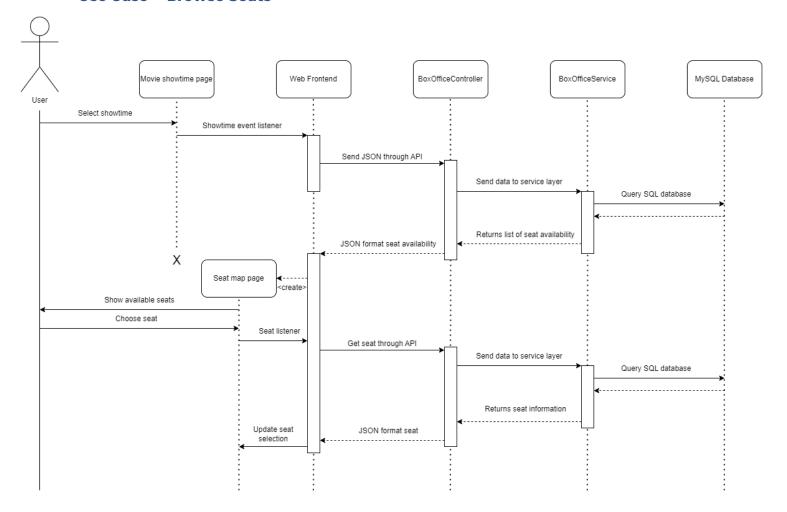
Use Case – Purchase Ticket



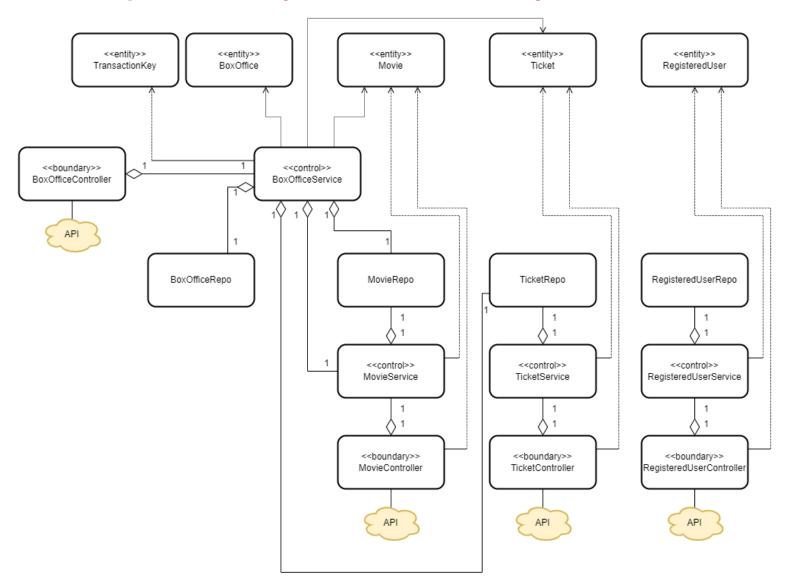
Use Case – Cancel Ticket



Use Case - Browse Seats



Design Level Class Specification – Relationships



Design Level Class Specification - Attributes

Model

BoxOffice

- id: TransactionKey
- + ticket: Ticket + movie: Movie

TransactionKey

+ ticketId: Long + movield: Long

Movie

- id: Long
- title: String
- showTime: String
- boxOffices: Set<BoxOffice>
- seat1: boolean
- seat2: boolean
- seat3: boolean
- seat4: boolean seat5: boolean
- seat6: boolean
- seat7: boolean seat8: boolean
- seat9: boolean
- seat10: boolean
- + addTicket(BoxOffice): void
- + removeTicket(BoxOffice): void
- equals(Object): boolean

Ticket

- id: Long
- seatNum: int
- price: double
- buyerStatus: boolean
- movieName: String
- mld: Long
- boxOffices: <Set>BoxOffices
- + addMovie(BoxOffice): void
- + removeMovie(BoxOffice): void

RegisteredUser

- id: Long name: String
- address: String email: String
- regDate: LocalDate
- password: String
- creditNum: int

Controller

BoxOfficeController

- boxOfficeService: BoxOfficeService
- + getAll(): List<BoxOffice>
- + getTicket(Long): BoxOffice
- + purchaseTicket(Long,Long,int,int): voi
- + cancelTicket(Long): void

MovieController

- movieService: MovieService
- + getMovies(): List<Movie>
- + getMovie(Long): Movie
- + deleteMovie(Long): void

TicketController

- ticketService: TicketService
- + getTickets(): List<Ticket>
- + getTicket(Long): Ticket + registerNewTicket(Ticket): void

RegisteredUserController

- userService: RegisteredUserService
- + getUsers(): List<RegisteredUser>
- + getUser(Long): RegisteredUser + getUser(String): RegisteredUser
- + registerNewUser(RegisteredUser): vo
- + updateUser(Long, String, String): void

Service

BoxOffice Service

- boxOfficeRepo: BoxOfficeRepo
- ticketRepo: TicketRepo - movieRepo: MovieRepo
- movieService: MovieService
- + getAllBoxOffice(): List<BoxOffice>
- + getOneTicket(Long): BoxOffice
- + purchaseTicketForMovie(
- Long, Long, int, int): void cancelTicketForMovie(Long): void

Movie Service

- movieRepo: MovieRepo
- + aetAllMovies(): List<Movie>
- + addNewMovie(Movie): void
- + purchaseSeat(Long, int): void
- + cancelSeat(Long, int): void + deleteMovie(Long): void
- + updateMovie(Long,String,String):void
- getMovieByTitle(String): Movie

TicketService

- ticketRepo: TicketRepo
- + getAllTickets(): List<Ticket>
- + addTicket(Ticket): void getTicketById(Long): Ticket

RegisteredUserService

- registeredUserRepo: RegisteredUserRepo
- + getAllRegisteredUsers(): List<RegisteredUser>
- addRegisteredUser(
- RegisteredUser): void
- getUserById(Long):
- RegisteredUser
- + getUserByEmail(String): RegisteredUser
- updateUser(Long, String, String): void

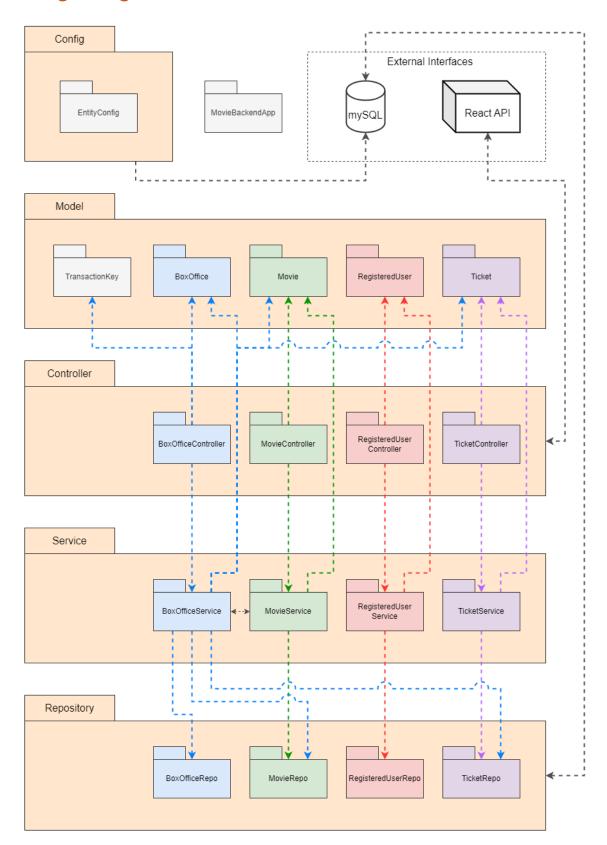
Repository

<<Interface>> BoxOfficeRepo <<Interface>> MovieRepo

<<Interface>> TicketRepo

<<Interface>> RegisteredUserRepo

Package Diagram



Deployment Diagram

