Software Requirements Specification Template

Software Engineering

The following annotated template shall be used to complete the Software Requirements Specification (SRS) assignment.

Template Usage:
Text contained within angle brackets (‘<’, ‘>’) shall be replaced by your project-specific information and/or details. For example, <Project Name> will be replaced with either ‘Smart Home’ or ‘Sensor Network’.

Italicized text is included to briefly annotate the purpose of each section within this template. This text should not appear in the final version of your submitted SRS.

This cover page is not a part of the final template and should be removed before your SRS is submitted.

<CheepTIX.com>

Software Requirements Specification

<1.0>

<09/14/2023>

<Group 11>
<Kalanikekai Tran>
<Elijah Agustin>
<Isaac Reveles>
<Jonathan Van>

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Author>	<First Revision>
<9/21>	<Version 1>	<Group 11>	<Requirements>

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document. (Note: the following subsection annotates are largely taken from the IEEE Guide to SRS).

1.1 Purpose

What is the purpose of this SRS and the (intended) audience for which it is written.

1.2 Scope

This subsection should:
(1) Identify the software product(s) to be produced by name; for example, Host DBMS, Report Generator, etc
(2) Explain what the software product(s) will, and, if necessary, will not do
(3) Describe the application of the software being specified. As a portion of this, it should:
  (a) Describe all relevant benefits, objectives, and goals as precisely as possible. For example, to say that one goal is to provide effective reporting capabilities is not as good as saying parameter-driven, user-definable reports with a 2 h turnaround and on-line entry of user parameters.
  (b) Be consistent with similar statements in higher-level specifications (for example, the System Requirement Specification) , if they exist.What is the scope of this software product.

1.3 Definitions, Acronyms, and Abbreviations

This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

1.4 References

This subsection should:
(1) Provide a complete list of all documents referenced elsewhere in the SRS, or in a separate, specified document.
(2) Identify each document by title, report number - if applicable - date, and publishing organization.
(3) Specify the sources from which the references can be obtained.
This information may be provided by reference to an appendix or to another document.

1.5 Overview

This subsection should:
(1) Describe what the rest of the SRS contains
(2) Explain how the SRS is organized.

2. General Description

This section of the SRS should describe the general factors that affect 'the product and its requirements. It should be made clear that this section does not state specific requirements; it only makes those requirements easier to understand.

2.1 Product Perspective

This subsection of the SRS puts the product into perspective with other related products or projects. (See the IEEE Guide to SRS for more details).

2.2 Product Functions

This subsection of the SRS should provide a summary of the functions that the software will perform.

2.3 User Characteristics

This subsection of the SRS should describe those general characteristics of the eventual users of the product that will affect the specific requirements. (See the IEEE Guide to SRS for more details).

2.4 General Constraints

This subsection of the SRS should provide a general description of any other items that will limit the developer's options for designing the system. (See the IEEE Guide to SRS for a partial list of possible general constraints).

2.5 Assumptions and Dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

3. Specific Requirements

This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.

Each requirement in this section should be:

- *Correct*
- *Traceable (both forward and backward to prior/future artifacts)*
- *Unambiguous*
- *Verifiable (i.e., testable)*
- *Prioritized (with respect to importance and/or stability)*
- *Complete*
- *Consistent*
- *Uniquely identifiable (usually via numbering like 3.4.5.6)*

Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface for the software will be compatible on Apple, Android and Google smartphones via App Store, Play Store, etc. The software is also compatible with most web browsers like Google Chrome, Safari, Microsoft Edge, and Mozilla.

3.1.2 Hardware Interfaces

The service is a web-based application, so it must require a stable internet connection with a minimum download speed of 10 mbps download and 3 mbps upload, with less than 150 ms latency. Any device capable of this is acceptable; ie. Modem, WAN-LAN, Ethernet

3.1.3 Software Interfaces

1. The ticketing system shall communicate with the Configurator to identify available assets to configure the software

2. The ticketing system shall communicate with the theater directories and databases to reference available movies and/or discounts available

3. The ticketing system shall communicate with the Pearpay system to establish authentic and acceptable payment methods to process them

4. The ticketing system shall log individual user data to record reward/loyalty points and membership perks
    a. email/phone number
    b. location
    c. preferred language
    d. veteran/student/senior status

5. The ticketing system shall direct/inbox the number of our help center provided by our company

6. The ticketing system shall communicate with the Sales department regarding digital transfers and delivery options including any purchase details and inquiries

7. The ticketing system shall communicate with state and city mandated Tax systems to calculate taxes.

3.1.4 Communications Interfaces

The ticketing system shall use HTTPS protocol for web-based interaction.

The ticketing system shall utilize SMS and email communication with information from the user information database.

The user shall communicate with an AI chat-bot to address FAQ, and then direct to customer support line if questions are not within parameters of the preset chat-bot.

The system shall communicate with movie rating databases ie. IMDb and RottenTomatoes to implement movie ratings available.

The system shall communicate with banking institutions to verify authenticity of payments.

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 <Functional Requirement or Feature #1>

3.2.1.1 Introduction

Discussed below are the functional requirements for CheepTIX. The contents are organized into defined categories, with more specific instances being featured in the use-cases. The input, processing, and output sections list numerous items to be covered in the software, in which hopefully a full scope of the application can be ascertained.

3.2.1.2 Inputs

The system shall allow users to select the movie they wish to purchase tickets for.

The system shall allow users to select the showtimes for the desired movie at different theaters.

The system shall allow users to purchase up to 10 tickets per order.

The system shall allow users to select which seats they would like to watch the movie from.

The system shall allow users to select IMAX, 4D, or regular screening.

The system shall allow users to rate and review the movie after viewing.

The system shall allow users to view and claim their rewards and points.

The system shall allow users to opt-in for open-seating for reduced price tickets.

The system shall allow users to exit from ticket viewing before payment.

3.2.1.3 Processing

The system shall process the location of the user and theater nearby their area

The system shall display movie type, IMAX, 4D, or regular screening, and showtimes within the theater

The system shall process the theater layout of available and occupied seats and selected number of tickets

The system shall process transactions, payment method and verification, made from selected movie tickets

3.2.1.4 Outputs

The system shall deliver and make accessible digital tickets via email, text, in-app QR code, or smartphone digital wallets.

The system will dispense receipts for orders via email or phone number.

The system shall send info on tickets purchased via CheepTIX to theaters to allow confirmation of tickets printable at theaters.

The system shall output a feedback survey after the movie's end-time to collect ratings and reviews dependent on movie showtime, ie. next day for evening showing, that evening for morning showing.

The system shall update membership/loyalty points and express the amount viewable under user account information.

3.2.1.5 Error Handling

The system shall prompt for new card details if the card is unverified for transactions.

The system shall offer ticket credit for duplicate tickets erroneously purchased.

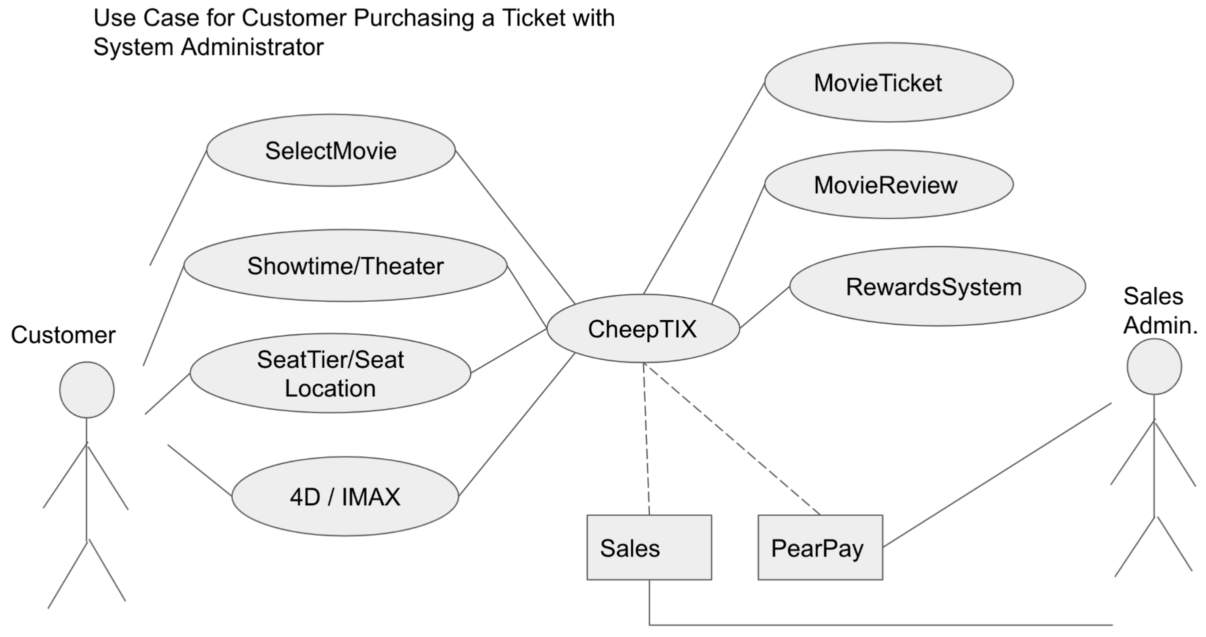
The system shall allow for new ticket delivery in case of incorrect ticket address or dropoff.

3.2.2 <Functional Requirement or Feature #2>

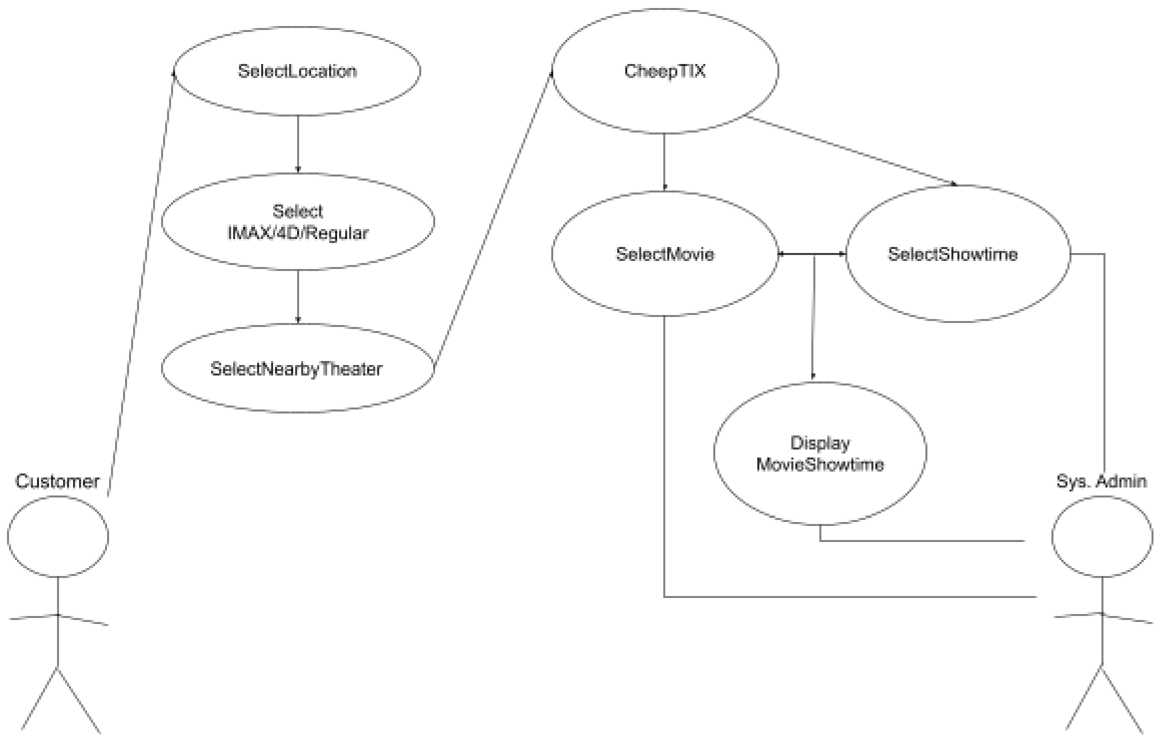
…

3.3 Use Cases

3.3.1 Use Case #1 Use Case for Ticket Purchasing

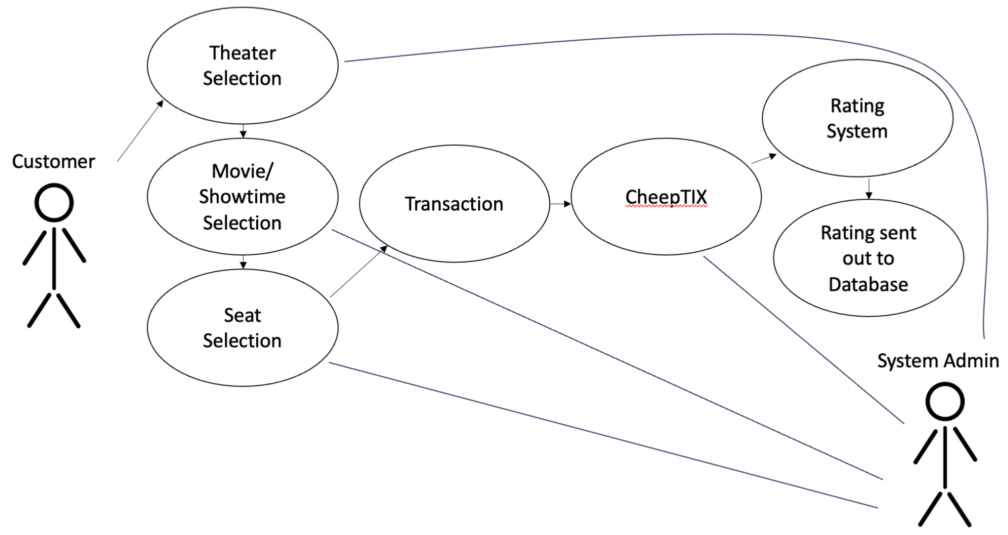


3.3.2 Use Case #2 Viewing Showtimes/ Theater Location



3.3.3 Use Case #3 Rating System

Use Case for the Rating System



3.4 Classes / Objects

3.4.1 <Class / Object #1>

3.4.1.1 Attributes
3.4.1.2 Functions
<Reference to functional requirements and/or use cases>

3.4.2 <Class / Object #2>

…

3.5 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

3.5.1 Performance

The system software must be fully functional on IPv6-only networks (mobile applications).

The system must have an execution time of 1/2 second or 500 milliseconds (ms).

The system must have a throughput of at least 1500 messages per second.

The system must have a minimum storage capacity of 3.5 GB per app.

3.5.2 Reliability

System will dispose of any extraneous data, such as junk or unused, in the system daily to keep performance high

System will run daily scans for potential viruses or unauthorized accesses

3.5.3 Availability

System should be available 24 hours a day and 7 days a week

System is available for purchasing tickets nationally, but only pertains to theaters within San Diego county.

3.5.4 Security

System should be required to keep the server and user information secured via firewall

System should require user logins to be password protected and two-factor authenticated.

System will allow password resets if security questions are verified.

System will message users about attempted logins from suspicious IP addresses.

System will temporarily lock users out of account after five incorrect password attempts.

3.5.5 Maintainability

System will be maintained by Lee Gandhi, of the Sugandeise-Knots Chinese Media Group.

3.5.6 Portability

System should be able to handle cross compatibility between mobile and desktop devices.

System should maintain user info across multiple devices used to access accounts.

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

3.8 Logical Database Requirements

Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.

4.1 Sequence Diagrams

4.3 Data Flow Diagrams (DFD)

4.2 State-Transition Diagrams (STD)

5. Change Management Process

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

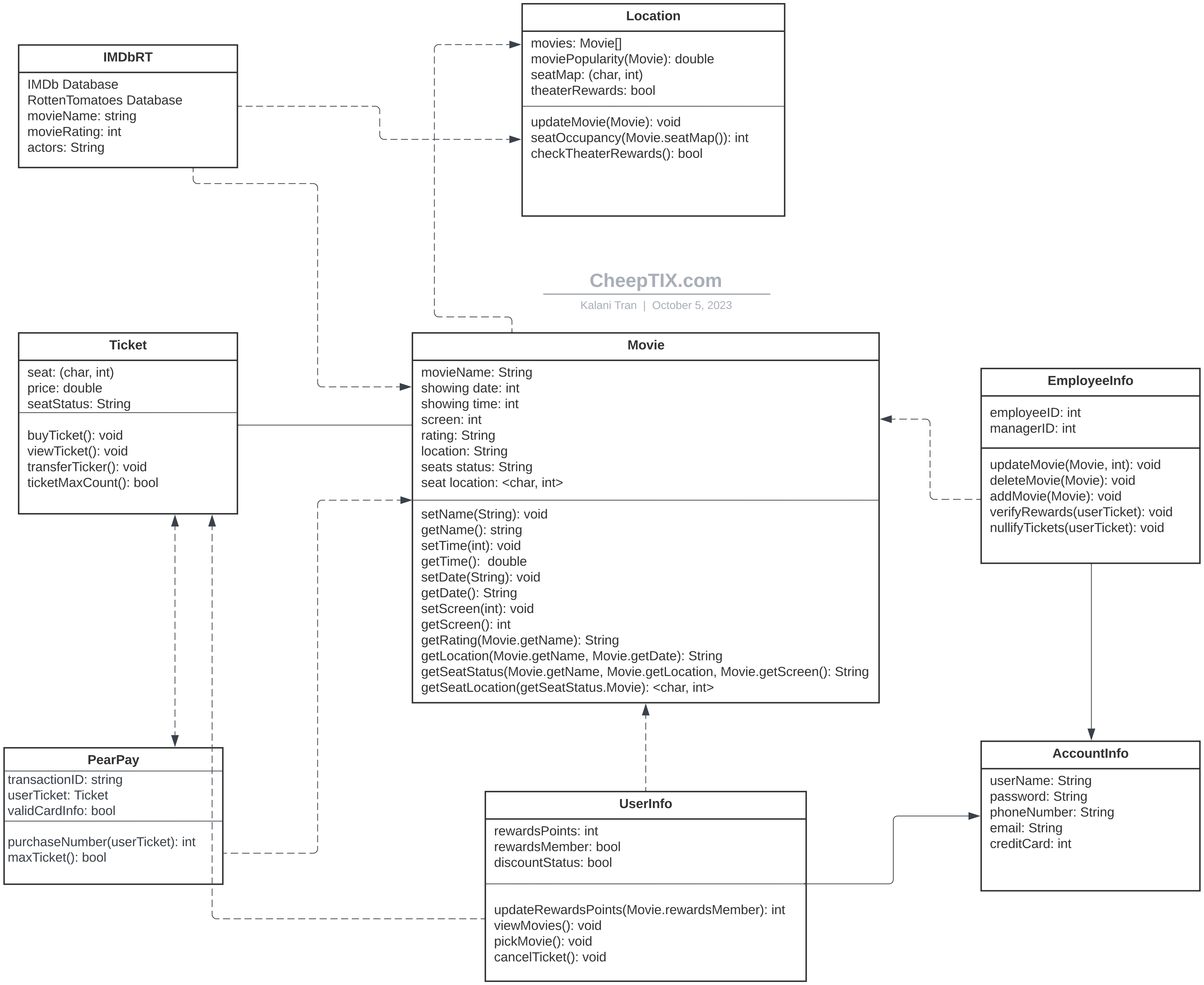
A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix 2


CheepTIX.com
Kalani Tran | October 5, 2023
Location
movies: Movie[]
moviePopularity(Movie): double
seatMap: (char, int)
theaterRewards: bool
updateMovie(Movie): void
seatOccupancy(Movie.seatMap()): int
checkTheaterRewards(): bool
IMDbRT
IMDb Database
RottenTomatoes Database
movieName: string
movieRating: int
actors: String
Ticket
seat: (char, int)
price: double
seatStatus: String
buyTicket(): void
viewTicket(): void
transferTicker(): void
ticketMaxCount(): bool
Movie
movieName: String
showing date: int
showing time: int
screen: int
rating: String
location: String
seats status: String
seat location: <char, int>
setName(String): void
getName(): string
setTime(int): void
getTime(): double
setDate(String): void
getDate(): String
setScreen(int): void
getScreen(): int
getRating(Movie.getName): String
getLocation(Movie.getName, Movie.getDate): String
getSeatStatus(Movie.getName, Movie.getLocation, Movie.getScreen(): String
getSeatLocation(getSeatStatus.Movie): <char, int>
EmployeeInfo
employeeID: int
managerID: int
updateMovie(Movie, int): void
deleteMovie(Movie): void
addMovie(Movie): void
verifyRewards(userTicket): void
nullifyTickets(userTicket): void
PearPay
transactionID: string
userTicket: Ticket
validCardInfo: bool
purchaseNumber(userTicket): int
maxTicket(): bool
UserInfo
rewardsPoints: int
rewardsMember: bool
discountStatus: bool
updateRewardsPoints(Movie.rewardsMember): int
viewMovies(): void
pickMovie(): void
cancelTicket(): void
AccountInfo
userName: String
password: String
phoneNumber: String
email: String
creditCard: int

Movie:

The Movie class holds all data regarding the specific movies showing in theaters, including: movie name, show dates, showtimes, screen numbers, ratings of the movies, what theaters are showing what movies, the seats positions at the theaters, and if there’s open seating in the theaters. This data can be manipulated within the class to set and get any of the aforementioned info.

Ticket:

The Ticket class holds information regarding ticket info, such as the particular seat number indicated by a character and then an integer, ex. K19, the price of the ticket indicated by a double, ex. 12.99, and the status of the seat such as whether it’s an open seat or a reserved purchase seat. This information can be manipulated within the class to buy the ticket which will call PearPay, view the ticket, transfer the ticket, and display whether the max amount of tickets, 10, are bought by one user.

Location:

The Location class holds all relevant data for the location of the theaters registered on CheepTIX.com. The class holds an array of all of the movies at a particular location, displays the ratings of the movies at a particular location, holds a seat map of the theaters at a particular location, and displays whether a theater is signed up for the rewards system compatible with CheepTIX.com. The Location class has functions including updating the movie at a location and checking the current seat occupancy of a location.

AccountInfo:

The AccountInfo class holds information regarding the account of a user, such as passwords, usernames, phone numbers, email addresses, and their credit card information. This class pertains more to logging in and verifying that the account exists within the system, and will also be implemented with security protocols to ensure only authorized accounts can log in to the system.

UserInfo:

The UserInfo class holds information for a user’s account, but differs from the AccountInfo class in that the variation in user profiles are held in this class. This consists of the variable information such as whether a use is a rewards member, how many points they have if they are a rewards member, whether they are a student, senior, or in the military. This data has functions that can

EmployeeInfo:

The EmployeeInfo class holds all the information about employees at CheepTIX.com. This includes whether they are just an employee or have managerial status, and allows employees to update movies, delete movies, add movies, verify a user's rewards status via their sticker, and nullify user tickets if fraud is suspected.

IMDbRT:

The IMDbRT class contains the database information from IMDb and Rotten Tomatoes. These two movie review sites hold valuable information such as the movie's cast list, reviews from critics, synopses, and additional information about the movie. This class has been simplified for easy viewing of the UML and a simple call to the SQL database is contained within the class, but this class would contain its own expanded UML as well.

PearPay:

PearPay is the proprietary payment system for CheepTIX.com. The class contains information such as the transaction ID for a payment, the user's ticket information, and a valid credit/debit card. This information is authenticated to allow the user to purchase their ticket(s), and checks to see if the maximum number of tickets have been purchased.

Test Case ID	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
user_login_1	userName	P0	Verify that a user can login with valid a username and password to their account	App is running, network established	1. Open the app 2. See the prompt for User account and password 3. Enter a valid User login and password 4. Press login	User will be able to login to the account	User logs in to the account	Pass	K.T.
user_login_2	userName	P0	Verify that a user cannot login with an invalid username and password to a specified account	App is running, network established	1. Open the app 2. See the prompt for User account and password 3. Enter an invalid User login and password 4. Press login	User will not be able to login to the account	User cannot login to the account	Pass	K.T.
staff_login_1	employeeID	P0	Verify that an employee can login with a valid username and password to their account	App is running, network established	1. Open the app 2. See the prompt for Employee account and password 3. Enter a valid Employee login and password 4. Press login	Employee will be able to login to the account	Employee logs in to the account	Pass	E.A.
staff_login_2	employeeID	P0	Verify that an employee cannot login with an invalid username and password to a specified account	App is running, network established	1. Open the app 2. See the prompt for Employee account and password 3. Enter an invalid Employee login and password 4. Press login	Employee will not be able will be able to login to the account	Employee cannot log in to the account	Pass	E.A.
pw_reset	password	P1	Verify that a user or employee can be prompted to reset their password if requested and confirmed via email	App is running, network established	1. Open the app 2. See the prompt for User/Employee account and password 3. See the "Forgot Password?" prompt below 4. Click it 5. Enter User/Employee account prompted 6. Click Confirm 7. Check for email from CheepTix 8. Click link in email 9. Confirm password reset on app	User should be able to click a prompt to reset their password and also be able to confirm password reset via email	User was able to click a prompt to reset their password and also able to confirm password reset via email	Pass	I.R.
movie_showtime	movies	P1	Verify that a user will be able to see displayed on the app movie showtimes	App is running, network established, login successful	1. Click "movies" 2. Check showtimes	User should see movie showtimes displayed on the app	User could see movie showtimes displayed on the app	Pass	I.R.
movie_seats	seat	P1	Verify that a user will be able to see displayed on the app available and unavailable seats in a theatre	App is running, network established, login successful	1. Check movie 2. Select theater location 3. Check theater number 4. See available/ unavailable seats	User should see available and unavailable seats in a theatre displayed on the app	User could see available and unavailable seats in a theatre displayed on the app	Pass	I.R.
movie_review	rating	P2	Verify that a user will be able to view a movie's reviews when selecting it on the app	App is running, network established, login successful	1. Check movie 2. Check review	User should be able to see a movie's review when selecting it on the app	User is able to see a movie's review when selecting it on the app	Pass	J.V.
payment_pearpay	transactionID	P0	Verify successful payment transaction when utilizing PearPay	App is running, network established, login successful	1. Select movie 2. Select showtime 3. Select theater 4. Select available seat 5. Process tickets 6. Confirm Tickets	User should have a smooth transaction when utilizing PearPay on the app	User is able to have a smooth transaction when utilizing PearPay on the app	Pass	J.V.
survey_review	rating	P2	Verify that a survey will be sent to the user a day after the showing of the movie they watched	App is running, network established, login successful	1. Send notification to user 2. Notification sent the next day after the movie.	User should receive a movie survey on the app a day after the movie they watched	User received a movie survey on the app a day after the movie they watched	Pass	J.V.

Verification Test Plan:
This section lays out test plans for verification. In addition to the test itself, you will need to clearly explain your test. Be as detailed as possible, identifying what features of your design you are testing, what the test sets/vectors are, and how your selected test(s) cover the targeted feature(s). (Hint: use or modify your design diagram to indicate the target and scope of the tests, what kind of failures you are covering, etc.). Feel free to use any methods discussed in class.

Test cases have been outlined to follow the general chronology of using the app, in which various circumstances are being covered within the breadth of the scenarios. Beginning with login, viewing movie information, purchasing tickets, and the post-viewing instances that occur through the app, the test cases cover the significant features of the app, denoted by the high occurrence of P0 and P1 priority cases. Our team of testers will consist of:

1. Kalanikekai Tran
2. Elijah Agustin
3. Isaac Reveles
4. Jonathan Van

We will be adhering to both black and white box testing. The test cases where we are inputting data and analyzing the output will conform to black box testing, as we are only monitoring the expected outcome and not how the function operated, while for all other test cases we can utilize white box testing as we will want to distinguish what within that test cases' function is not performing as it should. We will test for branch, condition, and statement coverage, as our functions involve for loops, if statements, and bool statements.

I. **Check user login with valid data** (Black box)
    A. Partition testing with valid user login details. Testing that when a user logins with valid information, they will be able to access their account. A valid password and username are required for this test. Testing with valid info will confirm whether users are able to access their accounts, and that the app has saved and logged this user's login information. Username cannot contain any spaces or special characters. Password must be at least 12 characters long, and contain an uppercase letter, a lowercase letter, a number, and a special character, ie. ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/
        1. Ex. Username: jvan0049
        2. Ex. Password: c0m3T0class!
    B. Variables

        1. userName: String
        2. Password: String

II. **Check user login with invalid data** (Black box)

  A. Partition testing with invalid user login details. Testing that when a user logins with invalid information, they will not be able to access a user account. An invalid password and username are required for this test. Testing with invalid info will confirm that invalid login information will not yield access to any accounts, and that the app has not saved and logged this login information as authenticated credentials. Username cannot contain any spaces or special characters. Password must be at least 12 characters long, and contain an uppercase letter, a lowercase letter, a number, and a special character, ie. ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/
    1. Ex. Username: kalani tran
    2. Ex. Password: password11

  B. Variables
    1. userName: String
    2. Password: String

III. **Check employee login with valid credentials** (Black box)

  A. Partition testing with valid employee login details. Testing that when an employee logins with valid information, they will be able to access their employee account. A valid password and username are required for this test. Testing with valid info will confirm that invalid info will not yield access to any employee accounts, and that the app has not saved and logged this employee's login information as authenticated credentials. Username is their employee or manager ID. Password must be at least 12 characters long, and contain an uppercase letter, a lowercase letter, a number, and a special character, ie. ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/
    1. Ex. Username: 131026347
    2. Ex. Password: Eye<3m0V13$

  B. Variables
    1. employeeID: int
    2. managerID: int
    3. Password: String

IV. **Check employee login with invalid credentials** (Black box)

  A. Partition testing with invalid employee login details. Testing that when an employee logins with invalid information, they will not be able to access any employee accounts. An invalid password and username are required for this test. Testing with invalid info will confirm whether employees are able to access their accounts, and that the app has saved and logged this employee's login information. Username is their employee or manager ID. Password must be at least 12 characters long, and contain an uppercase letter, a lowercase letter, a number, and a special character, ie. ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/

    1. Ex. Username: moe lester
    2. Ex. Password: 123456789

B. Variables
    1. employeeID: int
    2. managerID: int
    3. Password: String

V. **Prompt for password reset for user accounts** (Black box)

A. Testing the process of user selecting "Forgot Password?", in which app will prompt users to input their account username or employee ID registered with their account, in which an email will be sent to them with a link to where they will be redirected to enter in a new acceptable password. The tester selecting the "Forgot Password?" option is requisite when assessing this facet. Testing whether the reset password option is viable is essential to ensuring users are still able to access their accounts in the scenario they forget their passwords, have their passwords compromised, etc. Password must be at least 12 characters long, and contain an uppercase letter, a lowercase letter, a number, and a special character, ie. ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/

B. Variables
    1. Password: String
    2. email: String

VI. **Movie time and date are displayed on the app** (White box)

A. Testing whether users can view the movie times and dates of the currently showing and prospective movies while on the app. This feature is essential so that users are aware of and can browse the available movie showtimes to purchase tickets for. Not having the entire catalog will result in dissatisfied customers; wherein they will miss out on movies they want to watch, or not even have the option to watch a certain movie if it's not included. App must display both dates and showtimes of movies at select theaters.

B. Variables
    1. movieName: String

2. showingDate: int
3. showingTime: int
4. screen: int
5. location: String

C. Functions
1. setName(String): void
2. getName(): String
3. setTime(int): void
4. getTime(): double
5. setDate(String): void
6. getDate(): String
7. setScreen(int): void
8. getScreen(): int
9. getLocation(Movie.getName, Movie.getName): String

VII. **Movie theater seat maps are displayed on the app** (White box)

A. Testing whether users can view the seat maps of the currently showing and prospective movies while on the app, including both available and unavailable seats. This feature is essential so that users are aware of and can browse the available seats to purchase tickets for. Users will also be able to view the tier of the seats, such as open-seating vs. premium seating. Not having the entire catalog will result in dissatisfied customers; wherein they will miss out on seats they want to watch from.

B. Variables
1. seatStatus: String
2. seatLocation: <char, int>
3. seatMap: (char, int)

C. Functions
1. getSeatStatus(Movie.getName, Movie.getLocation, Movie.getScreen): String
2. getSeatLocation(getSeatStatus.Movie): <char, int>
3. seatOccupancy(Movie.seatMap()): int

VIII. **IMDb/Rotten Tomatoes reviews are viewable when viewing movies** (White box)

A. Testing whether the IMDb and Rotten Tomatoes scores and reviews are displayed on the app when viewing movies. Must reference the IMDbRT class which holds the databases for both movie critique sites. Testing is essential as including movie reviews and ratings makes for a comprehensive ticket purchasing app, wherein users can make more informed decisions if they are able to view extraneous information.

B. Variables:
   1. IMDb Database
   2. RottenTomatoes Database

C. Functions:
   1. rating: String
   2. getRating(Movie.getName): String
   3. movieName: string
   4. movieRating: int
   5. actors: String

IX. **Successful PearPay transaction with valid payment info** (Black box)

A. Testing whether users can make a successful PearPay transaction when purchasing a movie showtime on the app. This process is essential to make sure that movie transactions can utilize PearPay. Not being able to use PearPay will obstruct any transactions made on the app and will not process any orders made. The app must be able to utilize PearPay to make any type of transaction.

B. Variables
   1. transactionID: String
   2. userTicket: Ticket
   3. validCardInfo: bool

C. Functions:
   1. purchaseNumber(userTicket): int
   2. maxTicket(): bool

X. **Successful implementation of movie review survey the day after the showing** (White box)

A. Testing how the app displays notification to users about movie reviews. This is essential to the app because it gives users the ability to review the movie they watched with functional processes such as commented reviews as well as numbered reviews. Testing is required in order to enable seamless notifications and to ensure the functionality of the review page.

B. Variables:
   1. Movie_review : String
   2. Survey_review : String

C. Functions:
   1. getRating(Movie.getName): String

2. movieName: string
3. movieRating: int