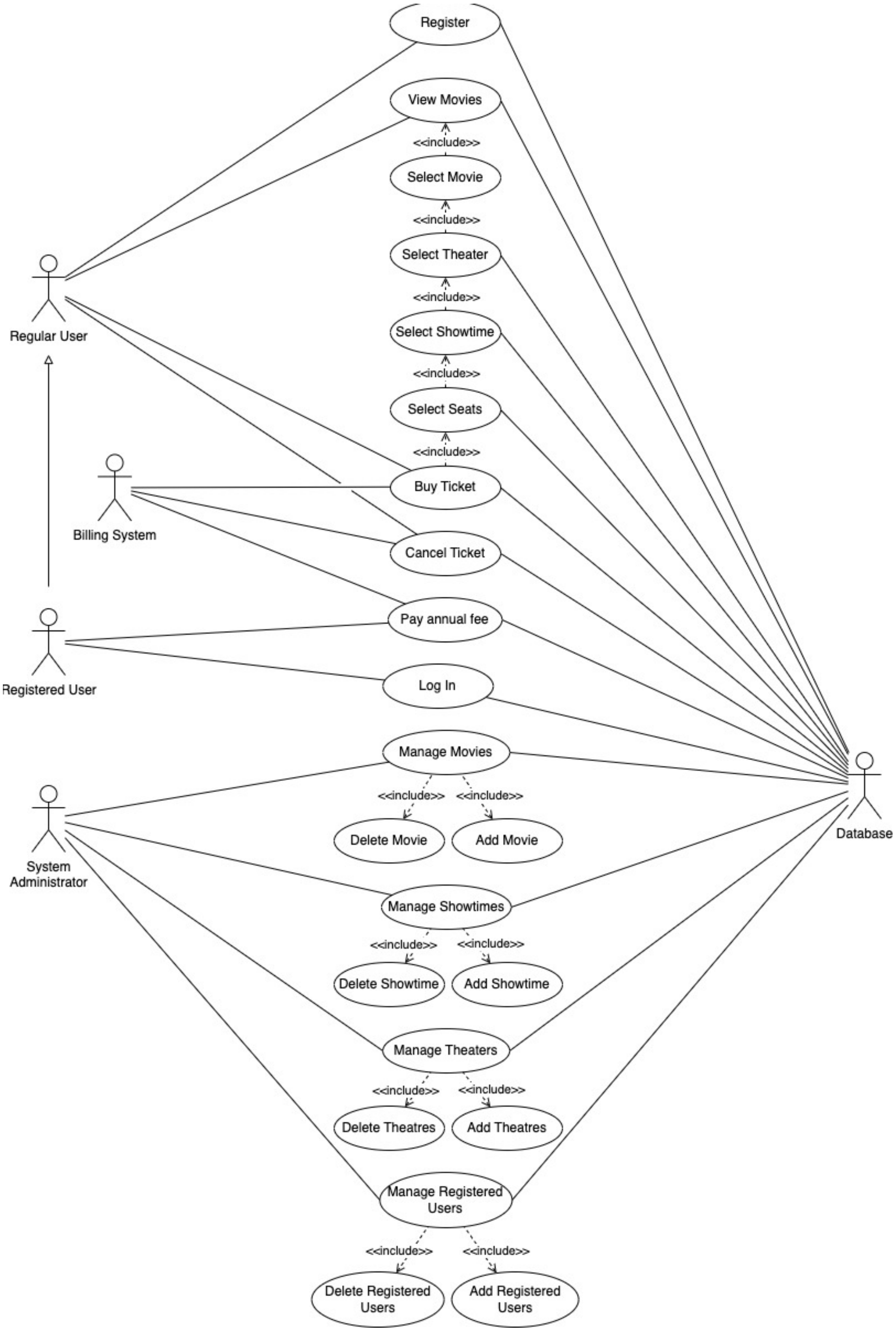
ENSF 614 Term Project: Design Report

Team Members: Kimberley Baldwin, Michael Metz,
Vladyslav Timofyeyev, David Zarinski

Due Date: December 4th, 2022

Use Case Diagram:



Use Case Scenarios:

Use Case	Actors	Regular and Registered Users
Register	Regular Users	User can *register* as a registered user by entering their name, email, address, credit card information and password. Duplicate user emails will not be allowed. Successful registration will show it was successful. Newly registered user will be redirected to the login page.
Login	Registered Users	Registered user can *login* by clicking on a login menu item, entering their email and password and *submitting* it. If it succeeds, they will be *taken back* to the view movies page. If it fails, an error message will be *displayed* explaining what went wrong, and the user will remain on the page.
View Movies	Regular and Registered Users	When visiting the website, the user will be able to *view* all publicly announced movies which are currently playing at theaters.
	Registered User	If logged in, in addition to publicly announced movies, registered users will also be able to *view* upcoming movies prior to their public announcement.
Authentication and Authorization	Registered User and System Administrator	This extends all use cases that make wish to make privileged requests to the API. If a user's authentication becomes unauthenticated or unauthorized by the time a privileged web request is submitted (ex. pre-release movie, registered user transaction), fail the request and *notify* the user to login or pay their membership.
Select Movie	Regular and Registered Users	From the available movies the user will *select* a movie they are interested in seeing.
Select Theater	Regular and Registered Users	After selecting a movie, the user will *see* all available theaters where the movie will have showtimes and *select* a theater. The user can *cancel* the selection page which will go back to the view movies page.

Use Case	Actors	Regular and Registered Users
Select Showtime	Regular and Registered Users	After selecting a theater the user can *see* all available showtimes for the movie at the given theater. The user can then select a showtime. The user can *cancel* the selection page which will go back to the view movies page.
Select Seats	Regular and Registered Users	After showtime is selected, the user will be *prompted* to *continue to seat selection*. User will graphically *view* seats. If a Registered User is not logged in, they will be treated as a Regular User. User can also choose to *cancel* and go back to the view movies page.
	Registered User	Registered user will *select* the desired available seat(s). If it is a pre-release movie, the user will be allowed to select seats until the 10% maximum seat limit is reached. Registered user will be prompted to *continue to payment*
	Regular User	Users will *select* desired available seat(s). User will be prompted to *continue to payment*
Buy Tickets	Regular and Registered Users	After seat selection, user will *view* the numbers of seats selected multiplied by ticket price and also *view* the total money owed in the transaction. If a Registered User is not logged in, they are treated as a Regular User. The user can optionally *cancel* at any time (other than when *submitting* a transaction) and go back to movie selection.

Use Case	**Actors**	**Regular and Registered Users**
	Registered User	Registered user will *view* the amount of cancellation credit that they can add (if available) and either click the *add credit* button to *add* the cancellation credit to the transaction or pay using their saved credit/debit information. If there is not enough cancellation credit applied, then a credit/debit card will be required. When there is enough cancellation credit *applied*, or a saved credit card is used, they will *submit* the transaction and the registered user will receive a copy of tickets via their email.
	Regular User	Regular user will enter their email address and optionally *add* one cancellation credit code to the transaction or start entering credit/debit information. If there is not enough cancellation credit applied, then a credit/debit card will be required. When there is enough cancellation credit *applied*, or a credit card is entered, the user can submit the payment. User will *receive* a transaction that shows ticket information and amount paid via the email address they entered.
	Billing System	Billing System *verifies* transaction submitted by user, and *sends to the financial institution*. If the transaction fails, the user is notified and remains on the page
Cancel Tickets	Regular and Registered Users	User can *navigate* to the cancel tickets page by clicking on a *Cancel Tickets* menu item. User will be *prompted* to enter ticket number and email to *cancel*. If cancellation window is closed, ticket has already been *cancelled* or ticket number is not found, user will receive an error message. If *cancel* is successful, cancellation credit information including code and expiry date will be *displayed* to user. User will also receive an email with this information.

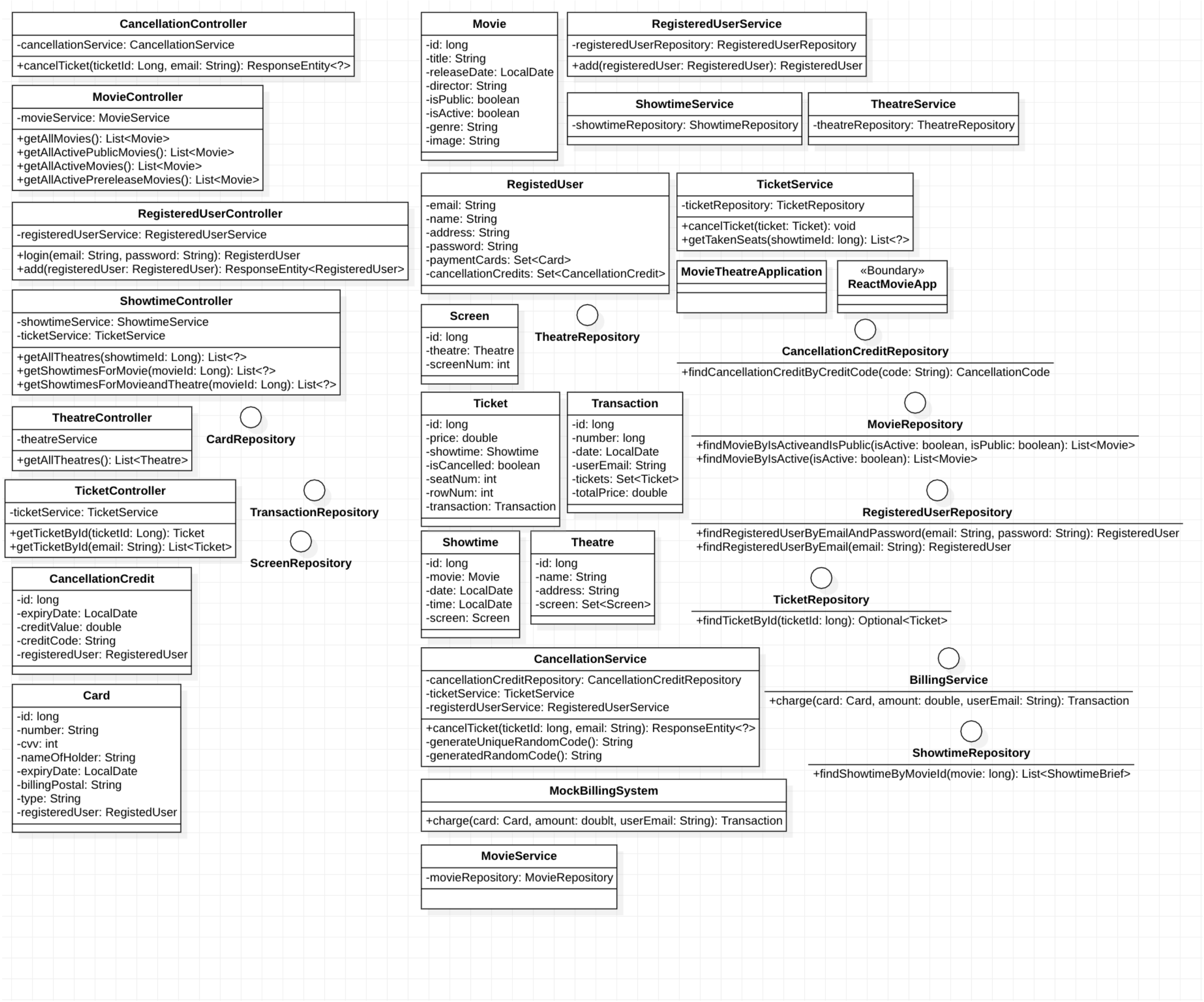
Use Case	Actors	Regular and Registered Users
	Billing System	Billing System *verifies* the tickets' transaction info submitted by the cancellation, and *sends to the financial institution* . If the transaction fails, the user is notified and remains on the page.
Pay Annual Fee	Registered User	Registered user can *navigate* to their membership page by clicking on a membership menu item (that shows when they are logged in) Every year user *pays* a $20.00 account fee (Membership Transaction) on the anniversary of their registration, using payment information associated with their Registered User. *Email* the user once the payment information expires. If the payment due date passes, mark the Registered user as unpaid (derived attribute).
	Billing System	Billing System *verifies* transaction submitted by user, and *sends to the financial institution*. If the transaction fails, the user is notified and remains on the page.
Manage Movies	System Administrator	User can *add* a movie to the catalogue. User can also *select* a movie, *modify* or *delete* it. When they *modify* a Movie, they can update attributes including making a Movie active, and or public. Registered users will be *emailed* pre-sale movie information as soon as a movie is active, not public, and has show times. If a movie is public, active, and has a showtime, they will also be *emailed* public sale Movie information.
Manage Showtimes	System Administrator	User can *select* a movie from catalogue, and choose to *add* a showtime to that movie, *modify* or *delete* an existing showtime.
Manage Theaters	System Administrator	User can *add* a theatre to the Theatre list, or *modify* a theatre, or *remove* an existing theatre from the list of theatres if not showtimes exist for the theatre's screens. When a theater is removed, all of its child data (screens) are deleted as well) Screens are also managed in the theatre's UI and API

Use Case	**Actors**	**Regular and Registered Users**
Manage Registered Users	System Administrator	Administrator can $\underline{\underline{delete}}$ registered users from the registered users list only if they have no usable cancellation credit (associated identifying information is removed (cancellation credit usage and transactions remain, and a delete marker is applied to the user). Administrator can also $\underline{\underline{add}}$ a new registered user to the list by entering all of the necessary information.
Persistence	Database	For all use cases that must persist information, this use case is included in those use cases. The database will $\underline{\underline{CRUD}}$ persistable information while web requests are processed on the server side.

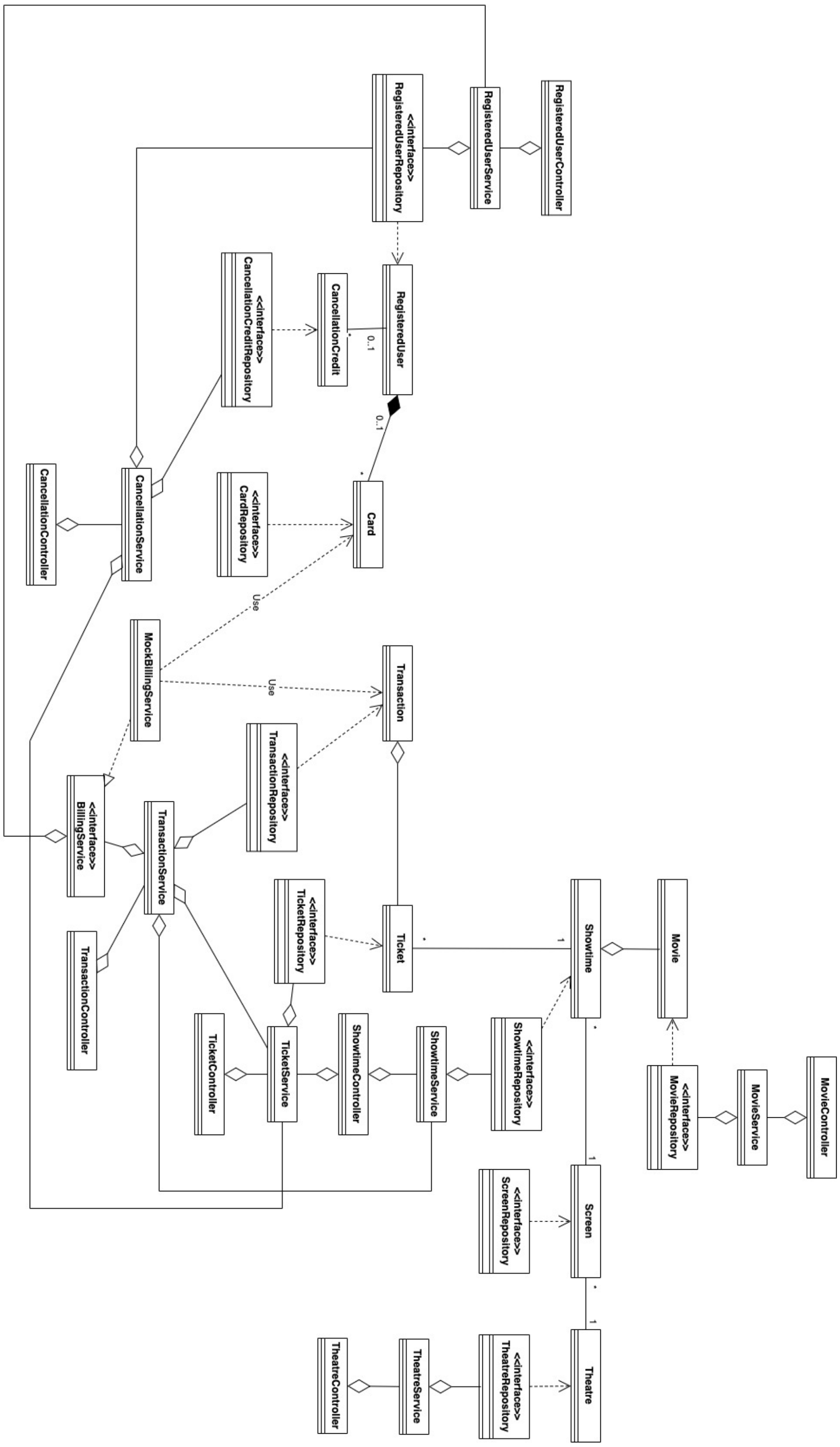
Candidate Objects:

Regular User (Only used on Client side, not persisted in the DB)
Registered User
Movie
Upcoming movie
Theater
Showtime
Seat
Ticket
Membership Transaction
Ticket Transaction
CreditDebit Card
Transaction
Cancellation
Billing System
Authentication
System Administrator
Database (Repository)

Class Diagram:


CancellationController
-cancellationService: CancellationService
+cancelTicket(ticketId: Long, email: String): ResponseEntity<?>
MovieController
-movieService: MovieService
+getAllMovies(): List<Movie>
+getAllActivePublicMovies(): List<Movie>
+getAllActiveMovies(): List<Movie>
+getAllActivePrereleaseMovies(): List<Movie>
RegisteredUserController
-registeredUserService: RegisteredUserService
+login(email: String, password: String): RegisterdUser
+add(registeredUser: RegisteredUser): ResponseEntity<RegisteredUser>
ShowtimeController
-showtimeService: ShowtimeService
-ticketService: TicketService
+getAllTheatres(showtimeId: Long): List<?>
+getShowtimesForMovie(movieId: Long): List<?>
+getShowtimesForMovieandTheatre(movieId: Long): List<?>
TheatreController
-theatreService
+getAllTheatres(): List<Theatre>
CardRepository
TicketController
-ticketService: TicketService
+getTicketById(ticketId: Long): Ticket
+getTicketById(email: String): List<Ticket>
TransactionRepository
ScreenRepository
CancellationCredit
-id: long
-expiryDate: LocalDate
-creditValue: double
-creditCode: String
-registeredUser: RegisteredUser
Card
-id: long
-number: String
-cvv: int
-nameOfHolder: String
-expiryDate: LocalDate
-billingPostal: String
-type: String
-registeredUser: RegistedUser
Movie
-id: long
-title: String
-releaseDate: LocalDate
-director: String
-isPublic: boolean
-isActive: boolean
-genre: String
-image: String
RegisteredUserService
-registeredUserRepository: RegisteredUserRepository
+add(registeredUser: RegisteredUser): RegisteredUser
ShowtimeService
-showtimeRepository: ShowtimeRepository
TheatreService
-theatreRepository: TheatreRepository
RegistedUser
-email: String
-name: String
-address: String
-password: String
-paymentCards: Set<Card>
-cancellationCredits: Set<CancellationCredit>
TicketService
-ticketRepository: TicketRepository
+cancelTicket(ticket: Ticket): void
+getTakenSeats(showtimeId: long): List<?>
MovieTheatreApplication
«Boundary»
ReactMovieApp
Screen
-id: long
-theatre: Theatre
-screenNum: int
TheatreRepository
CancellationCreditRepository
+findCancellationCreditByCreditCode(code: String): CancellationCode
Ticket
-id: long
-price: double
-showtime: Showtime
-isCancelled: boolean
-seatNum: int
-rowNum: int
-transaction: Transaction
Transaction
-id: long
-number: long
-date: LocalDate
-userEmail: String
-tickets: Set<Ticket>
-totalPrice: double
MovieRepository
+findMovieByIsActiveandIsPublic(isActive: boolean, isPublic: boolean): List<Movie>
+findMovieByIsActive(isActive: boolean): List<Movie>
RegisteredUserRepository
+findRegisteredUserByEmailAndPassword(email: String, password: String): RegisteredUser
+findRegisteredUserByEmail(email: String): RegisteredUser
Showtime
-id: long
-movie: Movie
-date: LocalDate
-time: LocalDate
-screen: Screen
Theatre
-id: long
-name: String
-address: String
-screen: Set<Screen>
TicketRepository
+findTicketById(ticketId: long): Optional<Ticket>
CancellationService
-cancellationCreditRepository: CancellationCreditRepository
-ticketService: TicketService
-registerdUserService: RegisteredUserService
+cancelTicket(ticketId: long, email: String): ResponseEntity<?>
-generateUniqueRandomCode(): String
-generatedRandomCode(): String
BillingService
+charge(card: Card, amount: double, userEmail: String): Transaction
ShowtimeRepository
+findShowtimeByMovieId(movie: long): List<ShowtimeBrief>
MockBillingSystem
+charge(card: Card, amount: doublt, userEmail: String): Transaction
MovieService
-movieRepository: MovieRepository

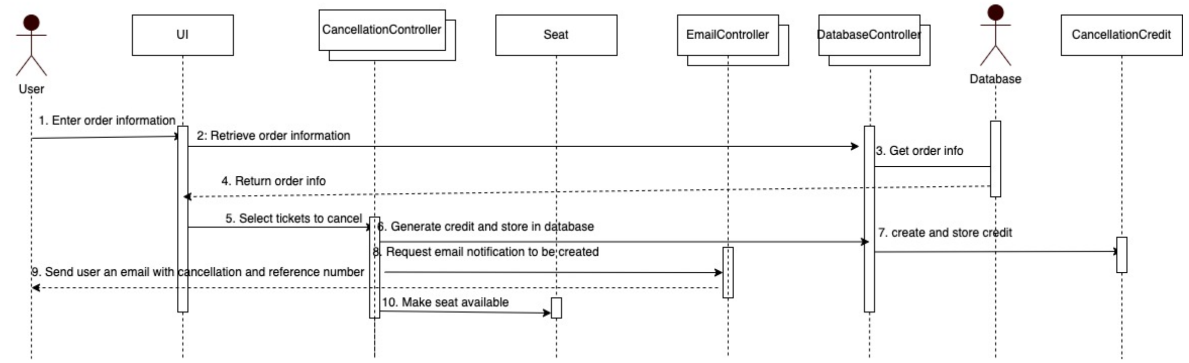
Class Relationship Diagram:



Sequence Diagrams:

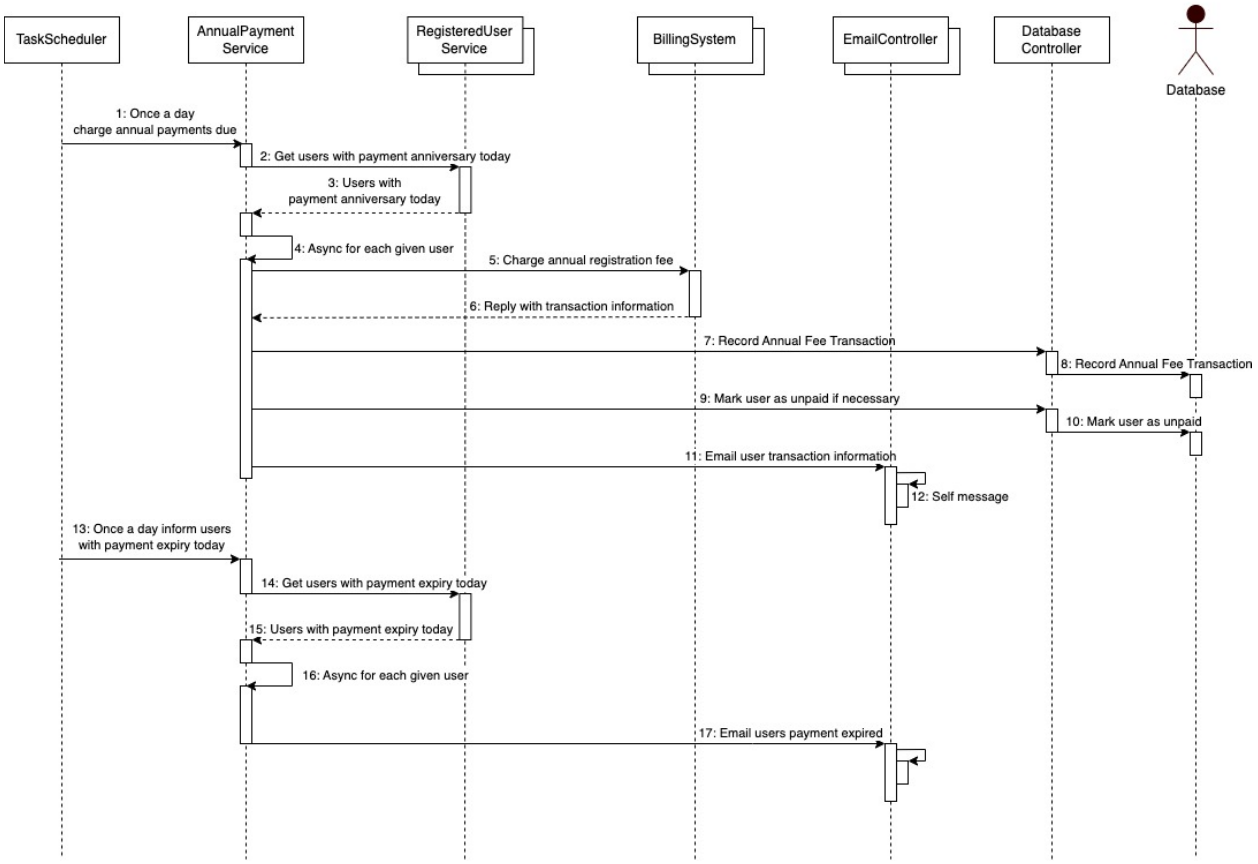
1. Cancel Ticket - Vlad

Vlad


User
UI
CancellationController
Seat
EmailController
DatabaseController
Database
CancellationCredit
1. Enter order information
2: Retrieve order information
3. Get order info
4. Return order info
5. Select tickets to cancel
6. Generate credit and store in database
7. create and store credit
8. Request email notification to be created
9. Send user an email with cancellation and reference number
10. Make seat available

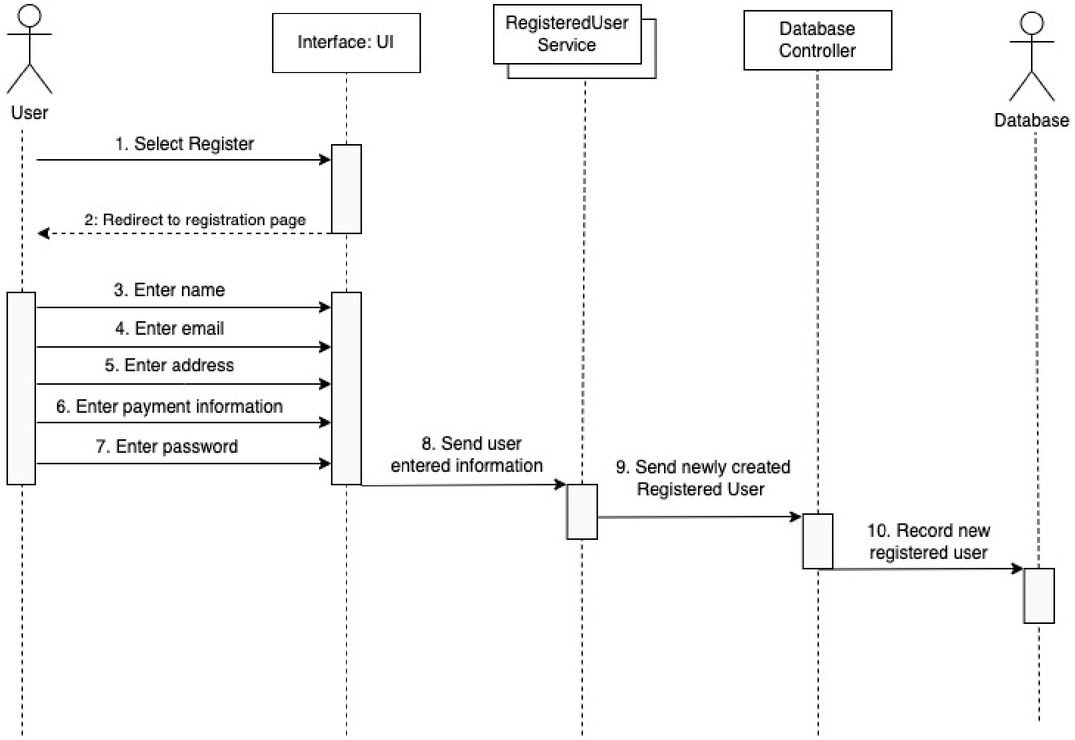
2. Pay Fee - David

David
Zarinski


TaskScheduler
AnnualPayment Service
RegisteredUser Service
BillingSystem
EmailController
Database Controller
Database
1: Once a day charge annual payments due
2: Get users with payment anniversary today
3: Users with payment anniversary today
4: Async for each given user
5: Charge annual registration fee
6: Reply with transaction information
7: Record Annual Fee Transaction
8: Record Annual Fee Transaction
9: Mark user as unpaid if necessary
10: Mark user as unpaid
11: Email user transaction information
12: Self message
13: Once a day inform users with payment expiry today
14: Get users with payment expiry today
15: Users with payment expiry today
16: Async for each given user
17: Email users payment expired

3. Register - Kim

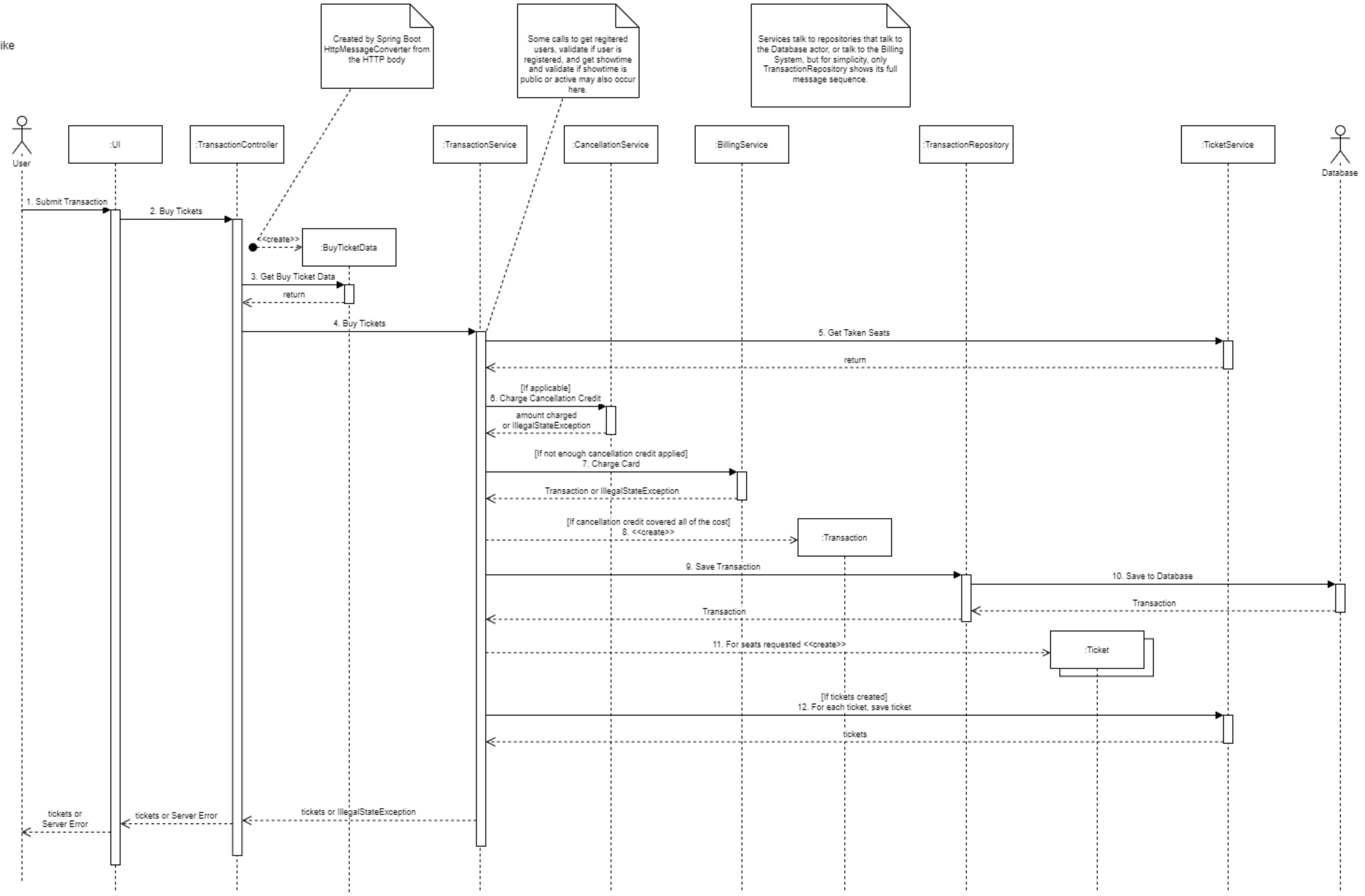
Kim



User
Interface: UI
RegisteredUser Service
Database Controller
Database
1. Select Register
2: Redirect to registration page
3. Enter name
4. Enter email
5. Enter address
6. Enter payment information
7. Enter password
8. Send user entered information
9. Send newly created Registered User
10. Record new registered user

4. Buy Ticket - Mike

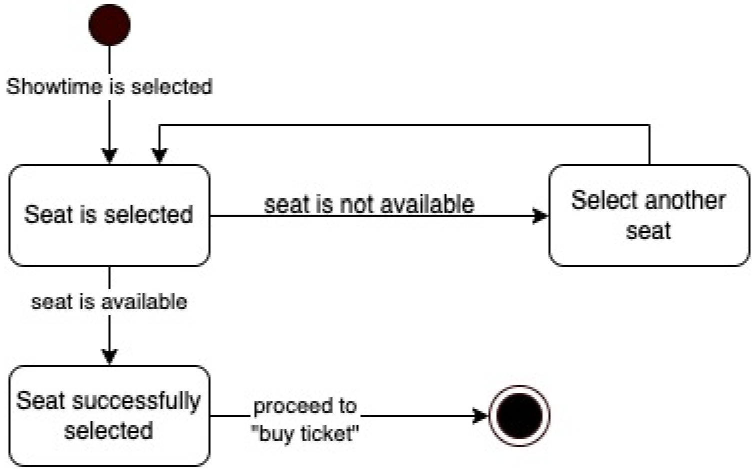
Mike


Created by Spring Boot HttpMessageConverter from the HTTP body
Some calls to get regitered users, validate if user is registered, and get showtime and validate if showtime is public or active may also occur here.
Services talk to repositories that talk to the Database actor, or talk to the Billing System, but for simplicity, only TransactionRepository shows its full message sequence.
User
:UI
:TransactionController
:TransactionService
:CancellationService
:BillingService
:TransactionRepository
:TicketService
Database
1. Submit Transaction
2. Buy Tickets
<<create>>
:BuyTicketData
3. Get Buy Ticket Data
return
4. Buy Tickets
5. Get Taken Seats
return
[If applicable]
6. Charge Cancellation Credit
amount charged
or IllegalStateException
[If not enough cancellation credit applied]
7. Charge Card
Transaction or IllegalStateException
[If cancellation credit covered all of the cost]
8. <<create>>
:Transaction
9. Save Transaction
10. Save to Database
Transaction
Transaction
11. For seats requested <<create>>
:Ticket
[If tickets created]
12. For each ticket, save ticket
tickets
tickets or IllegalStateException
tickets or Server Error
tickets or
Server Error

State Transition Diagrams:

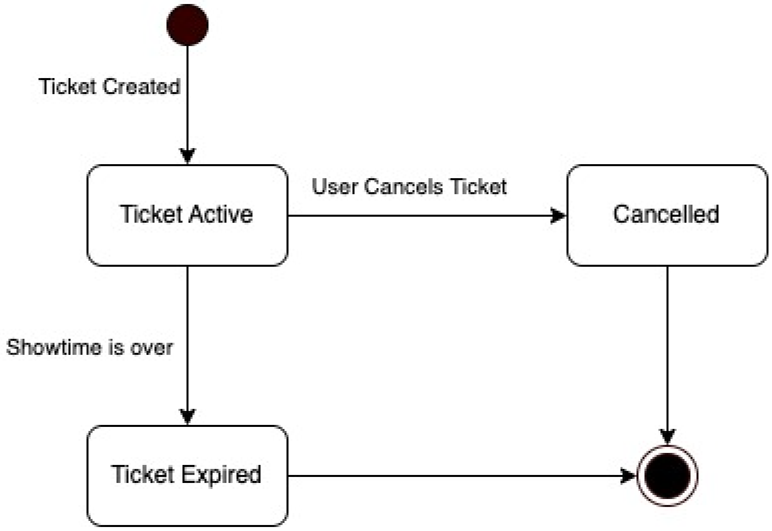
1. Select Seat Use Case - Vlad

Vlad

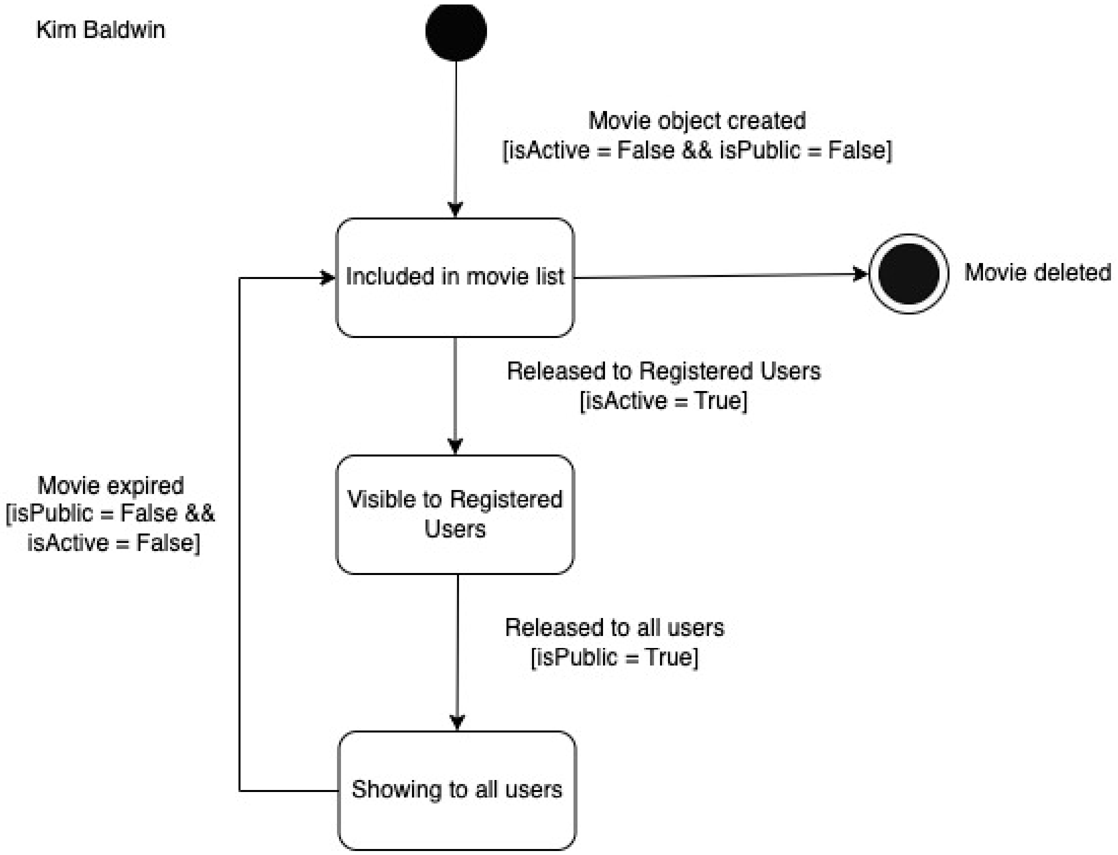


2. Ticket Object - David

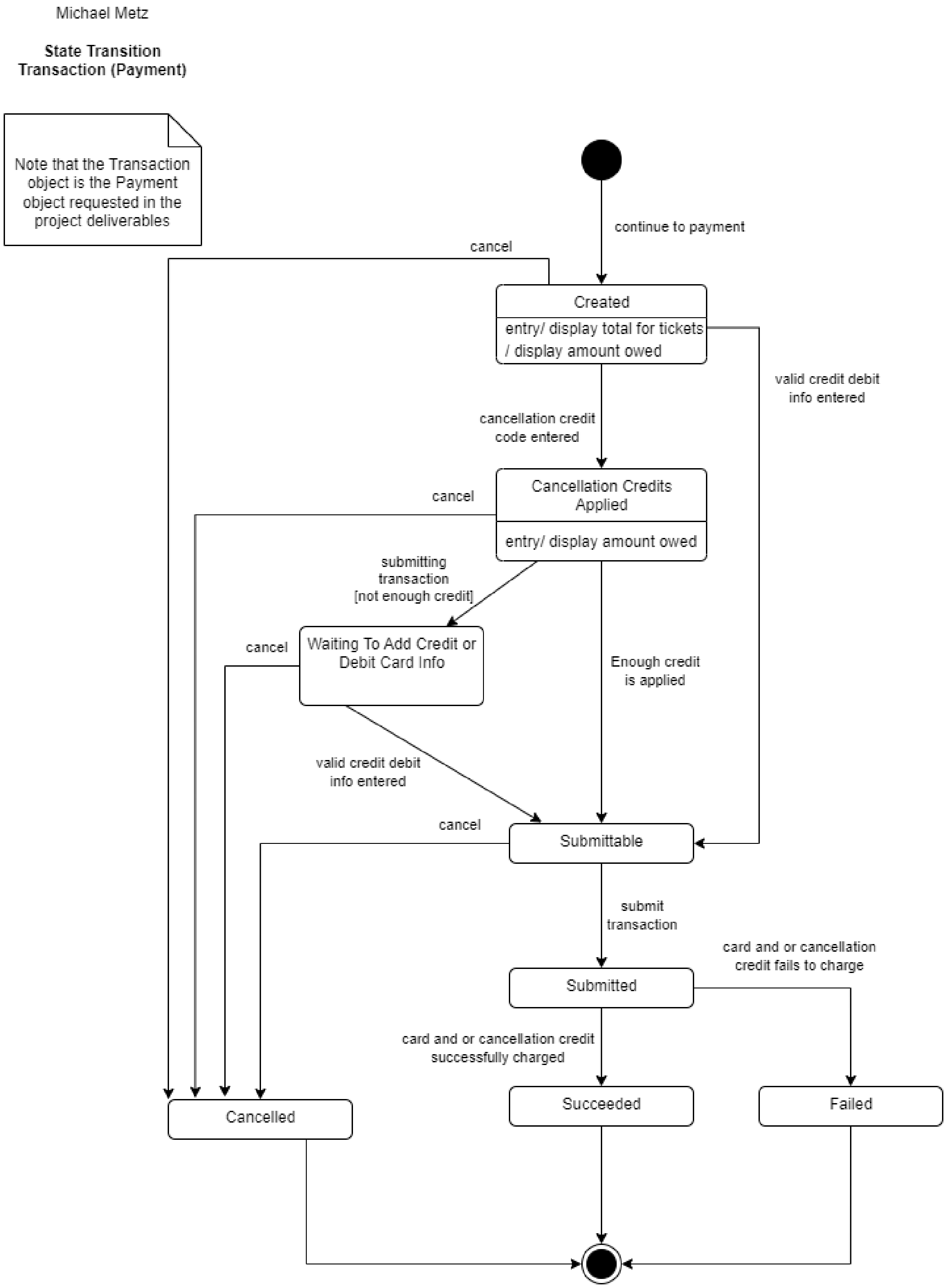
David
Zarinski



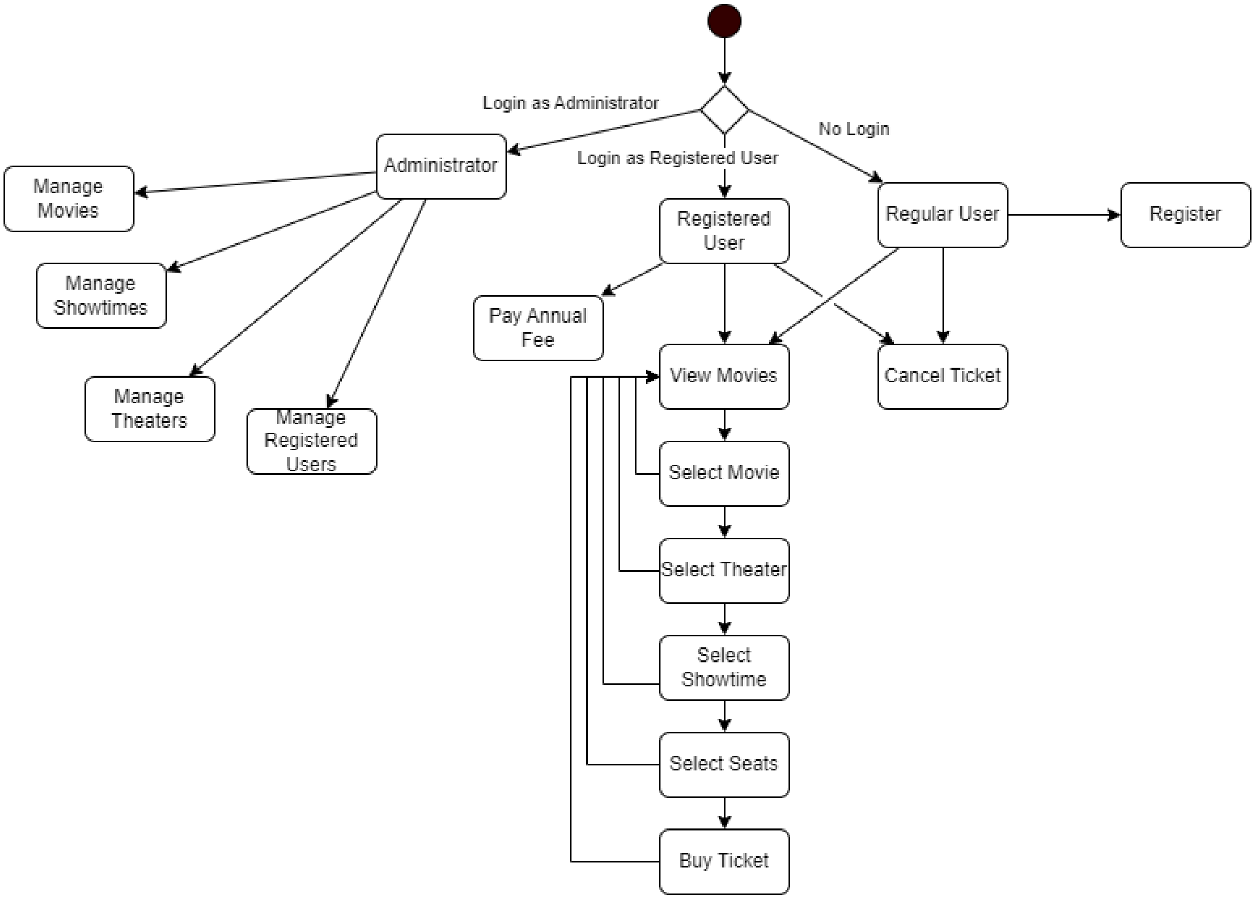
3. Manage Movie Use Case - Kim



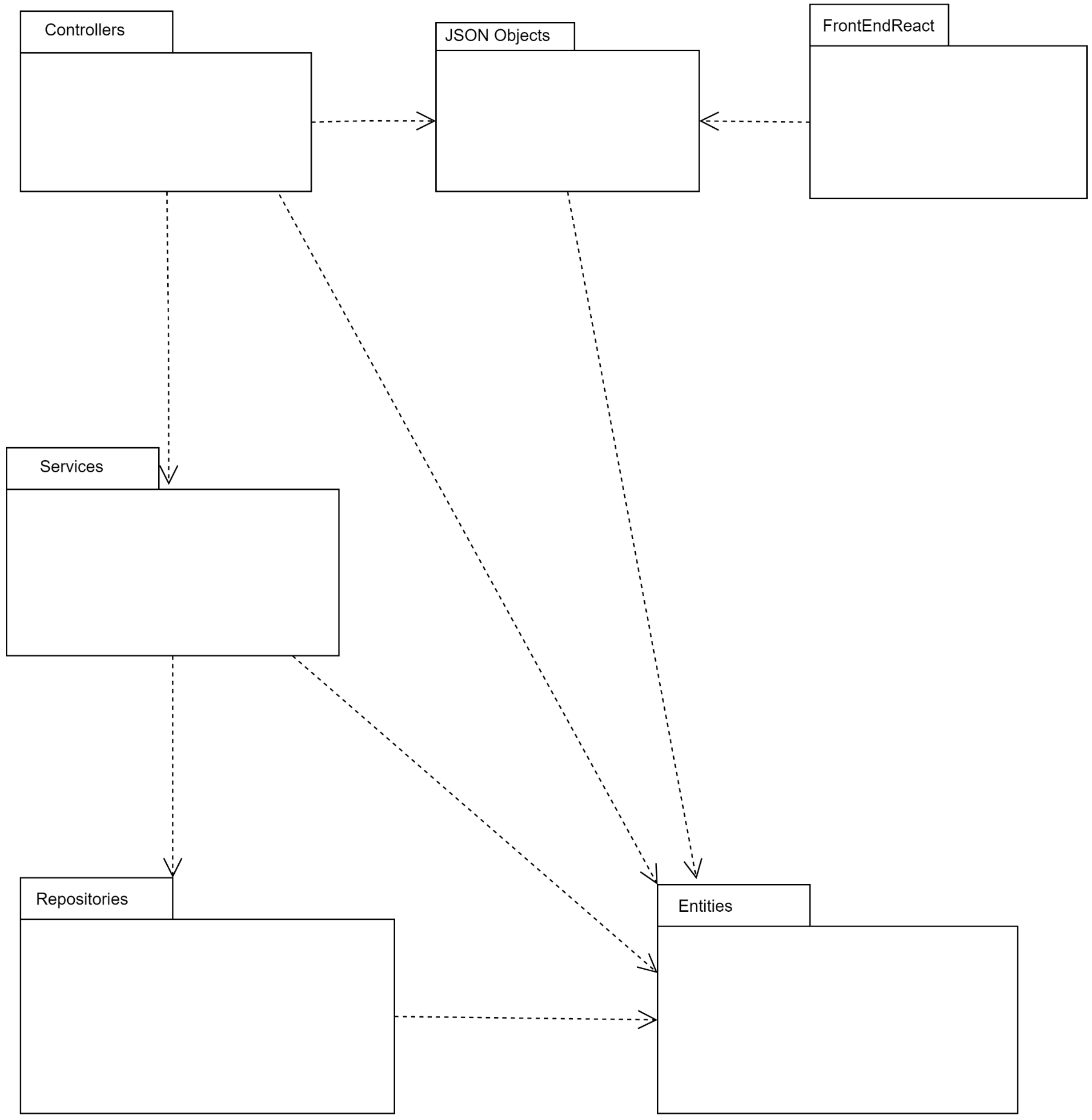
4. Transaction (Payment)


Michael Metz
State Transition
Transaction (Payment)
Note that the Transaction object is the Payment object requested in the project deliverables
continue to payment
cancel
Created
entry/ display total for tickets
/ display amount owed
valid credit debit info entered
cancellation credit code entered
Cancellation Credits Applied
entry/ display amount owed
cancel
submitting transaction [not enough credit]
Waiting To Add Credit or Debit Card Info
cancel
Enough credit is applied
valid credit debit info entered
cancel
Submittable
submit transaction
card and or cancellation credit fails to charge
Submitted
card and or cancellation credit successfully charged
Cancelled
Succeeded
Failed

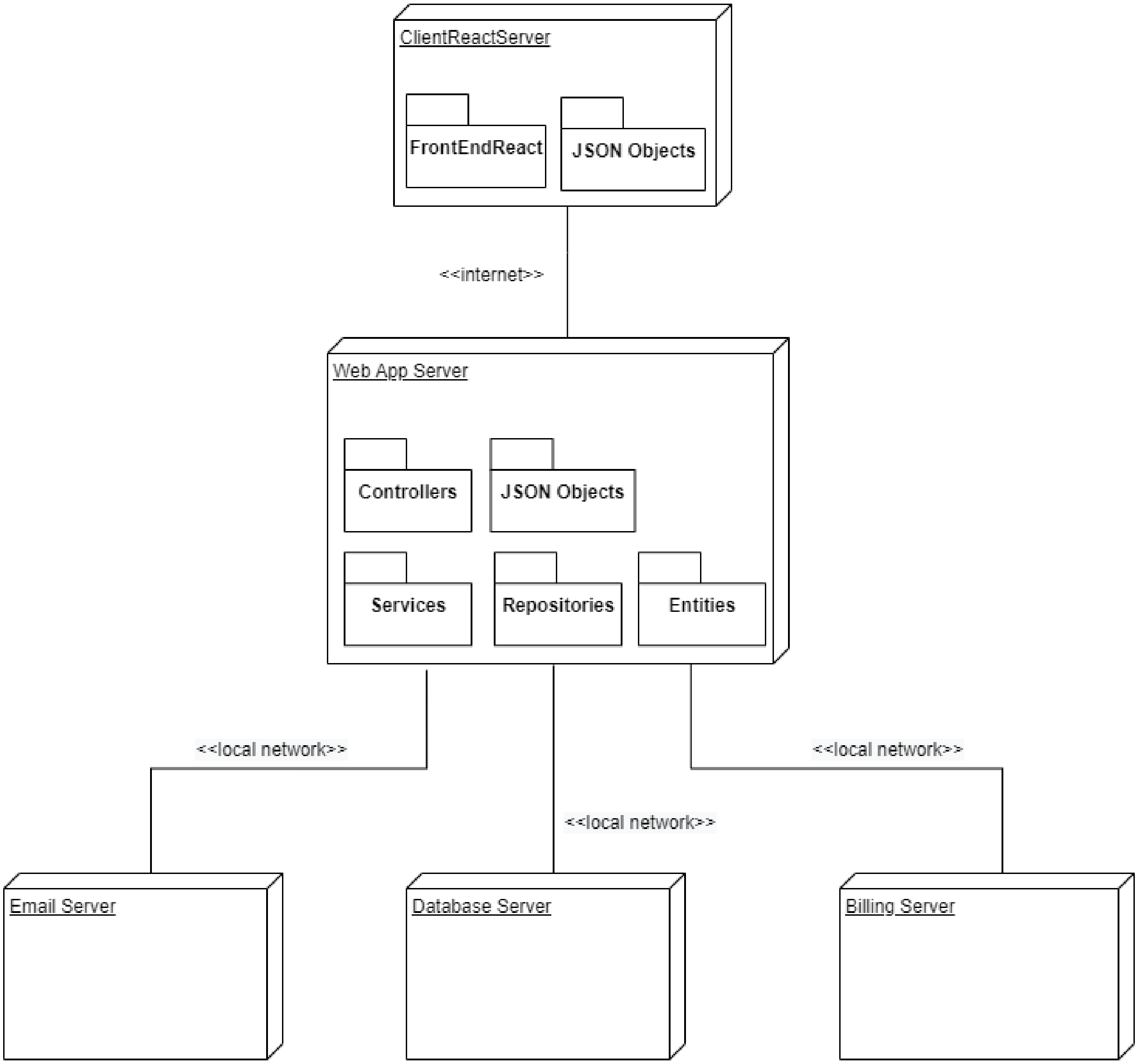
Activity Diagram



Package Diagram



Deployment Diagram


ClientReactServer
FrontEndReact
JSON Objects
<<internet>>
Web App Server
Controllers
JSON Objects
Services
Repositories
Entities
<<local network>>
<<local network>>
<<local network>>
Email Server
Database Server
Billing Server