M.E.S.S.

Movie Entertainment Software System

Final Report

11.29.2016

Group #9 - JEKKL
Kris Hermstad - Team Lead, Application Developer
Kirtan Patel - Application Developer, Technical Writer
Emily Horton - UI Developer, Client
Lindsay Stone - UI Developer, User
Jack Martin - Database Developer/Admin

Software Engineering - Tuesday & Thursday - 5:30pm to 7:15pm - FALL 2016

Table of Contents:

Introduction:

<u>Product Introduction:</u>

M.E.S.S: Movie Entertainment Software System

M.E.S.S is an OS independent Javabased POS (point of sale) application designed for the box office of a movie theater or venue. The application features a GUI that will interface with the venue's ticket database. M.E.S.S. will facilitate the sales, refunds and monitoring of tickets. Users will be able to provide refunds to customers while maintaining a valid state for the ticket database. M.E.S.S. will ensure the venue managers and employees have a simple userfriendly interface that will allow for quick transactions during rush business hours.

 ** StretchGoal: Implement A Concession Sales Functionality

Application Layer: Java
 UI: JavaFX
 Database: MySQL

M.E.S.S.:

Deliverable #2 (Requirements)

09.18.2016

Team JEKKL: Group 9
Kris Hermstad (Team Lead)
Kirtan Patel (Application Layer/Technical Writer)
Emily Horton (UI Development/Client)
Lindsay Stone (UI Development/User)
Jack Martin (Database Design/Product Support)

CONTAINS:

Note: Rationale for Project and Requirements is addressed within the Problem Statement.

<u>**M.E.S.S: Movie Entertainment Software System**</u>
<u>**by The JEKKL Software Development Team**</u>

PROBLEM STATEMENT

INTRODUCTION

Every year, various industry Point of Sale technologies improve in quality at an exponential rate, yet the average commercial Point of Sale system for a regional movie theater remains stuck in the last decade. Theaters are burdened with aging, unreliable and cumbersome legacy hardware to support old, outdated and buggy software. With the cost of hardware decreasing, the opportunity for a business to upgrade its POS systems and software becomes more likely.

PROPOSAL

1Our client has tasked us with designing a new POS system for their box office kiosks. The JEKKL Software Development Team proposes the Movie Entertainment Software System (M.E.S.S.), a system that shall provide a secure, open-source POS software solution that is not beholden to overly demanding hardware requirements or dependencies.

2The system should be OS independent. Our client has a variety of machines, from MacBooks to Dell Desktops, and they need the ability to access their POS software using any of these machines. To achieve platform independence, the entire M.E.S.S. shall be written in Java. This will allow any computer with the Java Virtual Machine to have access to M.E.S.S.

3The client will need a database solution that can accurately model the state of their venue. We shall use a cloud-based MySQL Database to hold the theaters relevant information (Auditoriums, Movies, Showtimes, Employees, etc). With the database being decentralized, the theater administration will not have to maintain an on-site database. In the event of a power failure, the theater's most pertinent data will be secure, safe in a cloud-based MySQL server.

4The client needs the ability to log into their specific MySQL server and its database. As long as the target database matches the schema requirements, M.E.S.S. has no limit on how many MySQL databases the user connects to it's interface. For many M.E.S.S. users, they will be managing multiple theaters, distributed over various MySQL databases. To satisfy this need, upon initial load, M.E.S.S. shall present the user with a Theater Database Log-In interface. From here, they can access any database they have privileged access to.

5Upon authorized access to the correct database, the client has requested there be an employee log-in interface. The employee functionality will ensure all transactions are logged with an employee ID. To achieve this, each theater's unique database shall contain an "Employee" Entity, each with their own unique id.

6Similar to the Employee ID, the client has requested the ability to grant certain employees "authorized" privileges. This will be accomplished by adding a Manager flag as an attribute to the employee table. Any employee with the Manager attribute shall have access to all of M.E.S.S.'s administrative capabilities.

7As a matter of security, the client has requested that upon an employee logging in, if their screen remains untouched for a certain period of time (yet to be determined), the employee shall automatically be logged out and the screen will go back to the employee log-in. This will ensure that in the event that an employee needs to leave their kiosk unattended at a moment's notice, the system will enact a self-lock out.

8 After a user has logged-in as an employee, they shall be presented with the main box office User Interface. At this point a new "session" is created to keep track of all transcactions created by a certain employee while logged in. The client has asked that a high priority be placed on usability. The system should log-in very swiftly. There should be no noticeable delay when changing between screens and activities while using M.E.S.S. The client frequently has high-volume influxes of transactions which require a responsive and efficient POS system. M.E.S.S. will aim to highly optimize all performance critical functionality without neglecting the necessary functional requirements.

9The main functionality of M.E.S.S. will be the vending of tickets that correspond to the various movies/shows the client's are conducting in that period of time. The client has requested a system whose database will properly reflect the current state of their theater. To accomplish this, a database schema shall be developed with accurately captures the modeling of the auditoriums and their various showtimes. This will allow for the user to have knowledge of their box office sales down to specific showtimes.

10The client needs for a way to keep track of their daily sales for each showtime, in extension allowing them to see total sales based on their required constraints. (Sales for the day, week, month, etc). To achieve this, a separate "sales" component shall be modeled in the theaters database. Each transaction will modify the state of the sales component in the database, providing an accurate up-to-date summary of the client's sales.

11 With each transaction, the user should be able to add, and remove tickets from a currently active transaction window. They should also have the ability to cancel a transaction at any time, creating a new, empty transaction. If at anytime a transaction is canceled, the system should ensure no updates are made to the ticket database. The employee shall also be able to process a transaction, completing the transaction. After a transaction is processed, the system shall be able to offer refunds to all previously entered transactions in the system.

12 As with the log in functionality for the database and employee, there should also be the ability to log-out from the database, and the employee. If a currently logged in employee logs out from a session, they should be returned to employee log-in screen. From employee log-in screen, the user should be able to log-out of the Theater database they are currently logged into.

13 Beyond all of the functional requirements above, the client has put a high priority on developing a newer, modern looking UI. The client's previous POS software have all been very simple, bland beveled grey GUIs. The client's kiosks are often customer facing, so they want to ensure their presentation is up-to-date and presentable to the public. The UI shall be very user friendly and easy to read, above any other requirements. The client has put an emphasis on speed over presentation, but the GUI presentation is still an incredibly important component to M.E.S.S.

M.E.S.S to be delivered to client on November 29th, 2016. (as of 9/16/2016)

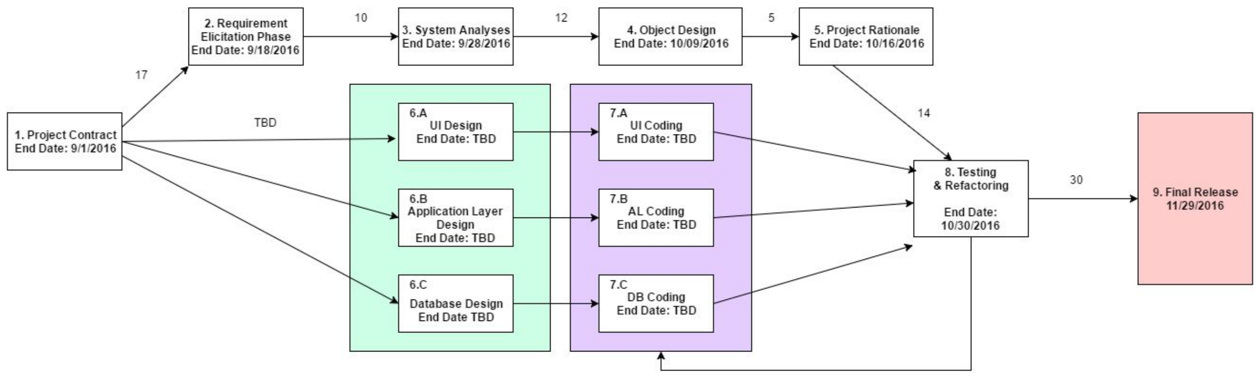
By:
Kris Hermstad
Jekkl Team Lead

M.E.S.S. - Requirements Trace Matrix (RTM)	- as of 9/18/2016		
Entry #	Paragraph #	RTM Entry	Entry Type
1	3	M.E.S.S. shall be OS independent, working on any machine with the JVM. M.E.S.S. shall be written completely in Java.	SW
2	4	M.E.S.S. shall connect to a cloud-based MySQL DB that adheres to required schema.	SW
3	5	M.E.S.S shall presente User with a Theater Database Log-In Interface	SW
4	6	Upon logging into Theater DB Log-in Interface, User is presented with Employee Log-in.	SW
5	7	Employees shall have a "isManager" attribute, providing user with authorized interactions	SW
6	8	M.E.S.S. shall have a time-out feature for when the User has not used the Application after a certain duration of time.	SW

7	**10**	**M.E.S.S. shall vend tickets while maintaining proper state with its MySQL Database**	**SW**
8	**10**	**M.E.S.S. shall accurately reflect current state of each unique showtime currently within the theaters DB.**	**SW**
9	**10**	**M.E.S.S. shall provide user with the ability to monitor sales states.**	**SW**
10	**9**	**All transactions and log-in interface interactions should be very swift. No noticeable delay when interacting with UI.**	**SWC**
11	**11**	**M.E.S.S. shall have a uniform, stylized and modernized UI**	**NTH**

M.E.S.S. Gantt for 9/18/2016					
Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?

Kris Hermstad (Team Lead)	Draft Problem Statement, RTM, and Gantt	3	9/15	9/18	Yes
Jack Martin	Database E-R Diagram	10	9/8	9/18	Yes
Emily Horton	Being drafting UI design mockups	16	9/2	9/18	Yes
Lindsay Stone	Draft UI mockups, experiment with JavaFX Scene Builder, create PERT chart	3	9/2	9/18	Yes
Kirtan Patel	Forumlate Documentation Standard for all project documents	16	9/2	9/18	Yes
Next Deliverable:	Due:				
System Analysis	9/28/2016				



M.E.S.S. PERT

9/18/2016

1. Project Contract
2. Requirements Elicitation
3. Systems Analyses
4. Object Design
5. Projection Rationale
6. Design Stage
7. Coding Stage
8. Test and Refactoring
9. Final Release

<u>Movie Entertainment Software System</u>
<u>Project Terminology:</u>

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

As of 9/18/2016

M.E.S.S. Deliverable #3

System Analysis and Design

Software Engineering: Tuesday/Thursday 5:30pm w/ Mr. Rao

Due: 10.02.2016

TEAM JEKKL (GROUP #9)

Kris Hermstad: Team Lead, Database Developer

Emily Horton: Client, UI Developer

Kirtan Patel: Technical Writer, Application Developer

Lindsay Stone: User, UI Developer

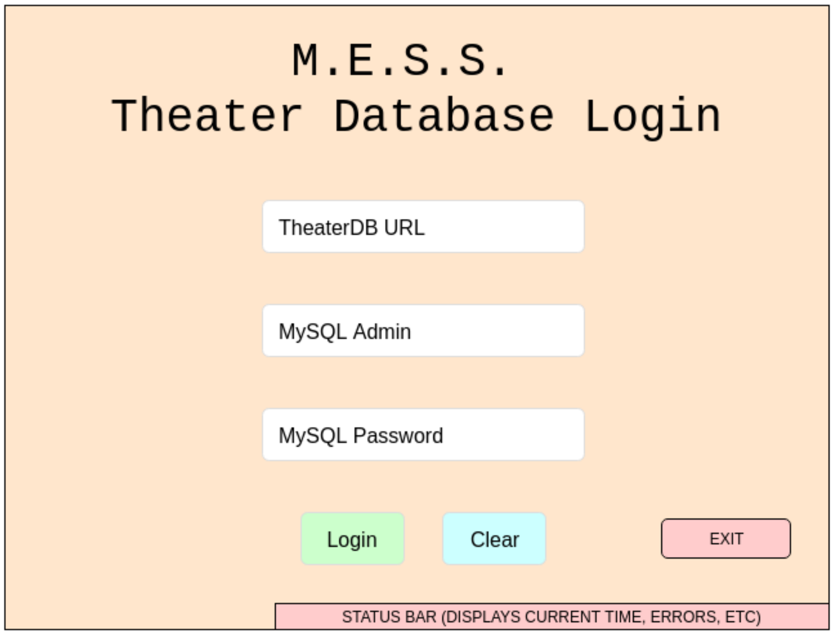
Jack Martin: Product Support, Database Developer

Contains:

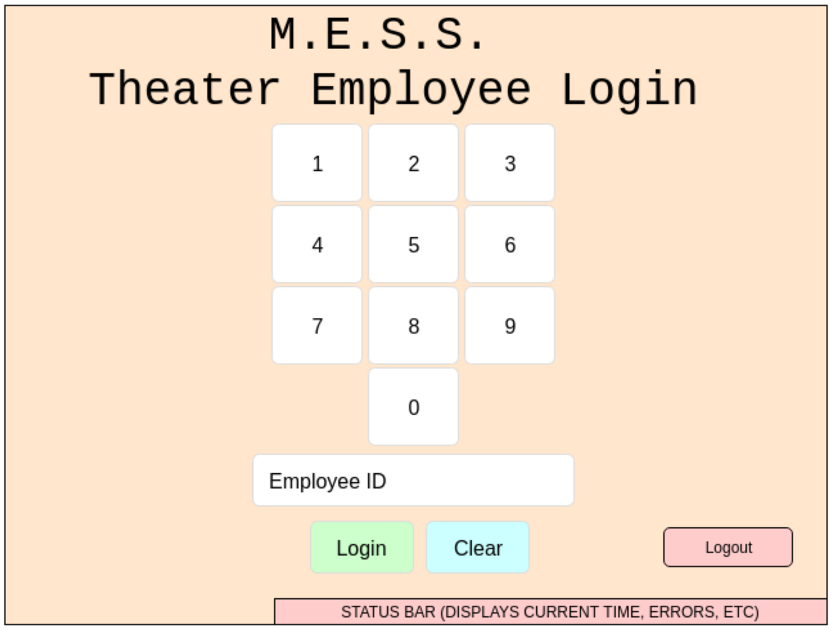
M.E.S.S. Horizontal Prototype

10/2/2016

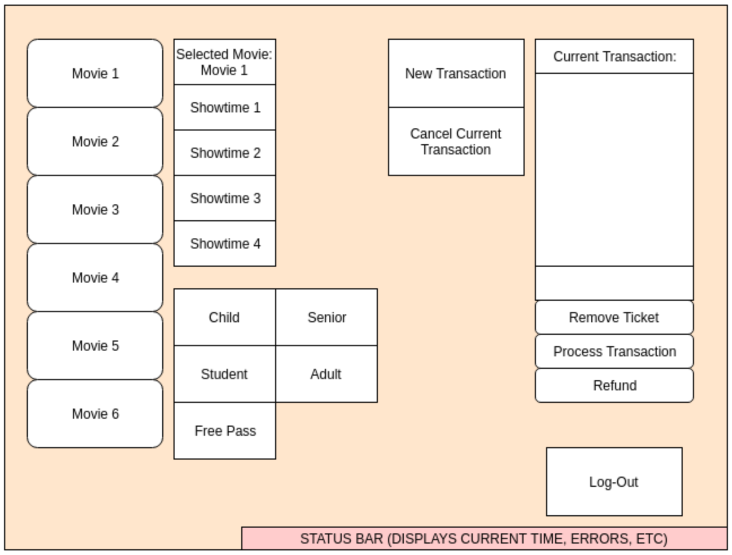
1. Login to Theater MySQL Database



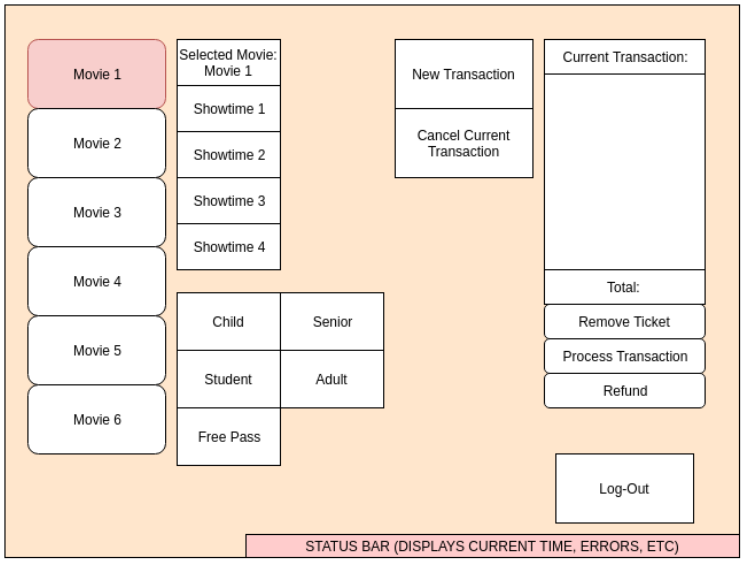
2. Login as Employee



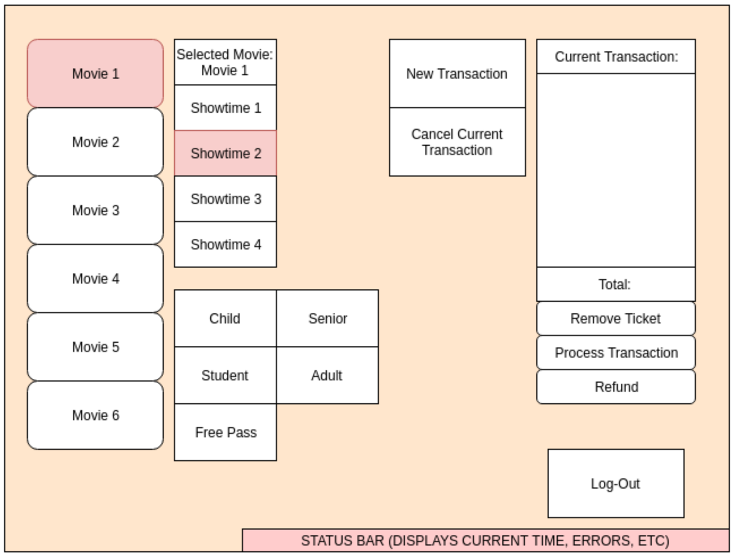
3. Main GUI Displayed



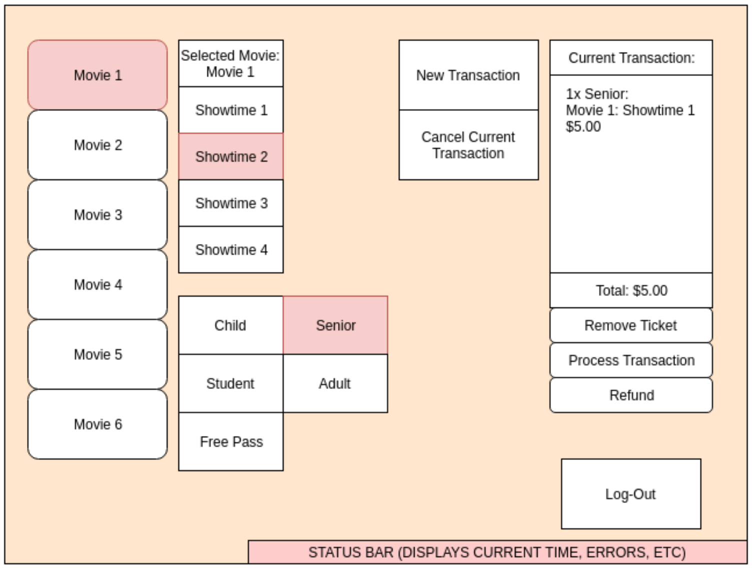
4. Select Movie



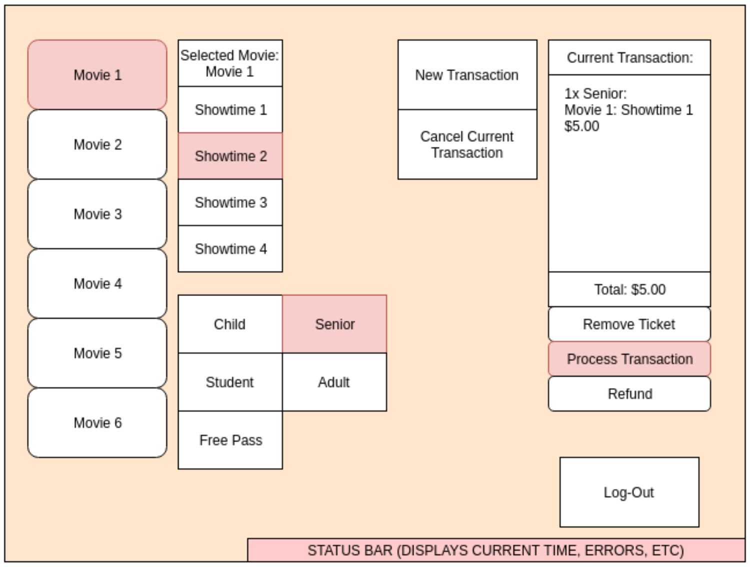
5. Select Showtime, after selecting Movie



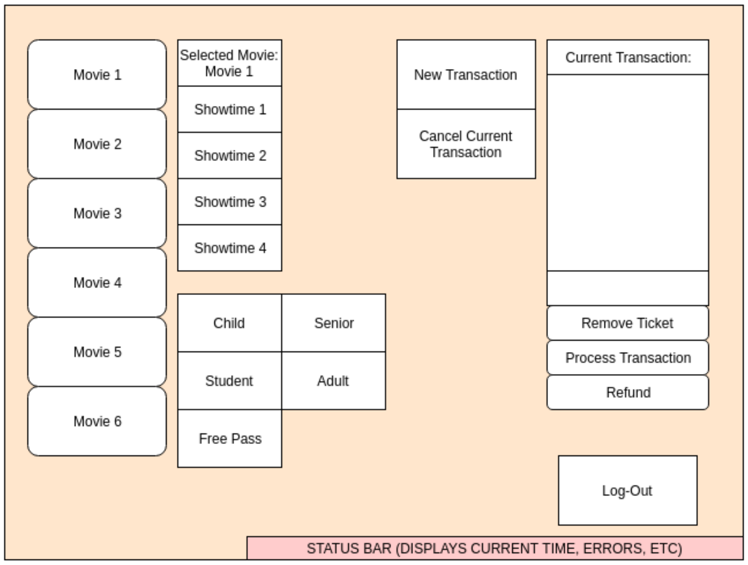
6. Select Ticket Type



7. Process Transaction



8. After a transaction is processed, a new empty transaction is created.



Requirements Trace Matrix: (as of 10/2/2016)

See Use Case Index to find corresponding Use Case for each Entry.

Entry #	Paragraph #	RTM Entry	Entry Type	Use Case
1	1	Shall be OS independent, working on any machine with the JVM.	SW	01
2	3	Shall have a Theater Login Interface that connects to a MySQL db via the internet	SW/HW	02
3	5	Shall have an employee Login interface that verifies an employee has access to the Theaters DB	SW	03
4	8	A new session shall be created when an employee successfully logs in.	SW	04
5	8	There shall be a MainUI from where all ticket vending occurs	SW/HW	05
6	9	There shall be a Currently Active Transaction created upon which all tickets will be added to.	SW	06
7	11	Shall be able to add ticket(s) to Currently Active Transaction	SW	07
8	11	Shall be able to remove any tickets added to Currently Active Transaction.	SW	08
9	11	Shall be able to Cancel any transaction currently active.	SW	09
10	11	Shall have the functionality to process a Currently Active Transaction	SW	10
11	11	Shall be able to offer a “refund” for any transaction in the Theater’s Transaction Database.	SW	11
12	7	Shall have time-out (lock) an employee’s session if they have not interacted with system after a certain specified period of time.	SW	12
13	12	Shall have a log-out functionality for a currently logged in employee.	SW	13
14	12	Shall have a log-out functionality for a currently logged in Theater.	SW	14
15	13	Shall have an Exit Application functionality.	SW	15
16	13	Updated Modern UI	NTH	
17	13	Fast, instantaneous transactions	NF	

M.E.S.S. USE CASES INDEX:

* ALL UNDERLINED EVENTS ARE SYSTEM EVENTS

USE CASE:	**UC_01_OpenApplication**
Actors/Participants:	Employee
Flow of Events	1. Employee starts M.E.S.S. executable via shortcut or command line 2. M.E.S.S. begins running, displaying the initial interface, Theater Log-In (UC_002).
Entry Condition	M.E.S.S. is loaded on a machine that satisfies the necessary system requirements. System running M.E.S.S. must have: 1. Up-to-date Java Virtual Machine 2. Internet Connection to Connect to Remote Database, or an ethernet connection to a database on a local server.
Exit Condition	When program has successfully loaded to the opening Theater Log-In screen, the exit condition is achieved.
Quality Constraints	The application should open, with the first log-in screen displayed within 3 seconds.

USE CASE:	**UC_02_TheaterLogin**
Actors/Participants:	Employee, Database
Flow of Events	1. User enters unique Theater Database URL into DatabaseURL field. 2. User enters Theater Database Admin ID into Admin ID field. 3. User enters Theater Database Admin's password into Admin Password field. 4. User clicks Submit Button 5. Database login data is submitted and evaluated. If login is successful, the exit condition has been reached. The application proceeds to display the next screen, initiating the next use case.
Entry Condition	System has reached exit condition for UC_01_OpenApplication. Program has successfully opened.
Exit Condition	The system has been granted access to Theater Database.
Quality Constraints	The connection should reflect the speed of the user's connection. If the user is making a request to a remote database hosted by a poor ISP, the connection will reflect this.

USE CASE:	**UC_03_EmployeeLogin**
Actors/Participants:	Employee, Database
Flow of Events	1. Employee selects “Employee ID” field to enter their employee ID. <u>2. Displays Numeric P{ad to enter User’s 3-digit employee ID.</u> 3. Employee enters ID via Numeric Pad. 4. Employee presses Submit button. <u>5. Checks submitted EmployeeID against Theater DB’s Employee Table.</u> <u>6. IF Employee ID is found within the Employee table, the user is granted access to the Main UI.</u> <u>7. IF Employee ID is not found, display error to user in M.E.S.S. status bar.</u>
Entry Condition	System has achieved exit condition of UC_002_TheaterLogin. The System has been granted access to Theater’ Employee Login interface.
Exit Condition	If Employee ID is found in the Theater DB’s Employee Table, the exit condition is achieved.
Quality Constraints	Employee Login should be instantaneous. The query to the Theater’s database should not take any longer than the previous use case’s transaction time.

USE CASE:	**UC_04_CreateNewSession**
Actors/Participants:	Employee
Flow of Events	1. After an Employee has been logged in via UC_03_EmployeeLogin, a "Session" object is created by MESS to log when an Employee has successfully logged into its database.
Entry Condition	Entry condition is met immediately after reaching UC_03 exit condition. (after successful Theater and Employee Login)
Exit Condition	After session object is created, exit condition is met.
Quality Constraints	Session creation should be instantaneous. It should occur immediately after employee has logged in, while GUI is loading the main ticket vendor screen.

USE CASE:	**UC_05_DisplayMainUI**
Actors/Participants:	Database
Flow of Events	1. Queries Theater Database to obtain current state of Auditoriums and their available showtimes. 2. UI populates. With correct state of Theater Database reflected.
Entry Condition	After a session has been created in UC_04, entry condition is met.
Exit Condition	After UI has been fully populated with theater info from data
Quality Constraints	The querying of the database, and the populating of the UI should be instantaneous. UC_04 and UC_05 should be completed together in under a second.

USE CASE:	**UC_06_BeginNewTransaction**

Actors/Participants:	Employee
Flow of Events	1. Employee Selects “New Transaction” 2. IF any Transactions currently in transaction window, <include UC_CancelTransaction> 3. System creates new “Current Transaction” object and clears Transaction Window in UI.
Entry Condition	At any point while employee is on main ticket vendor screen, and can click the New Transaction button.
Exit Condition	A new transaction object has been created and placed in “currentTransaction”
Quality Constraints	Instantaneous.

USE CASE:	**UC_07_AddTicketToTransaction**

Actors/Participants:	Employee
Flow of Events	1. Select Movie/Auditorium. 2. Select Showtime. 3. Select ticket type. (child, student, adult, senior, pass) <u>4. Add ticket to transaction window and add ticket object to current transaction object.</u>
Entry Condition	A new transaction has been created and is ready to have tickets added to it.
Exit Condition	Ticket has been added to current transaction.
Quality Constraints	There should be no delay for when any parameters are selected, and the adding of the ticket to the transaction should be instantaneous.

USE CASE:	**UC_08_RemoveTicketFromTransaction**
Actors/Participants:	Employee

Flow of Events	1. Select ticket from transaction window. 2. User presses delete button. 3. UI removes ticket from transaction window, and removes Ticket object from transaction object.
Entry Condition	The current transaction must not be empty.
Exit Condition	Ticket(s) removed from transaction window/object.
Quality Constraints	The removal should be immediate. There should be no noticeable delay when removing a ticket

USE CASE:	**UC_09_Cancel_Transaction**
Actors/Participants:	Employee

Flow of Events	1. Employee clicks cancel button. 2. System clears current transaction. 3. System creates new, empty transaction.
Entry Condition	When employee clicks cancel ticket
Exit Condition	When transaction has been cleared
Quality Constraints	Employee must be logged in to the system and have a transaction in the queue.

USE CASE:	**UC_10_Process_Transaction**
Actors/Participants:	Employee, Database

Flow of Events	1. Ask customer for payment method 2. Employee clicks Submit Button 3. Process payment information 4. If payment is accepted, then decrement tickets from database. 5. ELSE ask for another payment method and repeat from step 2.
Entry Condition	Employee Clicks Submit
Exit Condition	Customer gets handed the tickets
Quality Constraints	Employee must be logged into the system and have transactions in queue. Payment must be processed.

USE CASE:	**UC_11_Refund_Transaction**
Actors/Participants:	Employee, Database

Flow of Events	1. Employee selects refund option 2. System asks for manager authorization. 3. Employee enters Manager ID and PIN 4. System prompts for transaction number 5. Employee Enters transaction number. 6. System pulls transaction information from database. 7. Employee selects process transaction. 8. System subtracts refund amount from their current profit.
Entry Condition	User has valid authorization and valid transaction number
Exit Condition	Refund is completed or aborted
Quality Constraints	If the register does not currently contain the sum of the refund in profit, display a warning notification with the option to abort.

USE CASE:	**UC_12_Screen_TimeOut**

Actors/Participants:	Employee
Flow of Events	1. Employee stops interacting with the system for 10 minutes. 2. System locks most interactions, requiring the employee to input their ID and pin to continue transactions.
Entry Condition	No interaction with the program occurs for 10 minutes.
Exit Condition	Successful validation of employee ID and PIN.
Quality Constraints	The time-out shouldn't completely log-out the employee's session, but just lock input.

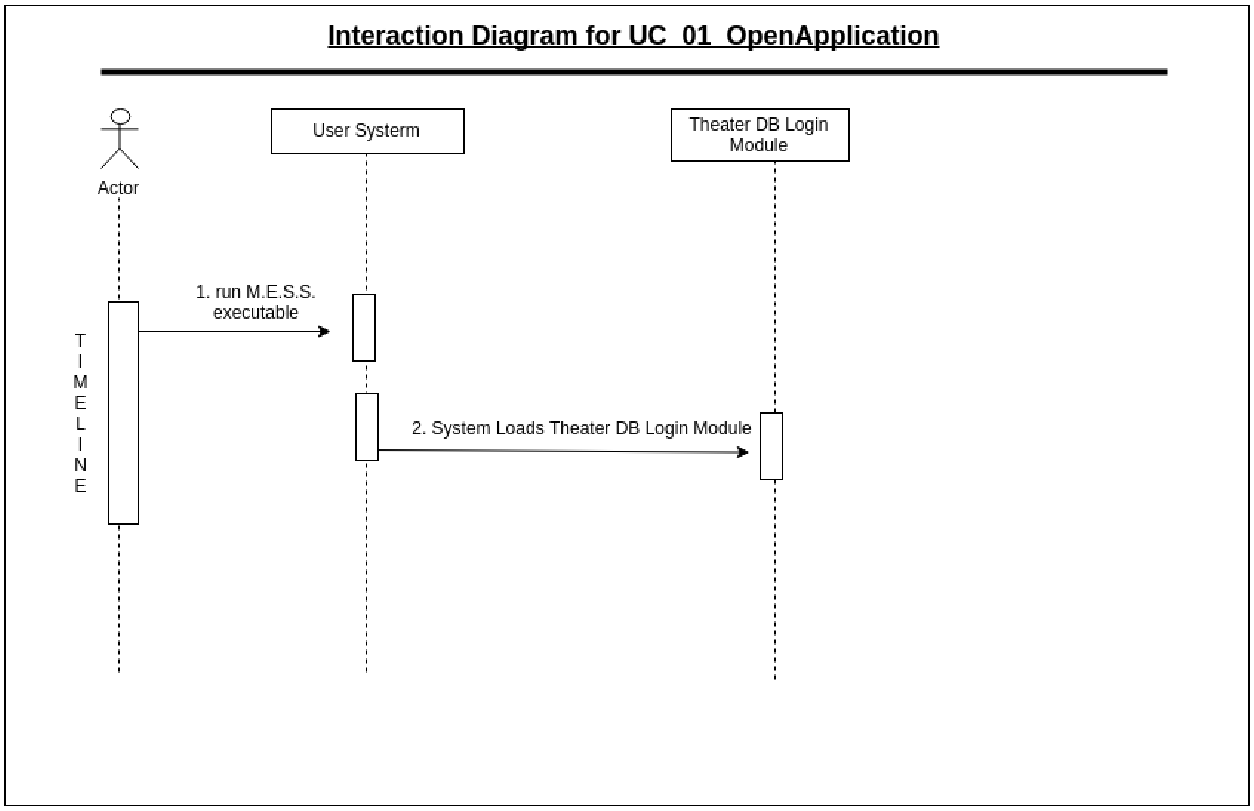
USE CASE:	**UC_13_LogOutEmployee**
Actors/Participants:	Employee
Flow of Events	1. Employee clicks “Log Out” button while currently logged into a main ticket UI. 2. Employee is logged out, and current session is ended. 3. Employee is returned to Employee Log-In screen.
Entry Condition	At any time while the Employee is successfully logged in and has access to main UI.
Exit Condition	Employee is Returned to UC_03_EmployeeLogin
Quality Constraints	Logging an employee out should be instantaneous. While the query to log the logout in the session database may take a moment to complete, the user will be able to interact with UC_003. The query will be completed in the background.

USE CASE:	**UC_14_LogOutTheater**
Actors/Participants:	Employee
Flow of Events	1. Employee clicks “Log Out of Theater” button on Employee Login screen. 2. Employee is logged out from Theater’s database. 3. System returns emloyee to Theater Login Screen. (UC_02).
Entry Condition	User is currently at Employee Login Screen (UC_03)
Exit Condition	After use has logged out of theater, they are returned to UC_02, the theater login screen.
Quality Constraints	Logging out should be instantaneous. Returning to UC_02 from UC_03 should be immediate.

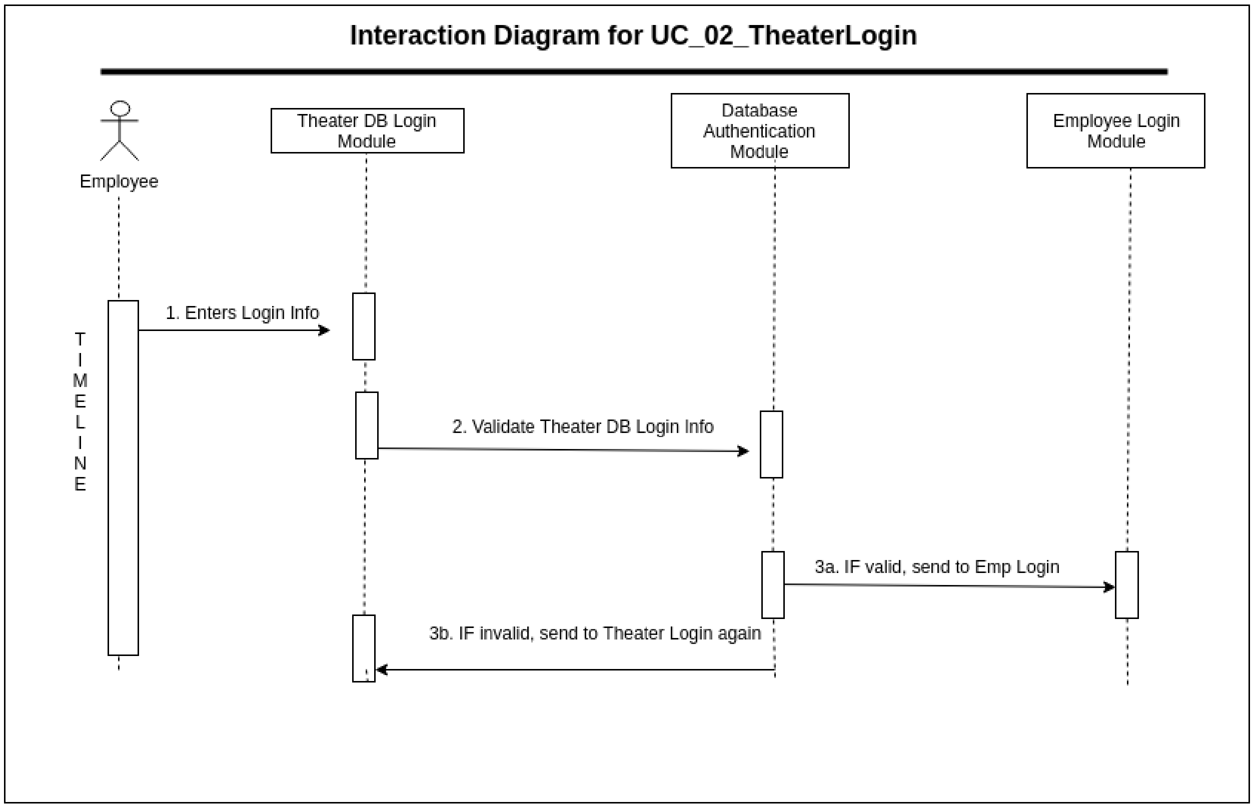
USE CASE:	**UC_15_CloseApplication**
Actors/Participants:	Employee
Flow of Events	1. User clicks “Exit” button on Theater Login screen (UC_002). 2. System terminates application.
Entry Condition	User is currently at Theater Log-In screen.
Exit Condition	System continues in current state, with M.E.S.S. terminated.
Quality Constraints	Immediate. THe user should be able to close the program swiftly with no worry about loss of data.

Interaction Diagrams

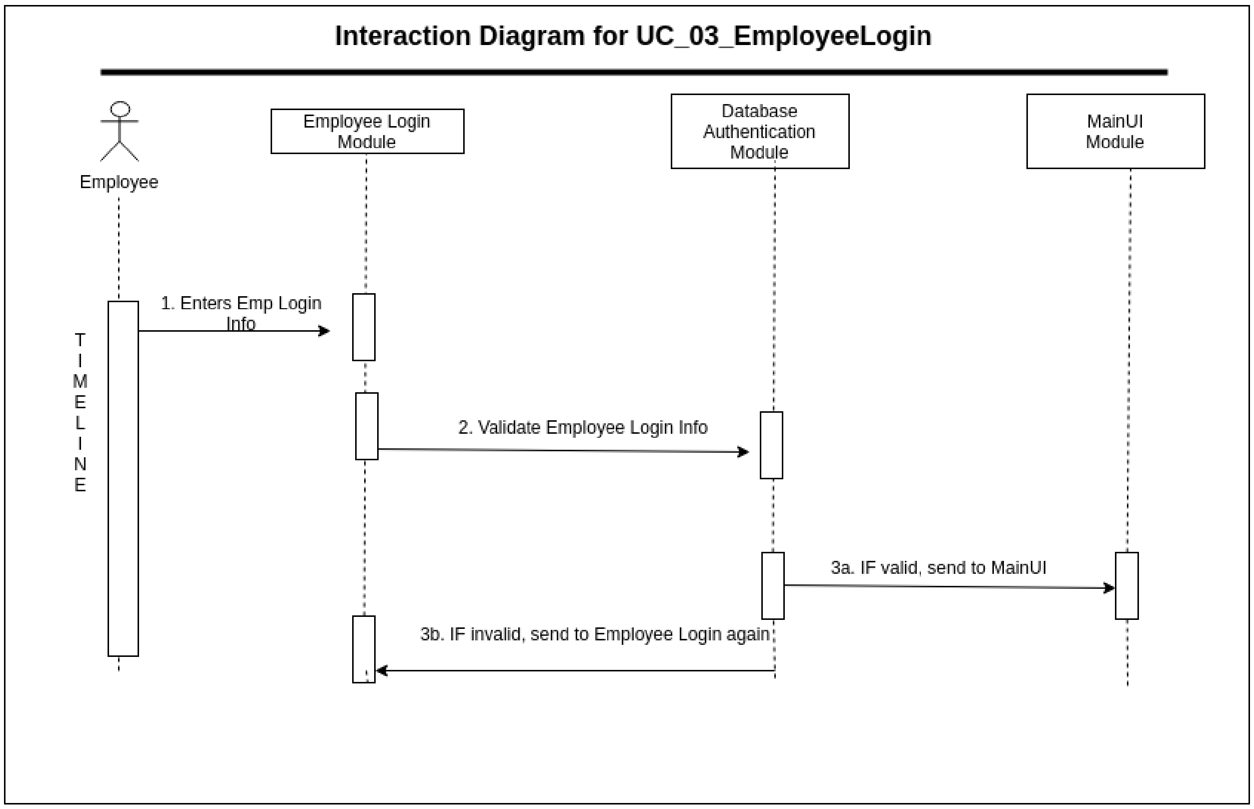
1. UC_01_OpenApplication


Interaction Diagram for UC_01_OpenApplication
Actor
User Systerm
Theater DB Login Module
1. run M.E.S.S. executable
2. System Loads Theater DB Login Module
TIMELINE

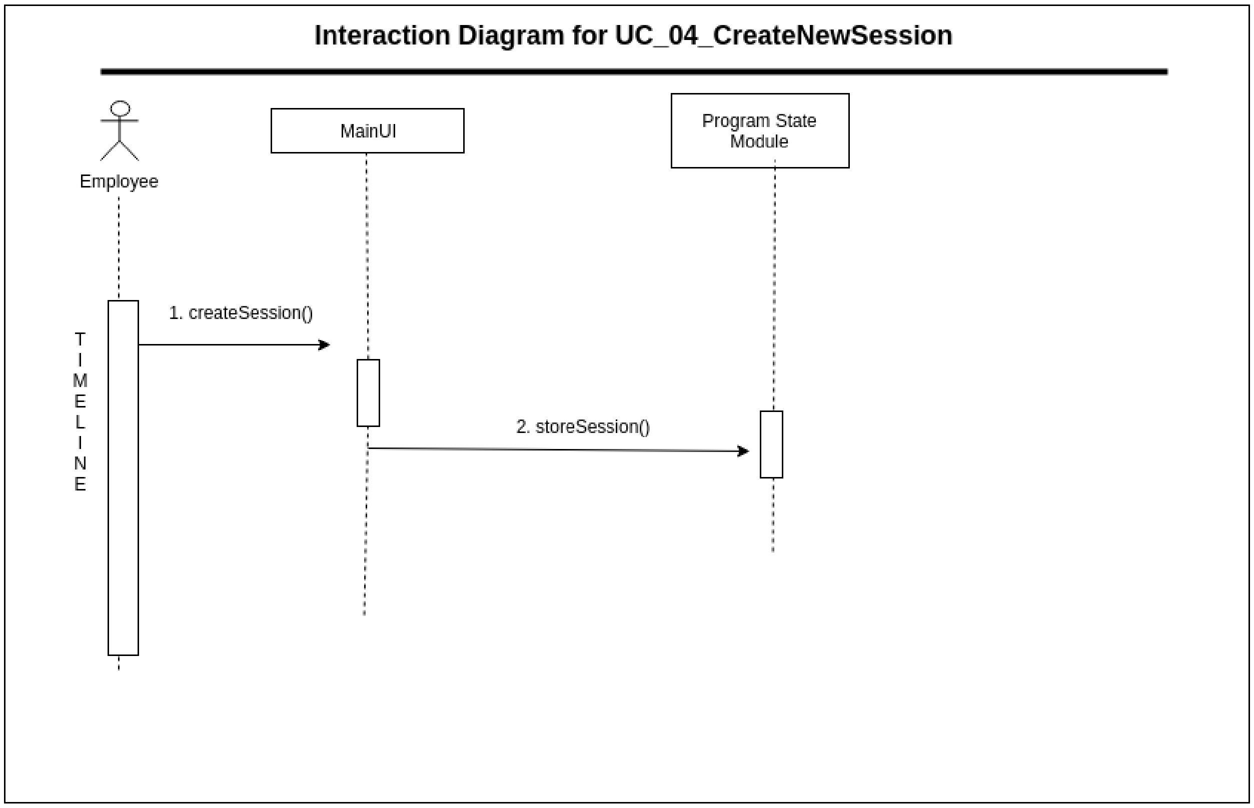
2. UC_02_TheaterLogin


Interaction Diagram for UC_02_TheaterLogin
Employee
Theater DB Login Module
Database Authentication Module
Employee Login Module
TIMELINE
1. Enters Login Info
2. Validate Theater DB Login Info
3a. IF valid, send to Emp Login
3b. IF invalid, send to Theater Login again

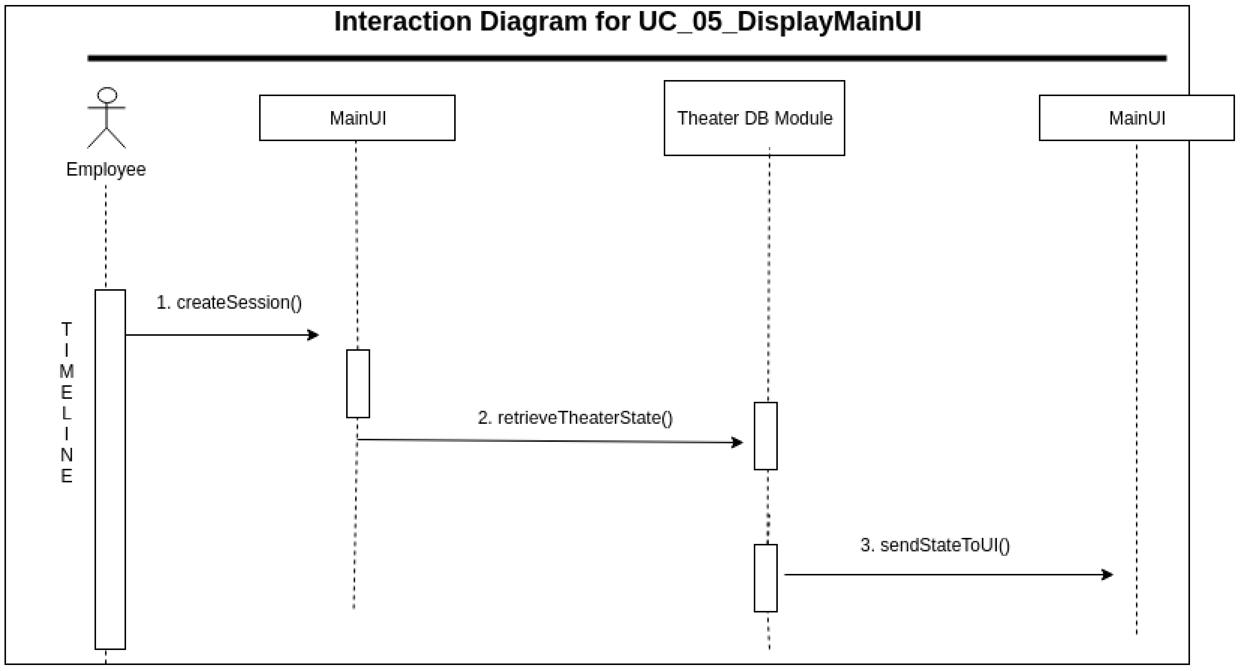
3. UC_03_EmployeeLogin


Interaction Diagram for UC_03_EmployeeLogin
Employee
Employee Login Module
Database Authentication Module
MainUI Module
T I M E L I N E
1. Enters Emp Login Info
2. Validate Employee Login Info
3a. IF valid, send to MainUI
3b. IF invalid, send to Employee Login again

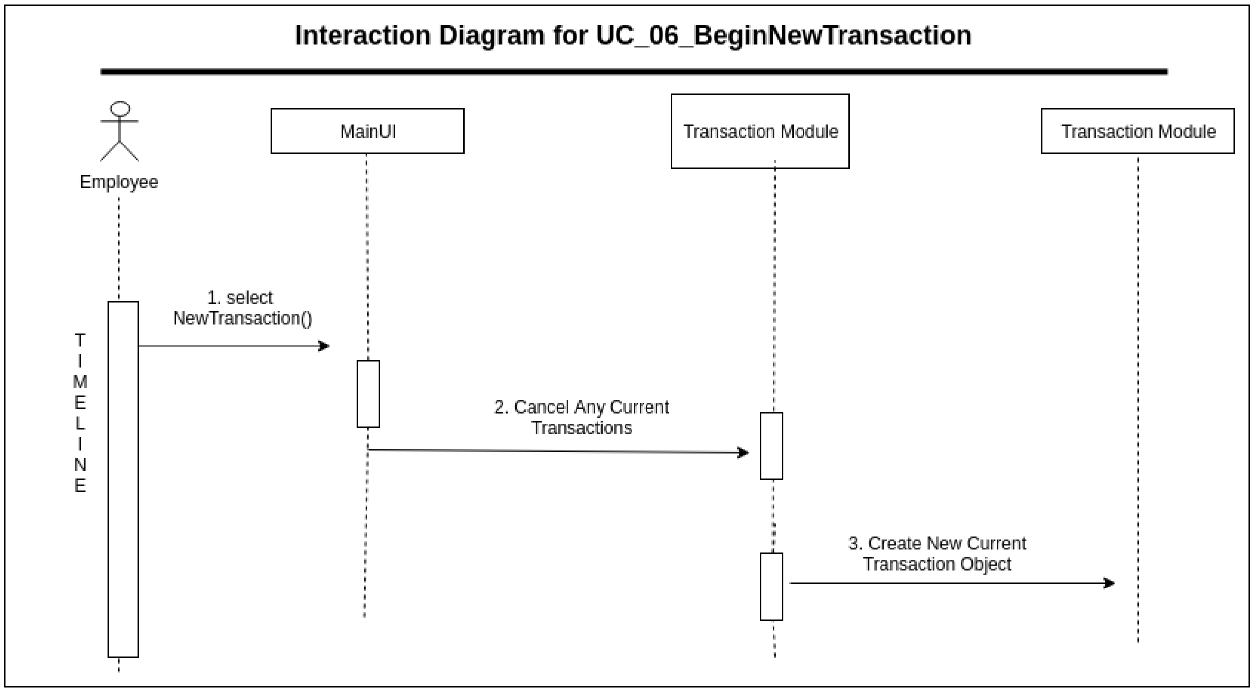
4. UC_04_CreateNewSession



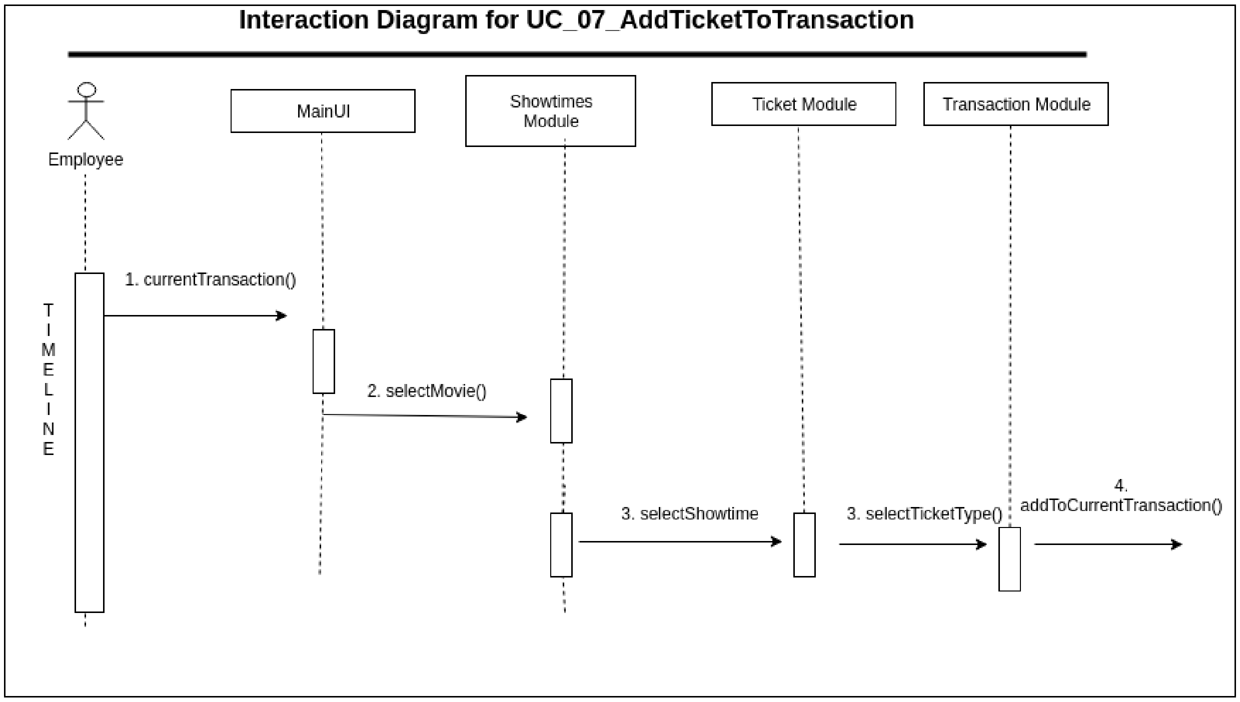
5. UC_05_DisplayMainUI


Interaction Diagram for UC_05_DisplayMainUI
Employee
MainUI
Theater DB Module
MainUI
TIMELINE
1. createSession()
2. retrieveTheaterState()
3. sendStateToUI()

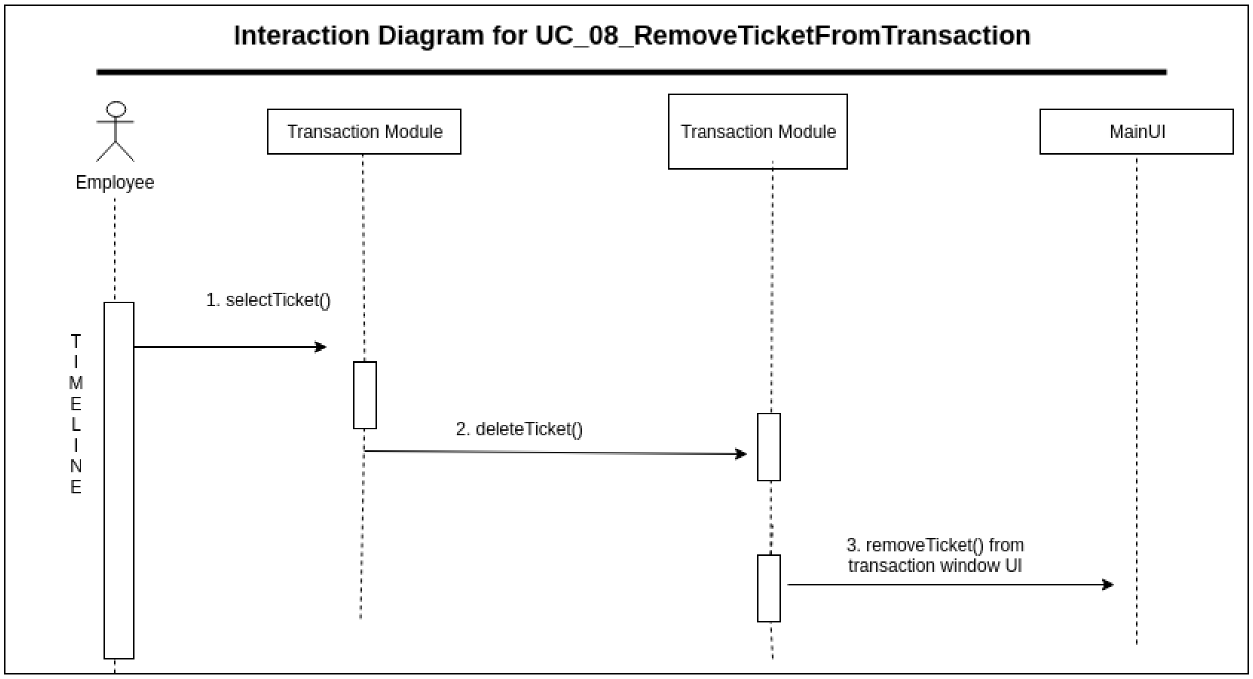
6. UC_06_BeginNewTransaction


Interaction Diagram for UC_06_BeginNewTransaction
Employee
MainUI
Transaction Module
Transaction Module
TIMELINE
1. select NewTransaction()
2. Cancel Any Current Transactions
3. Create New Current Transaction Object

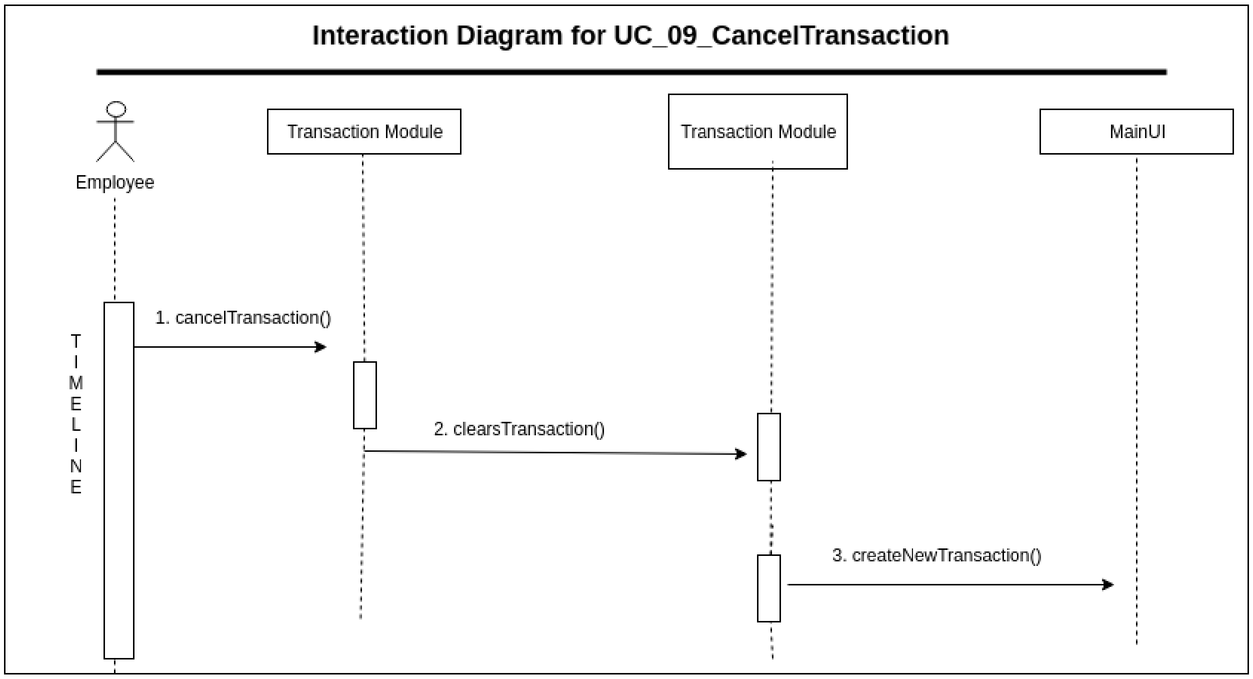
7. UC_07_AddTicketToTransaction



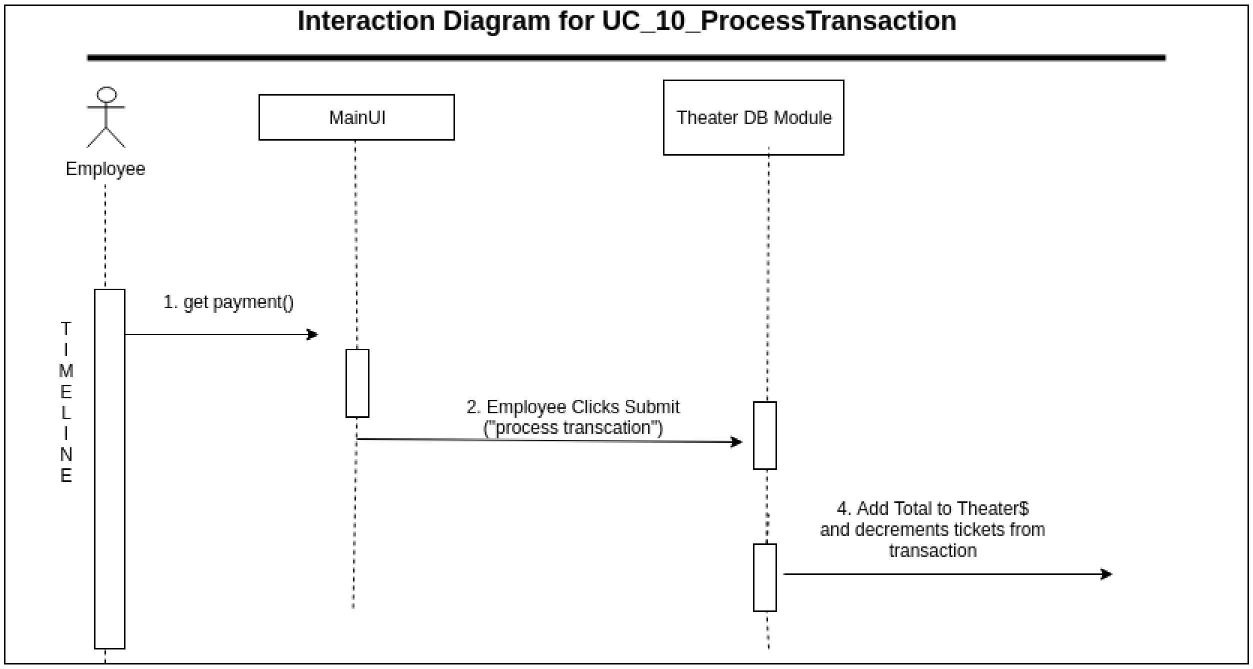
8. UC_08_RemoveTicketFromTransaction


Interaction Diagram for UC_08_RemoveTicketFromTransaction
Employee
Transaction Module
Transaction Module
MainUI
TIMELINE
1. selectTicket()
2. deleteTicket()
3. removeTicket() from transaction window UI

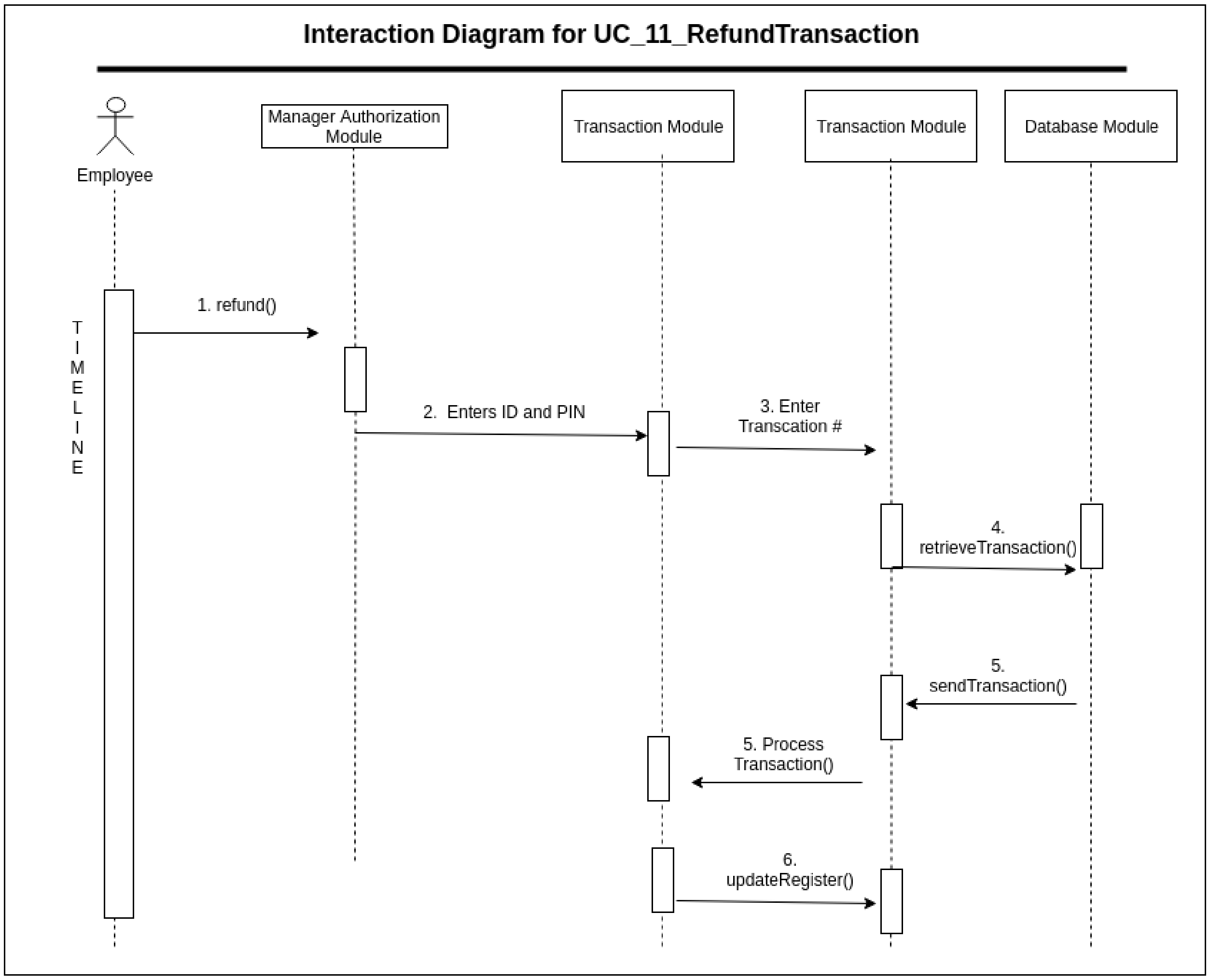
9. UC_09_CancelTransaction



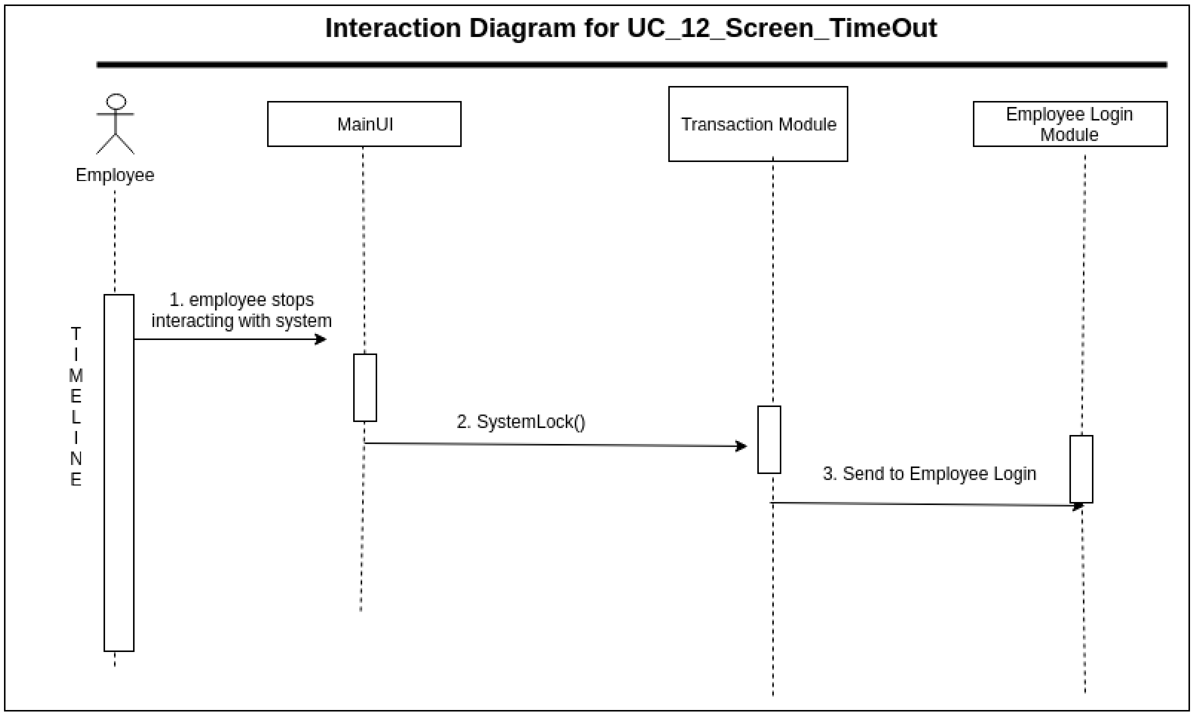
10. UC_11_ProcessTransaction


Interaction Diagram for UC_10_ProcessTransaction
Employee
MainUI
Theater DB Module
TIMELINE
1. get payment()
2. Employee Clicks Submit ("process transcation")
4. Add Total to Theater$ and decrements tickets from transaction

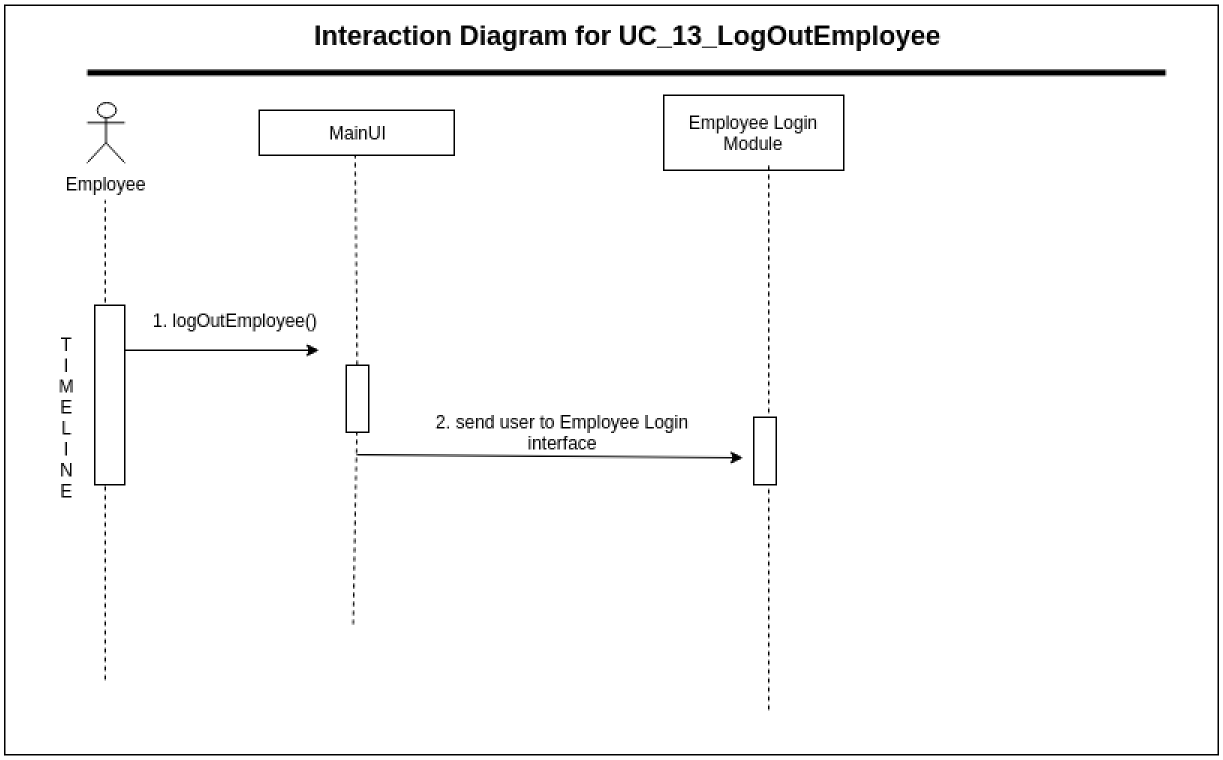
11. UC_11_Refund_Transaction



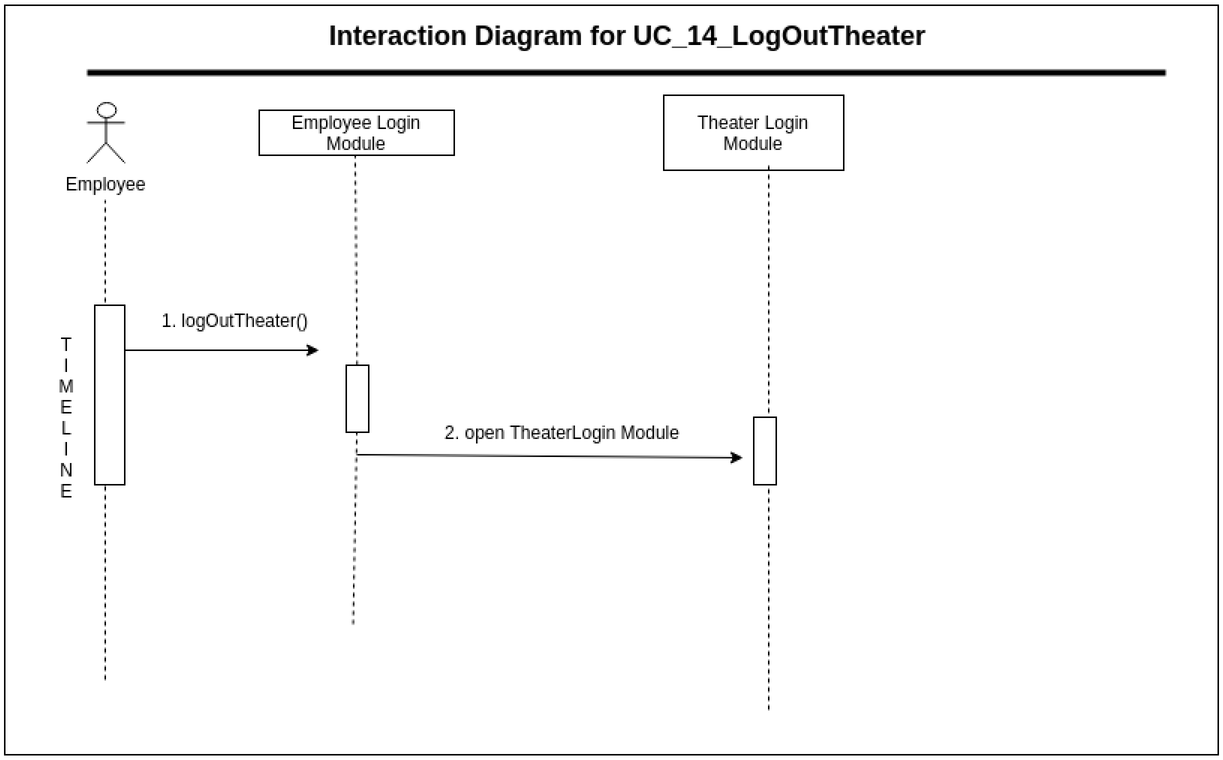
12. UC_12_Screen_TimeOut


Interaction Diagram for UC_12_Screen_TimeOut
Employee
MainUI
Transaction Module
Employee Login Module
TIMELINE
1. employee stops interacting with system
2. SystemLock()
3. Send to Employee Login

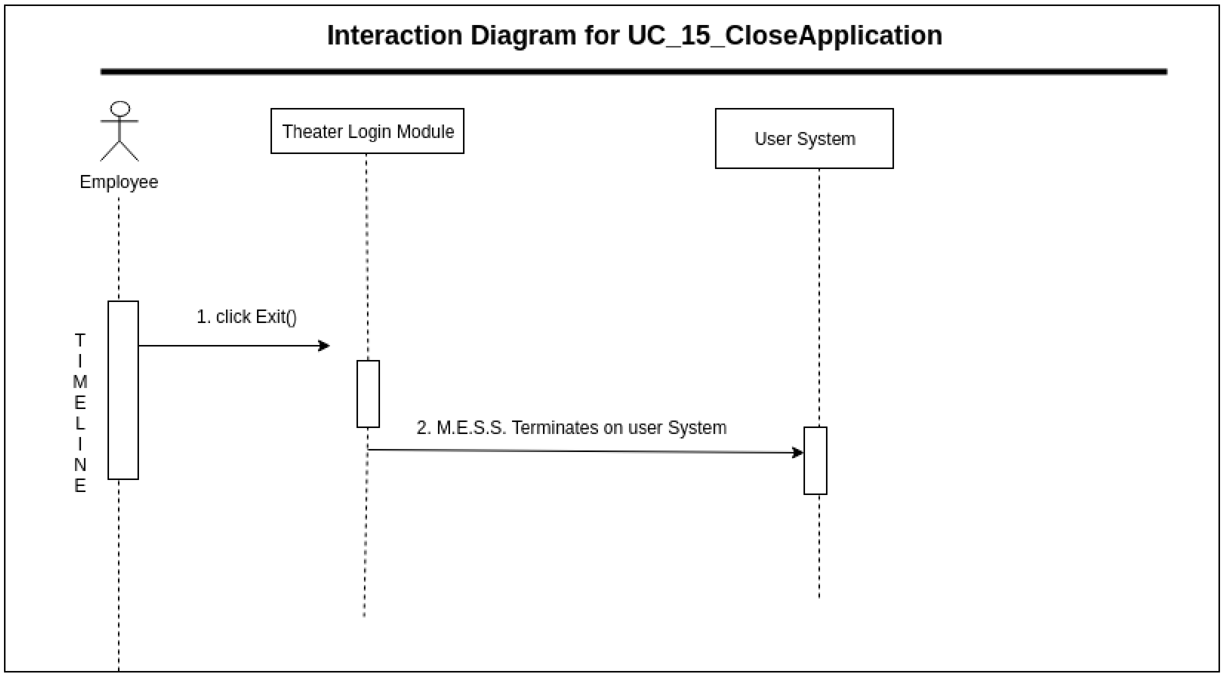
13. UC_13_LogOutEmployee


Interaction Diagram for UC_13_LogOutEmployee
Employee
MainUI
Employee Login Module
TIMELINE
1. logOutEmployee()
2. send user to Employee Login interface

14. UC_14_LogOutTheater



15. UC_15_CloseApplication


Interaction Diagram for UC_15_CloseApplication
Employee
Theater Login Module
User System
TIMELINE
1. click Exit()
2. M.E.S.S. Terminates on user System

Database Selection

A MySQL database will be required to properly interface with the Movie Entertainment Software System. Upon loading M.E.S.S., the user is greeted with a MySQL Database Log-In interface. Any database that interacts with M.E.S.S. must adhere to the proper schema specified.

M.E.S.S. will be hosting all customer MySQL databases via GoDaddy.com. A user can also use their own local MySQL database as long as it adheres to the proper schema required to interact with M.E.S.S.

M.E.S.S. will be using the JDBC Java Database Connection Driver within the application and database layers of the java source code to access the specified database.

Work Share Document: (10/2/2016)

Kris Hermstad:
Team Lead, Database & Application Layer Development

Lindsay Stone:
User, UI Layer Development

Emily Horton:
Client, UI Layer Development

Kirtan Patel:
Technical Writer, Application Layer Development

Jack Martin:
Product Support, Database Layer Development

MESS: GANTT for 10/2/2016

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	Use Cases, Database Selection, Deliverable 3	16	9/18	10/2	Yes
Jack Martin	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Emily Horton	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Lindsay Stone	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Kirtan Patel	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Next Deliverable:	Due:				
Object Design	10/9/2016	7	10/2	10/9	

PREVIOUS GANTTS:

M.E.S.S. Gantt for 9/18/2016					
Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	Draft Problem Statement, RTM, and Gantt	3	9/15	9/18	Yes
Jack Martin	Database E-R Diagram	10	9/8	9/18	Yes
Emily Horton	Being drafting UI design mockups	16	9/2	9/18	Yes
Lindsay Stone	Draft UI mockups, experiment with JavaFX Scene Builder, create PERT chart	3	9/2	9/18	Yes
Kirtan Patel	Forumlate Documentation Standard for all project documents	16	9/2	9/18	Yes
Next Deliverable:	Due:				
System Analysis	9/28/2016				

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

As of 10/2/2016

Rationale For Use Cases:

UC_01_OpenApplication:
This use case is necessary to represent the initial program execution. It captures all the necessary requirements to run application on user's system.

UC_02_TheaterLogin:
This use case is necessary to allow for users to connect to to their specific database. In order to access the movies/showtimes for a specific theater, the user must be authorized to do so. This is handled with UC_02.

UC_03_EmployeeLogin:
Employee Login is necessary to log all transactions and the employee who created them. Employee Login also provides another step of security for anyone accessing the theaters database. A malicious user may gained access to the Employee Login screen, but without correct Employee Login info, they won't be able to connect and manipulate the database.

UC_04_CreateNewSession:
A session is necessary in order to keep track of which employees processed which transactions. Each time an employee is logged in, a "session" is created so all activity within a session is logged with a unique employee id attached. This helps theater management have a clear understanding of who has done what.

UC_05_DisplayMainUI:
This is necessary to accurately display the contents of a theaters database. Every UI is unique to the theater and it's requirements derived by the theaters Movies/Auditoriums/Showtimes. This use case shows how the UI is derived via the Database. This is necessary every time a new session is created, as each theater has different movies/showtimes.

UC_06_BeginNewTransaction:
A "Transaction" object is required every time a new transaction is administered. A transaction can only hold Tickets. A new transaction is created after a transaction is processed or cancelled. When the program loads, a New Transaction is created. The "CurrentTransaction" object is important to maintaining an organized record of transactions. Each transaction has a unique ID which can later be recalled when a refund is needed.

UC_07_AddTicketToTransaction:
This use case is an absolute necessity as this allows for tickets to be added to the Current Transaction. The entire functionality of this system revolves around this Use Case.

UC_08_RemoveTicketFromTransaction:
Just as it is important to add tickets, the system also needs to be able to remove tickets from Current Transactions that have yet to be processed. This allows for a more modifiable Current Transaction.

UC_09_CancelTranscation:
Cancel Transaction is needed to clear Transaction quickly. If a Current Transaction has 10 tickets in the ticket window, the user would have to remove each ticket one by one, via UC_08_RemoveTicketFromTransaction. This allows for a Transaction to be cleared, regardless of its contents. Important use case, as it allows for quicker usability.

UC_10_ProcessTransaction:
Required to process a transaction. This Use Case is instrumental in modeling an accurate state of the theater and its database. When a transaction is processed, The ticket value is added to the Theaters Sales, and the Tickets that were sold are decremented from the Theater's Ticket Database.

UC_11_RefundTransaction:
This is necessary to allow for customers to get a refund. Most importantly, when a refund is offered, new tickets are added to the Movie's Ticket Database. Refunding helps reflect the accurate amount of seats actually filled. (ie. A customer could buy all the tickets for a showtime, Selling Out the show. If this customer refunded their transaction the system needs to add all of the refunded tickets back to the AVAILABLE tickets.

UC_12_TimeOut_Screen:
An important security feature. If a user walks away from their interface, the system is left open to vulnerabilities. With a time-out, if the user hasn't interacted after a certain time, it locks the system out until interacted with. After being locked out, the employee is required to log-in again to using their Employee ID.

UC_13_LogOutEmployee:
As multiple employees may use one system, an employee log-out functionality is very important. This allows for employees to log in and out of the system, without having to reconnect to the theaters database each time.

UC_14_LogOutTheater:
LogOutTheater allows for an extra level of security while your system is being operated. It also allows for the ability to easily switch between different theater databases upon Logging Out.

UC_15_CloseApplication:
The program must have a way for the user to terminate the system, from the UI.

M.E.S.S. Deliverable #4

Object Design

Software Engineering: Tuesday/Thursday 5:30pm w/ Mr. Rao

Due: 10.16.2016

TEAM JEKKL (GROUP #9)

Kris Hermstad: Team Lead, Database Developer

Emily Horton: Client, UI Developer

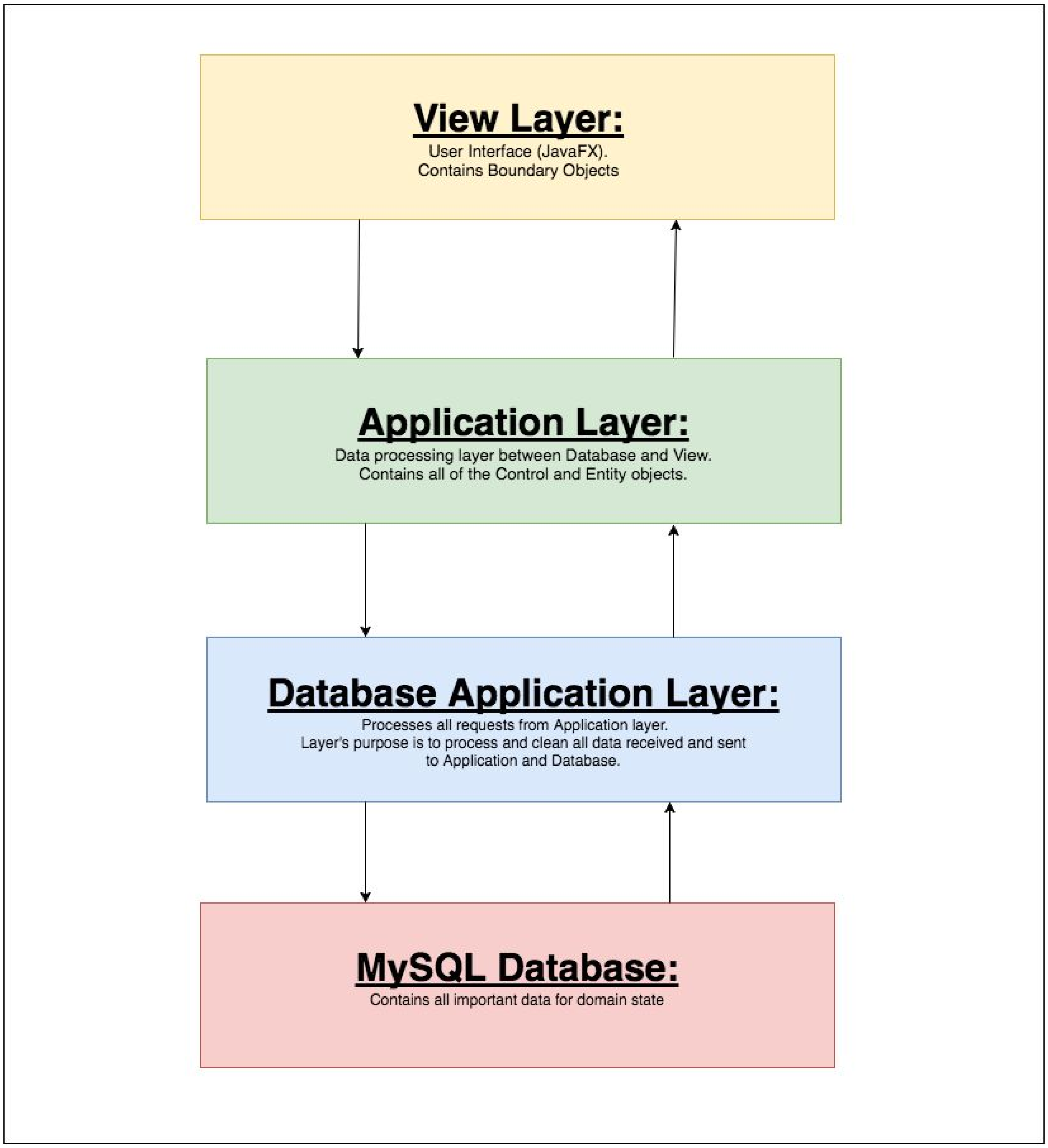
Kirtan Patel: Technical Writer, Application Developer

Lindsay Stone: User, UI Developer

Jack Martin: Product Support, Database Developer

Contains:

Software Architecture for M.E.S.S.


View Layer:
User Interface (JavaFX).
Contains Boundary Objects
Application Layer:
Data processing layer between Database and View.
Contains all of the Control and Entity objects.
Database Application Layer:
Processes all requests from Application layer.
Layer's purpose is to process and clean all data received and sent to Application and Database.
MySQL Database:
Contains all important data for domain state

Software Architecture Continued:

View Layer:

GUI implemented in JavaFX. The main interface that the user (employee) interacts with. Elements within the GUI contain the control objects from the application layer. When elements from the GUI are interacted with, their respective control objects/methods are triggered.

Application Layer:

Application layer contains control objects that are triggered by interactions with the View Layer. The application layer interacts with the View Layer and the Database Application Layer. The View Layer will trigger a control object from the application layer, which will then pass requests to the Database Application layer.

All updates that come from the database must pass through the application layer. For example, when the user logs their theater in, a request is sent to the database to retrieve all theater information. This data is then sent BACK through the database application layer, to the application layer, where it then updates the View Layer.

While many of the interactions are one way, such as sending a finished transaction to the database, most interactions will have a bidirectional flow.

Database Application Layer:

The Database Application Layer takes requests from the Application layer and interacts with the Database. The database application layer contains all methods necessary to interact with the database. The Database Application Layer is separate from the Application Layer because the Database Layer is specifically written for MySQL. In the future, a developer may choose to use a different database, and rather than having to completely rewrite the application layer, they will only have to rework the Database Application Layer. The DAL uses the JDBC Database Connector to interact with the MySQL database.

Data received by the Database will also be sent back via the Database Application Layer to the Application Layer. All updates coming from the Database to the Application Layer will be passed to the View layer providing an update to the UI.

MySQL Database:

Contains all data relevant to theater. The database is accessed via the Database Application Layer. The MySQL Database is the only system that exists outside of M.E.S.S.

Requirements Trace Matrix: (as of 10/16/2016)

See Use Case Index to find corresponding Use Case for each Entry.

Entry #	Paragraph #	RTM Entry	Entry Type	Use Case
1	1	Shall be OS independent, working on any machine with the JVM.	SW	01
2	3	Shall have a Theater Login Interface that connects to a MySQL db via the internet	SW/HW	02
3	5	Shall have an employee Login interface that verifies an employee has access to the Theaters DB	SW	03
4	8	A new session shall be created when an employee successfully logs in.	SW	04
5	8	There shall be a MainUI from where all ticket vending occurs	SW/HW	05
6	9	There shall be a Currently Active Transaction created upon which all tickets will be added to.	SW	06
7	11	Shall be able to add ticket(s) to Currently Active Transaction	SW	07
8	11	Shall be able to remove any tickets added to Currently Active Transaction.	SW	08
9	11	Shall be able to Cancel any transaction currently active.	SW	09
10	11	Shall have the functionality to process a Currently Active Transaction	SW	10
11	11	Shall be able to offer a “refund” for any transaction in the Theater’s Transaction Database.	SW	11
12	7	Shall have time-out (lock) an employee’s session if they have not interacted with system after a certain specified period of time.	SW	12
13	12	Shall have a log-out functionality for a currently logged in employee.	SW	13
14	12	Shall have a log-out functionality for a currently logged in Theater.	SW	14
15	13	Shall have an Exit Application functionality.	SW	15
16	13	Updated Modern UI	NTH	
17	13	Fast, instantaneous transactions	NF	

Weighting Factor Estimate: (UFP)

Measurement Parameters							
	COUNT		**SIMPLE**	**AVERAG E**	**COMPLEX**		**TOTAL**
Number of User Inputs	27	X	3	4	6	=	90
Number of User Outputs	7	X	4	5	7	=	36
Number of User Inquiries	5	X	3	4	6	=	20
Number of Internal Files	5	X	7	10	15	=	49
Number of External Interface of Files	1	X	5	7	10	=	10
GRAND TOTAL (FP)	UFP					=	**205**

Calculation Table:

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**
ILFs	**2**	**Low**	**X 7 =**	**14**	
	2	**Average**	**X 10 =**	**20**	
	1	**High**	**X 15 =**	**15**	
					49

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**

ELFs		**Low**	**X 5 =**		
		Average	**X 7 =**		
	1	**High**	**X 10 =**		
					10

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**
EIs	**22**	**Low**	**X 3 =**	**66**	
	3	**Average**	**X 4 =**	**12**	
	2	**High**	**X 6 =**	**12**	
					90

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**
EOs	**3**	**Low**	**X 4 =**	**12**	
	2	**Average**	**X 5 =**	**10**	
	2	**High**	**X 7 =**	**14**	
					36

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**
EQs	**2**	**Low**	**X 3 =**	**6**	
	2	**Average**	**X 4 =**	**8**	
	1	**High**	**X 6 =**	**6**	
					20

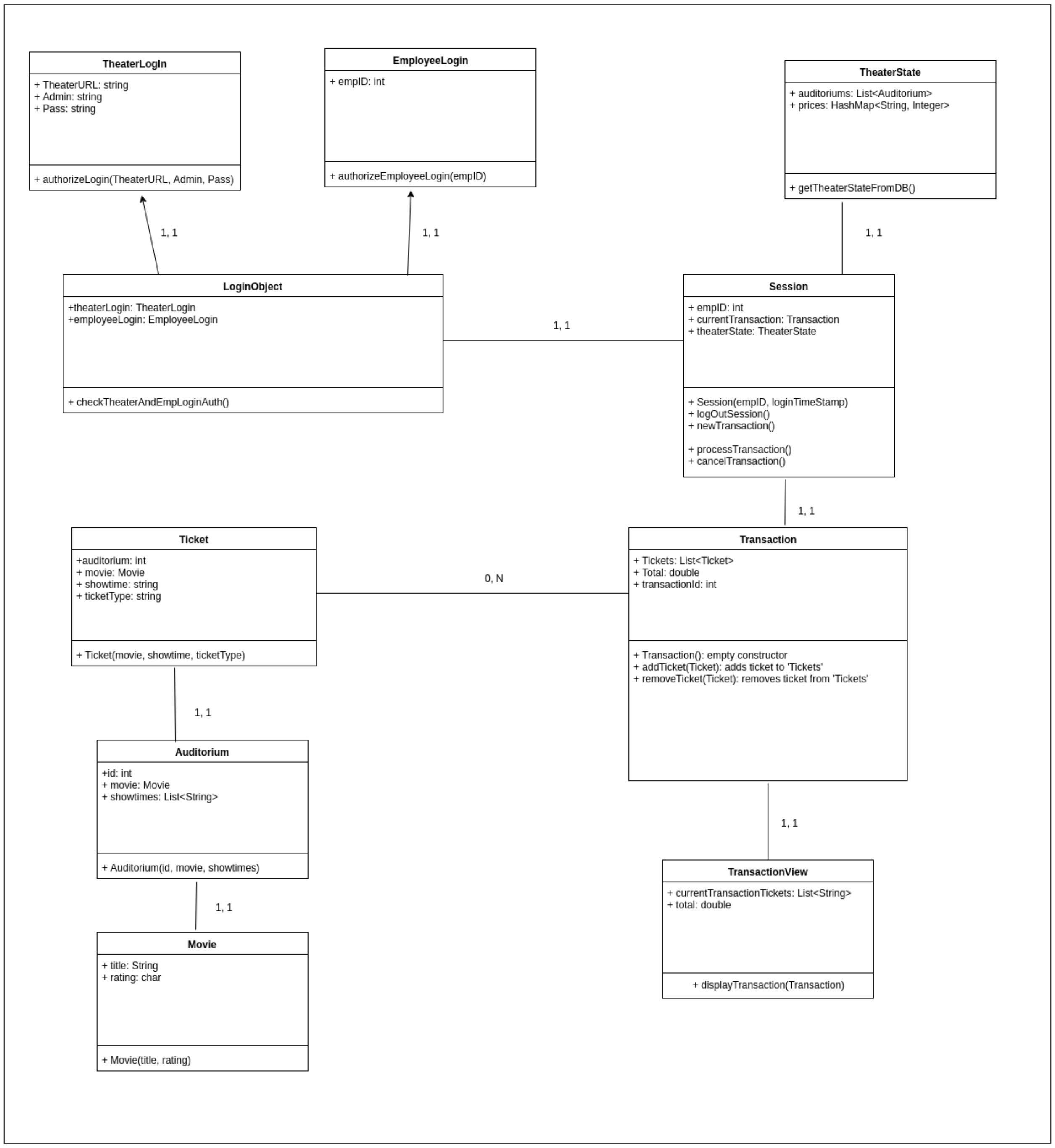
Category Ratings:

Category	Rating
1. Does the system require reliable backup and recovery?	2
2. Are data communications required?	3
3. Are there distributed processing functions?	1
4. Is performance critical?	4
5. Will the system run in an existing heavily utilized operational environment?	5
6. Does the system require on-line data entry?	3
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?	1
8. Are the master files updated on-line?	5
9. Are the inputs, outputs, files or inquiries complex?	2
10. Is the internal processing complex?	2
11. Is the code designed to be reusable?	5
12. Are conversion and installation included in the design?	4
13. Is the system designed for multiple installations in different organizations?	5
14. Is the application designed to facilitate change and ease of use by the user?	3
TOTAL SUM OF ALL CATEGORY RATINGS:	**45**

VAF = [0.65 + 0.01 * (45)]
VAF = 1.1
FPC = UFP * VAF
FPC = 205 * 1.1

FPC = 225.5

Class Diagrams for M.E.S.S.


TheaterLogIn
+ TheaterURL: string
+ Admin: string
+ Pass: string
+ authorizeLogin(TheaterURL, Admin, Pass)
EmployeeLogin
+ empID: int
+ authorizeEmployeeLogin(empID)
TheaterState
+ auditoriums: List<Auditorium>
+ prices: HashMap<String, Integer>
+ getTheaterStateFromDB()
1, 1
1, 1
1, 1
LoginObject
+theaterLogin: TheaterLogin
+employeeLogin: EmployeeLogin
+ checkTheaterAndEmpLoginAuth()
1, 1
Session
+ empID: int
+ currentTransaction: Transaction
+ theaterState: TheaterState
+ Session(empID, loginTimeStamp)
+ logOutSession()
+ newTransaction()
+ processTransaction()
+ cancelTransaction()
1, 1
Ticket
+auditorium: int
+ movie: Movie
+ showtime: string
+ ticketType: string
+ Ticket(movie, showtime, ticketType)
0, N
Transaction
+ Tickets: List<Ticket>
+ Total: double
+ transactionId: int
+ Transaction(): empty constructor
+ addTicket(Ticket): adds ticket to 'Tickets'
+ removeTicket(Ticket): removes ticket from 'Tickets'
1, 1
Auditorium
+id: int
+ movie: Movie
+ showtimes: List<String>
+ Auditorium(id, movie, showtimes)
1, 1
1, 1
TransactionView
+ currentTransactionTickets: List<String>
+ total: double
+ displayTransaction(Transaction)
Movie
+ title: String
+ rating: char
+ Movie(title, rating)

Work Share Document: (10/16/2016)

<u>Kris Hermstad:</u>
Team Lead, Database & Application Layer Development

<u>Lindsay Stone:</u>
User, UI Layer Development

<u>Emily Horton:</u>
Client, UI Layer Development

<u>Kirtan Patel:</u>
Technical Writer, Application Layer Development

<u>Jack Martin:</u>
Product Support, Database Layer Development

MESS: GANTT for 10/16/2016

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Object Design	Function Point Cost,Interface design ,Object relationship, Updates to the task and role charts	14	10/2	10/16	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	System Architecture, Object Design, CIDs, Deliverable 3	14	10/2	10/16	Yes
Jack Martin	Object Design	14	10/2	10/16	Yes
Emily Horton	Object Design & PCI	14	10/2	10/16	Yes
Lindsay Stone	Object Design & PCI	14	10/2	10/16	Yes
Kirtan Patel	Object Design & PCI	14	10/2	10/16	Yes
Next Deliverable:	Due:				
Project Rationale	10/23/2016	7	10/16	10/23	

PREVIOUS GANTTS

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delivered by End Date?
Kris Hermstad (Team Lead)	Use Cases, Database Selection, Deliverable 3	16	9/18	10/2	Yes
Jack Martin	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Emily Horton	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Lindsay Stone	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Kirtan Patel	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Next Deliverable:	Due:				
Object Design	10/9/2016	7	10/2	10/9	

M.E.S.S. Gantt for 9/18/2016					

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	Draft Problem Statement, RTM, and Gantt	3	9/15	9/18	Yes
Jack Martin	Database E-R Diagram	10	9/8	9/18	Yes
Emily Horton	Being drafting UI design mockups	16	9/2	9/18	Yes
Lindsay Stone	Draft UI mockups, experiment with JavaFX Scene Builder, create PERT chart	3	9/2	9/18	Yes
Kirtan Patel	Forumlate Documentation Standard for all project documents	16	9/2	9/18	Yes

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

As of 10/16/2016

LoginObject:
The LoginObject encapsulates both the TheaterLoginObject as well as the EmployeeLoginObject. This will allow the user to login to the theater via the TheaterLoginObject and then the EmployeeLoginObject. When a user logs out of the Theater, the LoginObject will be destroyed, and a new blank LoginObject will be created. The LoginObject should help save state information about what thater is currently logged in, and which employee is currently accessing the system.

TheaterLogin:
The TheaterLogin object contains the TheaterDatabaseURL, the Admin name, and the password. This object is passed to the Database Connector which will connect M.E.S.S. to the Theater Database URL that was entered. This object is necessary to authorize M.E.S.S. to get access to the proper database.

EmployeeLogin:
The employee login contains only one field, the Employee ID (empID). The user will input their empId into the Employee Login interface, and this empID will be checked against the TheaterDB's employee table. This object is necessary to add an extra step of security to the system.

Session:
A session object is necessary to keep track of the current transactions being made. By having a Session object that contains an empID, the system can link each transaction to an employeeID. This will allow for management to know which employee processed a specific transaction.
Also, when a session is created, the current Theater State is needed from the Theater Database.

TheaterState:
The TheaterState object is necessary to obtain all the accurate information needed to populate the UI. When a Session object is created, a TheaterState object is instantiated. This object will contain all auditoriums,movies and their respective showtimes. The GUI will will then extract the TheaterState from this object and display

it. Ie. The TheaterState returns with 3 auditoriums and their movies and showtimes. So the UI will create one button for each auditorium, with the correct movie title and showtimes displayed.

Transaction:
A transaction object is necessary to capture all ticket information for a specific transaction. Tickets are added to the transaction, and a total is created. A customer will pay, and then Transaction is then processed. From here, the transaction object is used to properly update the amount of tickets available in the theater database. The transaction object also updates the current sales amount. A Transaction object can also be cancelled. In this case, an empty Transaction is instantiated in its place.

Ticket:
A Ticket object holds all specific information for a single ticket purchase.
Ie. ([Movie] @ [Showtime] in [Auditorium] for Adult)
Tickets are added to Transactions, and they will only exist within transaction objects. A ticket helps display the correct Movie/Showtime/Auditorium information to the Transaction Window. This can be reviewed before processing a transaction. -- Each ticket has a ticketType: child, adult, student, senior, pass, etc. Each ticketType has a different price, so when a ticket is added to a transaction, it is compared to a price table that will accurately update the transaction total.

Auditorium:
An auditorium contains a theater as well as a list of showtimes. The auditorium object exists within a ticket object. The Auditorium object is necessary as it contains all Movie information that will be included in the Ticket. Instead of selling tickets by the movie, the M.E.S.S. sells tickets by the Auditorium, which can contain any movie object.

Movie:
The Movie object contains just the title and the rating of the movie. A Movie Object only exists within an Auditorium Object. (which only is instantiated in a Ticket Object.) The movie object simply is used to display the correct Movie title and rating within the Ticket and its transaction. The tickets are sold by auditorium and showtimes, rather than Movie title, so the Movie object acts more as a visual aid, when it is displayed in the TransactionView.

TransactionView:
The TransactionView is the window within the UI that displays the currentTransaction information. The TransactionView is necessary as it will display the data within the

current Transaction object, which is within the current session. The TransactionView will display all tickets currently in the transaction, as well as the current total. The TransactionView can be edited easily, allowing for the removal and addition of Transactions.

Rationale for System Architecture:

M.E.S.S. is designed on a Four-Tier System architecture. The system requirements demand an architecture that allows for future development and expansion. With this in mind, the system and its components need to be decoupled in a way that new modules and features can be added without issue.

The System contains four layers:
1. View: the GUI, which acts as the boundary object within our system.
2. Application: this layer processes all requests from the View and Database layers.
3. Database Application: this layer turns all application requests in to a proper SQL statement that will be sent to the database.
4. MySQL Database: contains all data that system interacts with.

The system is built around a Model-View-Controller architecture. The model being the database, the controller being the Application and Database Application layers, and the GUI being the view layer.

While one might conclude that M.E.S.S. actually are using a three-tier system architecture because the Application Layer and the Database Application Layer are both “controller” types, it is important to decouple the Application Layer and the Database Application Layer. In the future, the client may want to change their database from MySQL to something else. This would mean only the Database Application layer would need to be reworked in order to provide a connection between the application layer and the Database.

The application layer will be written in such a way that developers will only have to create new Database Interface layers to add support for more databases. If M.E.S.S. used a strict three-tier architecture, then the application layer would be tightly coupled with all of our Database Interface code. So, for this reason M.E.S.S. stresses a four-tier system architecture.

As of 10/16/2016

M.E.S.S. Deliverable #5

Project Rationale

Software Engineering: Tuesday/Thursday 5:30pm w/ Mr. Rao

Due: 10.23.2016

TEAM JEKKL (GROUP #9)

Kris Hermstad: Team Lead, Database Developer

Emily Horton: Client, UI Developer

Kirtan Patel: Technical Writer, Application Developer

Lindsay Stone: User, UI Developer

Jack Martin: Product Support, Database Developer

Contains:

Rationale For Use Cases:

UC_01_OpenApplication:
This use case is necessary to represent the initial program execution. It captures all the necessary requirements to run application on user's system.

UC_02_TheaterLogin:
This use case is necessary to allow for users to connect to to their specific database. In order to access the movies/showtimes for a specific theater, the user must be authorized to do so. This is handled with UC_02.

UC_03_EmployeeLogin:
Employee Login is necessary to log all transactions and the employee who created them. Employee Login also provides another step of security for anyone accessing the theaters database. A malicious user may gained access to the Employee Login screen, but without correct Employee Login info, they won't be able to connect and manipulate the database.

UC_04_CreateNewSession:
A session is necessary in order to keep track of which employees processed which transactions. Each time an employee is logged in, a "session" is created so all activity within a session is logged with a unique employee id attached. This helps theater management have a clear understanding of who has done what.

UC_05_DisplayMainUI:
This is necessary to accurately display the contents of a theaters database. Every UI is unique to the theater and it's requirements derived by the theaters Movies/Auditoriums/Showtimes. This use case shows how the UI is derived via the Database. This is necessary every time a new session is created, as each theater has different movies/showtimes.

UC_06_BeginNewTransaction:
A "Transaction" object is required every time a new transaction is administered. A transaction can only hold Tickets. A new transaction is created after a transaction is processed or cancelled. When the program loads, a New Transaction is created. The "CurrentTransaction" object is important to maintaining an organized record of transactions. Each transaction has a unique ID which can later be recalled when a refund is needed.

UC_07_AddTicketToTransaction:
This use case is an absolute necessity as this allows for tickets to be added to the Current Transaction. The entire functionality of this system revolves around this Use Case.

<u>UC_08_RemoveTicketFromTransaction:</u>
Just as it is important to add tickets, the system also needs to be able to remove tickets from Current Transactions that have yet to be processed. This allows for a more modifiable Current Transaction.

<u>UC_09_CancelTranscation:</u>
Cancel Transaction is needed to clear Transaction quickly. If a Current Transaction has 10 tickets in the ticket window, the user would have to remove each ticket one by one, via UC_08_RemoveTicketFromTransaction. This allows for a Transaction to be cleared, regardless of its contents. Important use case, as it allows for quicker usability.

<u>UC_10_ProcessTransaction:</u>
Required to process a transaction. This Use Case is instrumental in modeling an accurate state of the theater and its database. When a transaction is processed, The ticket value is added to the Theaters Sales, and the Tickets that were sold are decremented from the Theater's Ticket Database.

<u>UC_11_RefundTransaction:</u>
This is necessary to allow for customers to get a refund. Most importantly, when a refund is offered, new tickets are added to the Movie's Ticket Database. Refunding helps reflect the accurate amount of seats actually filled. (ie. A customer could buy all the tickets for a showtime, Selling Out the show. If this customer refunded their transaction the system needs to add all of the refunded tickets back to the AVAILABLE tickets.

<u>UC_12_TimeOut_Screen:</u>
An important security feature. If a user walks away from their interface, the system is left open to vulnerabilities. With a time-out, if the user hasn't interacted after a certain time, it locks the system out until interacted with. After being locked out, the employee is required to log-in again to using their Employee ID.

<u>UC_13_LogOutEmployee:</u>
As multiple employees may use one system, an employee log-out functionality is very important. This allows for employees to log in and out of the system, without having to reconnect to the theaters database each time.

<u>UC_14_LogOutTheater:</u>
LogOutTheater allows for an extra level of security while your system is being operated. It also allows for the ability to easily switch between different theater databases upon Logging Out.

<u>UC_15_CloseApplication:</u>
The program must have a way for the user to terminate the system, from the UI.

LoginObject:
The LoginObject encapsulates both the TheaterLoginObject as well as the EmployeeLoginObject. This will allow the user to login to the theater via the TheaterLoginObject and then the EmployeeLoginObject. When a user logs out of the Theater, the LoginObject will be destroyed, and a new blank LoginObject will be created. The LoginObject should help save state information about what thater is currently logged in, and which employee is currently accessing the system.

TheaterLogin:
The TheaterLogin object contains the TheaterDatabaseURL, the Admin name, and the password. This object is passed to the Database Connector which will connect M.E.S.S. to the Theater Database URL that was entered. This object is necessary to authorize M.E.S.S. to get access to the proper database.

EmployeeLogin:
The employee login contains only one field, the Employee ID (empID). The user will input their empId into the Employee Login interface, and this empID will be checked against the TheaterDB's employee table. This object is necessary to add an extra step of security to the system.

Session:
A session object is necessary to keep track of the current transactions being made. By having a Session object that contains an empID, the system can link each transaction to an employeeID. This will allow for management to know which employee processed a specific transaction.
Also, when a session is created, the current Theater State is needed from the Theater Database.

TheaterState:
The TheaterState object is necessary to obtain all the accurate information needed to populate the UI. When a Session object is created, a TheaterState object is instantiated. This object will contain all auditoriums,movies and their respective showtimes. The GUI will will then extract the TheaterState from this object and display

it. Ie. The TheaterState returns with 3 auditoriums and their movies and showtimes. So the UI will create one button for each auditorium, with the correct movie title and showtimes displayed.

Transaction:
A transaction object is necessary to capture all ticket information for a specific transaction. Tickets are added to the transaction, and a total is created. A customer will pay, and then Transaction is then processed. From here, the transaction object is used to properly update the amount of tickets available in the theater database. The transaction object also updates the current sales amount. A Transaction object can also be cancelled. In this case, an empty Transaction is instantiated in its place.

Ticket:
A Ticket object holds all specific information for a single ticket purchase.
Ie. ([Movie] @ [Showtime] in [Auditorium] for Adult)
Tickets are added to Transactions, and they will only exist within transaction objects. A ticket helps display the correct Movie/Showtime/Auditorium information to the Transaction Window. This can be reviewed before processing a transaction. -- Each ticket has a ticketType: child, adult, student, senior, pass, etc. Each ticketType has a different price, so when a ticket is added to a transaction, it is compared to a price table that will accurately update the transaction total.

Auditorium:
An auditorium contains a theater as well as a list of showtimes. The auditorium object exists within a ticket object. The Auditorium object is necessary as it contains all Movie information that will be included in the Ticket. Instead of selling tickets by the movie, the M.E.S.S. sells tickets by the Auditorium, which can contain any movie object.

Movie:
The Movie object contains just the title and the rating of the movie. A Movie Object only exists within an Auditorium Object. (which only is instantiated in a Ticket Object.) The movie object simply is used to display the correct Movie title and rating within the Ticket and its transaction. The tickets are sold by auditorium and showtimes, rather than Movie title, so the Movie object acts more as a visual aid, when it is displayed in the TransactionView.

TransactionView:
The TransactionView is the window within the UI that displays the currentTransaction information. The TransactionView is necessary as it will display the data within the

current Transaction object, which is within the current session. The TransactionView will display all tickets currently in the transaction, as well as the current total. The TransactionView can be edited easily, allowing for the removal and addition of Transactions.

Rationale for System Architecture:

M.E.S.S. is designed on a Four-Tier System architecture. The system requirements demand an architecture that allows for future development and expansion. With this in mind, the system and its components need to be decoupled in a way that new modules and features can be added without issue.

The System contains four layers:
1. View: the GUI, which acts as the boundary object within our system.
2. Application: this layer processes all requests from the View and Database layers.
3. Database Application: this layer turns all application requests in to a proper SQL statement that will be sent to the database.
4. MySQL Database: contains all data that system interacts with.

The system is built around a Model-View-Controller architecture. The model being the database, the controller being the Application and Database Application layers, and the GUI being the view layer.

While one might conclude that M.E.S.S. actually are using a three-tier system architecture because the Application Layer and the Database Application Layer are both “controller” types, it is important to decouple the Application Layer and the Database Application Layer. In the future, the client may want to change their database from MySQL to something else. This would mean only the Database Application layer would need to be reworked in order to provide a connection between the application layer and the Database.

The application layer will be written in such a way that developers will only have to create new Database Interface layers to add support for more databases. If M.E.S.S. used a strict three-tier architecture, then the application layer would be tightly coupled with all of our Database Interface code. So, for this reason M.E.S.S. stresses a four-tier system architecture.

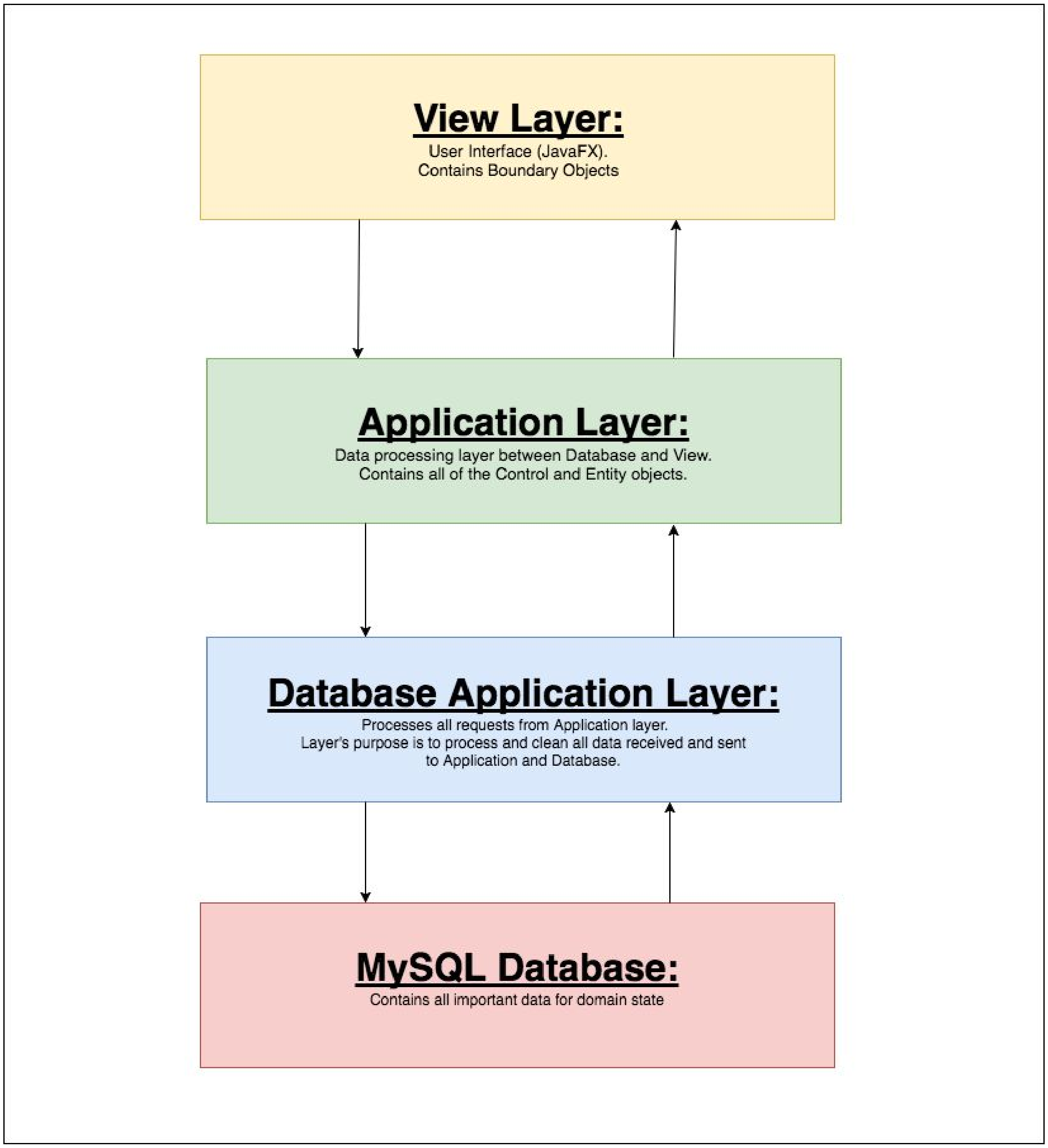
Requirements Trace Matrix: (as of 10/16/2016)

See Use Case Index to find corresponding Use Case for each Entry.

Entry #	Paragraph #	RTM Entry	Entry Type	Use Case
1	1	Shall be OS independent, working on any machine with the JVM.	SW	01
2	3	Shall have a Theater Login Interface that connects to a MySQL db via the internet	SW/HW	02
3	5	Shall have an employee Login interface that verifies an employee has access to the Theaters DB	SW	03
4	8	A new session shall be created when an employee successfully logs in.	SW	04
5	8	There shall be a MainUI from where all ticket vending occurs	SW/HW	05
6	9	There shall be a Currently Active Transaction created upon which all tickets will be added to.	SW	06
7	11	Shall be able to add ticket(s) to Currently Active Transaction	SW	07
8	11	Shall be able to remove any tickets added to Currently Active Transaction.	SW	08
9	11	Shall be able to Cancel any transaction currently active.	SW	09
10	11	Shall have the functionality to process a Currently Active Transaction	SW	10
11	11	Shall be able to offer a “refund” for any transaction in the Theater’s Transaction Database.	SW	11

12	7	Shall have time-out (lock) an employee's session if they have not interacted with system after a certain specified period of time.	SW	12
13	12	Shall have a log-out functionality for a currently logged in employee.	SW	13
14	12	Shall have a log-out functionality for a currently logged in Theater.	SW	14
15	13	Shall have an Exit Application functionality.	SW	15
16	13	Updated Modern UI	NTH	
17	13	Fast, instantaneous transactions	NF	

Software Architecture for M.E.S.S.


View Layer:
User Interface (JavaFX).
Contains Boundary Objects
Application Layer:
Data processing layer between Database and View.
Contains all of the Control and Entity objects.
Database Application Layer:
Processes all requests from Application layer.
Layer's purpose is to process and clean all data received and sent to Application and Database.
MySQL Database:
Contains all important data for domain state

Software Architecture Continued:

View Layer:

GUI implemented in JavaFX. The main interface that the user (employee) interacts with. Elements within the GUI contain the control objects from the application layer. When elements from the GUI are interacted with, their respective control objects/methods are triggered.

Application Layer:

Application layer contains control objects that are triggered by interactions with the View Layer. The application layer interacts with the View Layer and the Database Application Layer. The View Layer will trigger a control object from the application layer, which will then pass requests to the Database Application layer.

All updates that come from the database must pass through the application layer. For example, when the user logs their theater in, a request is sent to the database to retrieve all theater information. This data is then sent BACK through the database application layer, to the application layer, where it then updates the View Layer.

While many of the interactions are one way, such as sending a finished transaction to the database, most interactions will have a bidirectional flow.

Database Application Layer:

The Database Application Layer takes requests from the Application layer and interacts with the Database. The database application layer contains all methods necessary to interact with the database. The Database Application Layer is separate from the Application Layer because the Database Layer is specifically written for MySQL. In the future, a developer may choose to use a different database, and rather than having to completely rewrite the application layer, they will only have to rework the Database Application Layer. The DAL uses the JDBC Database Connector to interact with the MySQL database.

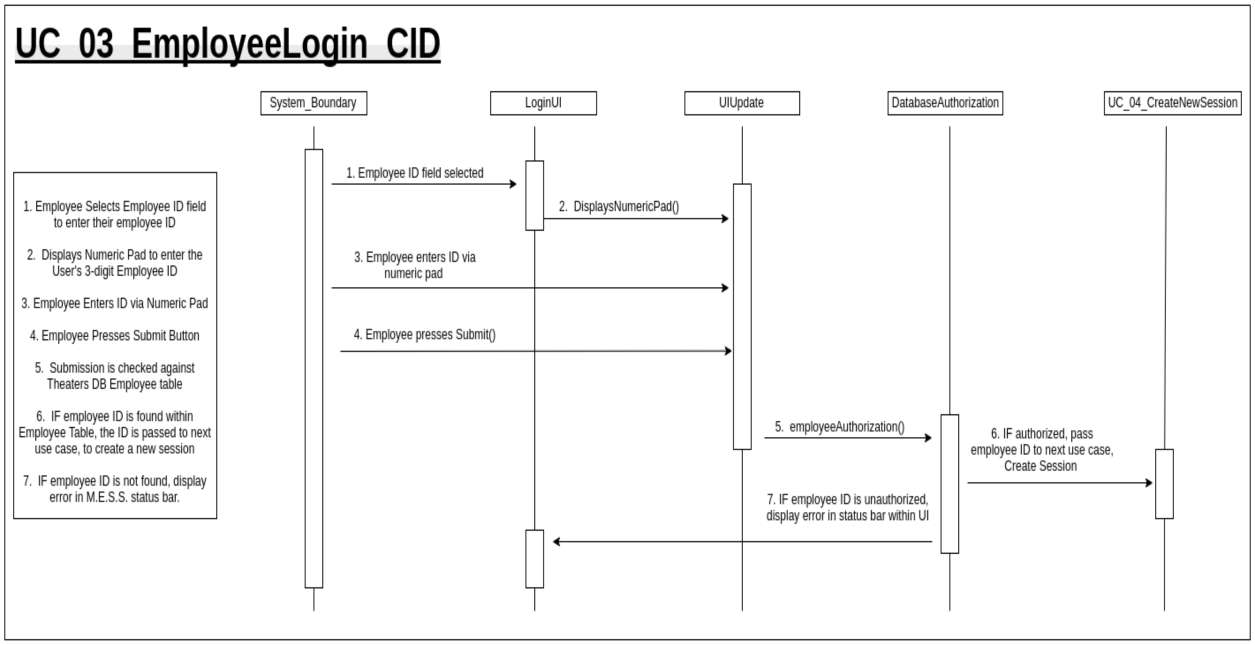
Data received by the Database will also be sent back via the Database Application Layer to the Application Layer. All updates coming from the Database to the Application Layer will be passed to the View layer providing an update to the UI.

MySQL Database:

Contains all data relevant to theater. The database is accessed via the Database Application Layer. The MySQL Database is the only system that exists outside of M.E.S.S.

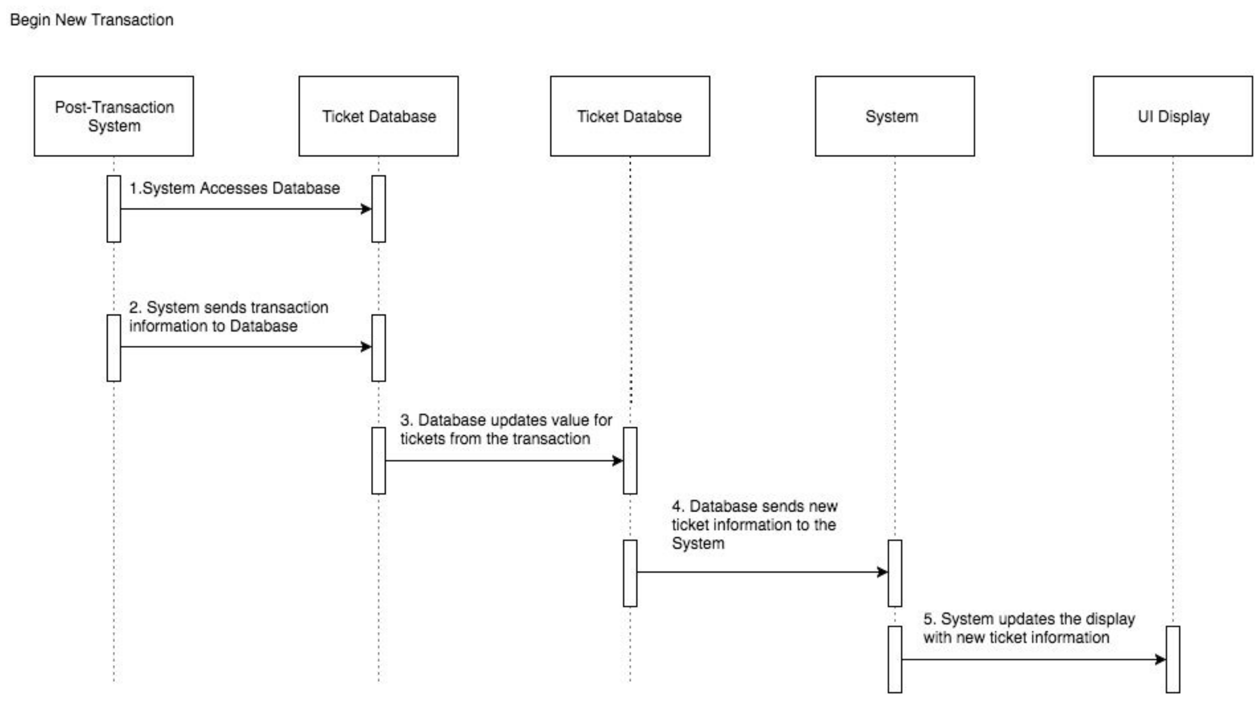
Category Interaction Diagrams:

UC_03_EmployeeLogin_CID


UC_03_EmployeeLogin_CID
System_Boundary
LoginUI
UIUpdate
DatabaseAuthorization
UC_04_CreateNewSession
1. Employee ID field selected
2. DisplaysNumericPad()
3. Employee enters ID via numeric pad
4. Employee presses Submit()
5. employeeAuthorization()
6. IF authorized, pass employee ID to next use case, Create Session
7. IF employee ID is unauthorized, display error in status bar within UI

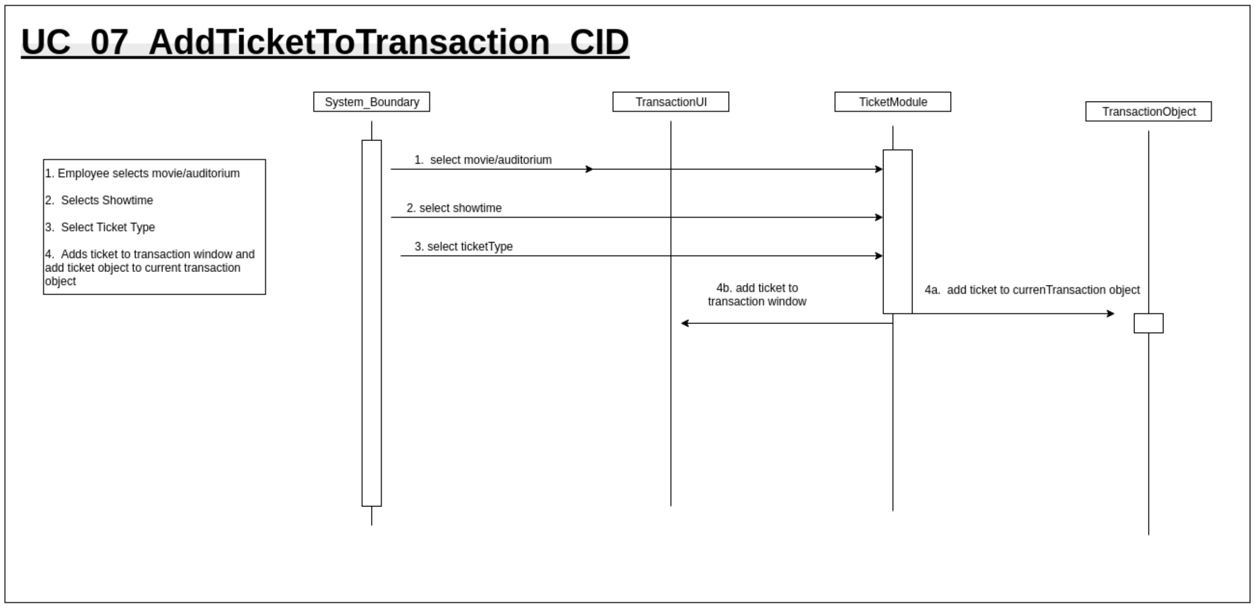
1. Employee Selects Employee ID field to enter their employee ID
2. Displays Numeric Pad to enter the User's 3-digit Employee ID
3. Employee Enters ID via Numeric Pad
4. Employee Presses Submit Button
5. Submission is checked against Theaters DB Employee table
6. IF employee ID is found within Employee Table, the ID is passed to next use case, to create a new session
7. IF employee ID is not found, display error in M.E.S.S. status bar.

UC_06_BeginNewTransation_CID

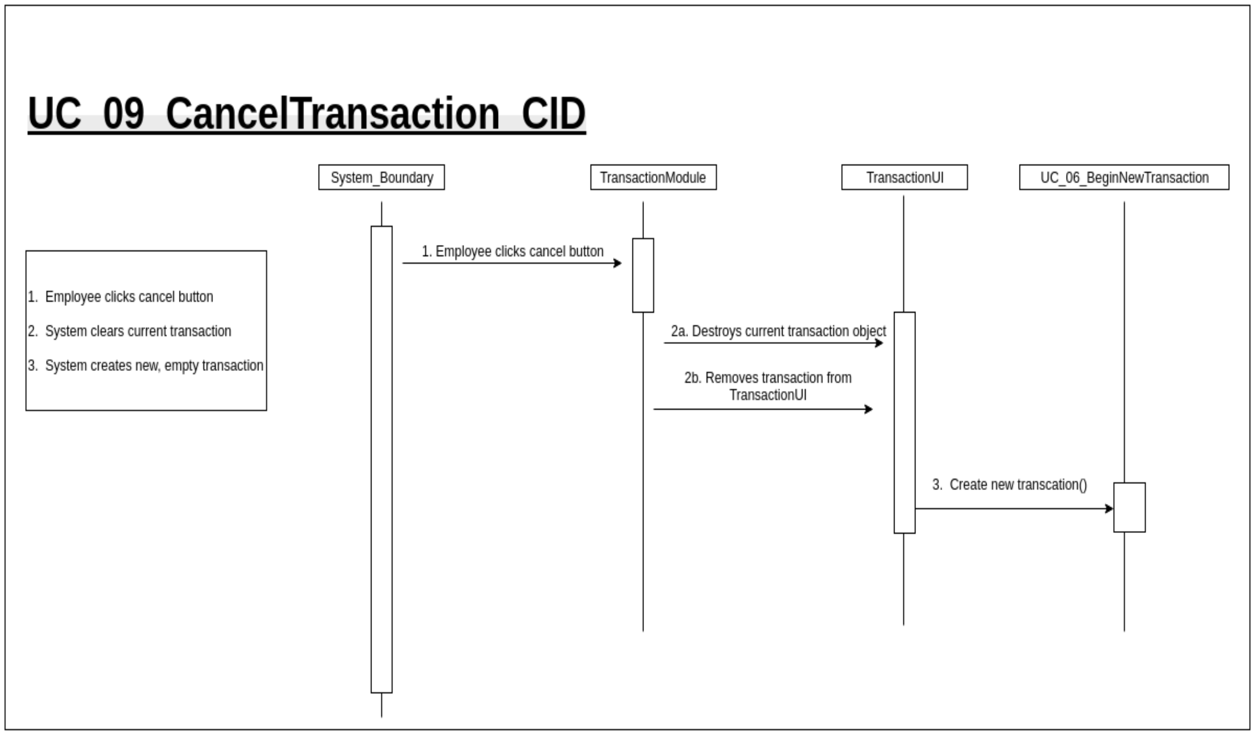

Begin New Transaction
Post-Transaction System
Ticket Database
Ticket Databse
System
UI Display
1.System Accesses Database
2. System sends transaction information to Database
3. Database updates value for tickets from the transaction
4. Database sends new ticket information to the System
5. System updates the display with new ticket information

1. **System Accesses Database**
2. **System sends transaction information to Database**
3. **Database updates value for tickets from the transaction**
4. **Database sends new ticket information to the System**
5. **System Updates the display with new ticket information**

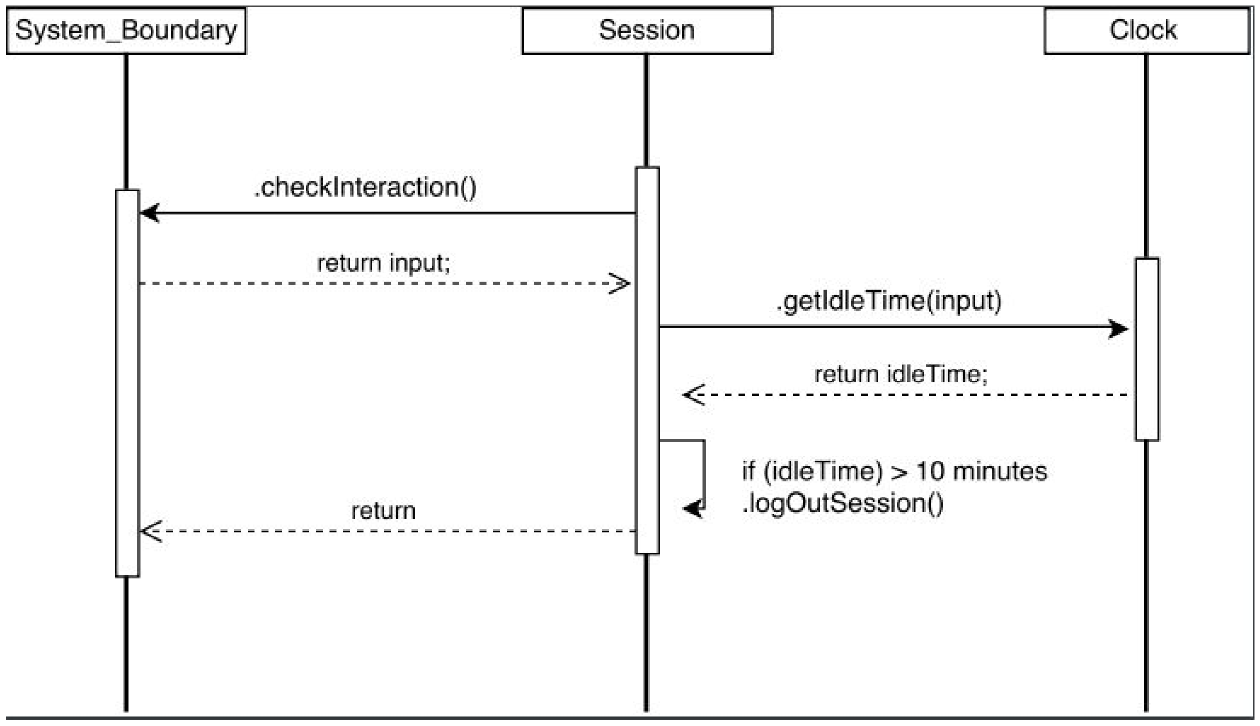
UC_07_AddTicketToTransaction_CID


UC_07_AddTicketToTransaction_CID
System_Boundary
TransactionUI
TicketModule
TransactionObject
1. Employee selects movie/auditorium
2. Selects Showtime
3. Select Ticket Type
4. Adds ticket to transaction window and add ticket object to current transaction object
1. select movie/auditorium
2. select showtime
3. select ticketType
4b. add ticket to transaction window
4a. add ticket to currenTransaction object

UC_09_CancelTransaction_CID


UC_09_CancelTransaction_CID
System_Boundary
TransactionModule
TransactionUI
UC_06_BeginNewTransaction
1. Employee clicks cancel button
2. System clears current transaction
3. System creates new, empty transaction
1. Employee clicks cancel button
2a. Destroys current transaction object
2b. Removes transaction from TransactionUI
3. Create new transcation()

UC_12_ScreenTimeOut



1. **Employee stops interacting with the system for 10 minutes.**

2. **System locks most interactions, requiring the employee to input their ID and pin to continue transactions**

Work Share Document: (10/23/2016)

Kris Hermstad:
Team Lead, Database & Application Layer Development

Lindsay Stone:
User, UI Layer Development

Emily Horton:
Client, UI Layer Development

Kirtan Patel:
Technical Writer, Application Layer Development

Jack Martin:
Product Support, Database Layer Development

MESS: GANTT for 10/23/2016

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Object Design	Function Point Cost,Interface design ,Object relationship, Updates to the task and role charts	14	10/2	10/16	YES
Project Rationale	All project rationale, CIDs, system architecture used, etc	7	10/16	10/23	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delivered by End Date?
Kris Hermstad (Team Lead)	CID for use case, organize deliverable	7	10/16	10/23	Yes
Jack Martin	CID for use case	7	10/16	10/23	Yes
Emily Horton	CID for use case	7	10/16	10/23	Yes

Lindsay Stone	CID for use case	7	10/16	10/23	Yes
Kirtan Patel	CID for use case	7	10/16	10/23	Yes
Next Deliverable:	Due:				
Test Document	10/30/2016	7	10/23	10/30	

PREVIOUS GANTTS

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Object Design	Function Point Cost,Interface design ,Object relationship, Updates to the task and role charts	14	10/2	10/16	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	System Architecture, Object Design, CIDs, Deliverable 3	14	10/2	10/16	Yes
Jack Martin	Object Design	14	10/2	10/16	Yes
Emily Horton	Object Design & PCI	14	10/2	10/16	Yes

Lindsay Stone	Object Design & PCI	14	10/2	10/16	Yes
Kirtan Patel	Object Design & PCI	14	10/2	10/16	Yes
Next Deliverable:	Due:				
Project Rationale	10/23/2016	7	10/16	10/23	

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delivered by End Date?
Kris Hermstad (Team Lead)	Use Cases, Database Selection, Deliverable 3	16	9/18	10/2	Yes
Jack Martin	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Emily Horton	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Lindsay Stone	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes
Kirtan Patel	Use Cases & Interaction Diagrams	16	9/18	10/2	Yes

Next Deliverable:	Due:				
Object Design	10/9/2016	7	10/2	10/9	

M.E.S.S. Gantt for 9/18/2016					
Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delievered by End Date?
Kris Hermstad (Team Lead)	Draft Problem Statement, RTM, and Gantt	3	9/15	9/18	Yes
Jack Martin	Database E-R Diagram	10	9/8	9/18	Yes
Emily Horton	Being drafting UI design mockups	16	9/2	9/18	Yes
Lindsay Stone	Draft UI mockups, experiment with JavaFX Scene Builder, create PERT chart	3	9/2	9/18	Yes
Kirtan Patel	Forumlate Documentation Standard for all project documents	16	9/2	9/18	Yes

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

M.E.S.S. Deliverable #6

Test Document and Code So Far

Software Engineering: Tuesday/Thursday 5:30pm w/ Mr. Rao

Due: 11.6.2016

TEAM JEKKL (GROUP #9)

Kris Hermstad: Team Lead, Database Developer

Emily Horton: Client, UI Developer

Kirtan Patel: Technical Writer, Application Developer

Lindsay Stone: User, UI Developer

Jack Martin: Product Support, Database Developer

Contains:

Requirements Trace Matrix: (as of 11/6/2016)

See Use Case Index to find corresponding Use Case for each Entry.

Entry #	Paragraph #	RTM Entry	Entry Type	Use Case
1	1	Shall be OS independent, working on any machine with the JVM.	SW	01
2	3	Shall have a Theater Login Interface that connects to a MySQL db via the internet	SW/HW	02
3	5	Shall have an employee Login interface that verifies an employee has access to the Theaters DB	SW	03
4	8	A new session shall be created when an employee successfully logs in.	SW	04
5	8	There shall be a MainUI from where all ticket vending occurs	SW/HW	05
6	9	There shall be a Currently Active Transaction created upon which all tickets will be added to.	SW	06
7	11	Shall be able to add ticket(s) to Currently Active Transaction	SW	07
8	11	Shall be able to remove any tickets added to Currently Active Transaction.	SW	08
9	11	Shall be able to Cancel any transaction currently active.	SW	09
10	11	Shall have the functionality to process a Currently Active Transaction	SW	10
11	11	Shall be able to offer a “refund” for any transaction in the Theater’s Transaction Database.	SW	11
12	7	Shall have time-out (lock) an employee’s session if they have not interacted with system after a certain specified period of time.	SW	12
13	12	Shall have a log-out functionality for a currently logged in employee.	SW	13
14	12	Shall have a log-out functionality for a currently logged in Theater.	SW	14
15	13	Shall have an Exit Application functionality.	SW	15
16	13	Updated Modern UI	NTH	
17	13	Fast, instantaneous transactions	NF	

TEST CASES:

Test Case Identifier	TheaterLogin
Test Location	TheaterLogin UI, upon loading application
Feature to be tested	Form submission should return true/false depending on results of Database connection
Feature Pass/Fail Criteria	The test passes if true or false are returned. The test fails if an exception is thrown from unsuccessful login.
Means of Control	1. The submit/login button is pressed on the TheaterLogin UI. the JDBC "setConnection()" method is called within the JDBC Connector.
Data	2. The form data is collected and submitted and stored in the applications Session object. 3. If the submission fails, the form is cleared.
Test Procedure	The test is started by first filling in info into the UI's form. (Theater URL, Admin, and Password) -> the User will then press submit. The process should only take 2 to 3 seconds.
Special Requirements	The forms must not be empty before submitting, otherwise user will be asked to fill in information before submitting. - No special test stub needed.

Test Case Identifier	EmployeeLogin
Test Location	EmpLogin UI, upon logging in via TheaterLogin
Feature to be tested	Form submission should return true/false depending on results of comparison to Theater's employee database table.
Feature Pass/Fail Criteria	The test passes if true or false are returned. The test fails if an exception is thrown from unsuccessful employee login.
Means of Control	1. Test user interacts with a numeric pad to enter their test case id into the EmployeeID field. 2. The
Data	2. The form data is collected and submitted and stored in the applications Session object, as previously done in TheaterLoginUI test case. . 3. If the submission fails, the form is cleared.
Test Procedure	Test is initialized by user entering their test ID via the Numeric Pad. After entering the id, the Employee id is submitted via the login button. The Employee authorization should only take 1 to 2 seconds.
Special Requirements	The test stub Employee is needed to test the lookup of id's against the user submitted id.

Test Case Name: Begin new Transaction

Test Location: Main Ticket UI

Input: A ticket sale will be completed through the system.

Oracle: The ticket value presented by the UI will be consistent with the ticket value in the database. These values will have been reduced by the exact amount that the ticket sale specified.

Log: A sale of one ticket is completed through the system. The Database value for this ticket has been reduced by one. The UI displays the new ticket value, matching the database.

Test Case – Add a Ticket

Attributes	Description
name	Add a Ticket
location	currentTransaction
input	Ticket
oracle	currentTransaction with new ticket added
log	Ticket added to currentTransaction

Test Case – Remove a Ticket

Attributes	Description
name	Remove a Ticket
location	currentTransaction
input	Ticket to be removed
oracle	currentTransaction with a ticket removed
log	Ticket removed from currentTransaction

Test Case Identifier	CancelTransaction
Test Location	Transaction UI
Feature to be tested	Cancel Button should remove every order in the queue
Feature Pass/Fail Criteria	The test passes if all the orders are removed from the current queue
Means of Control	1. The CancelTransaction only shows up once an order has been inputted into the system.
Data	2. The orders are all removed from queue and memory
Test Procedure	The test is started by pressing the cancel button. The test will bring up the home screen again and the orders will be removed. The test should take no more than 2 seconds
Special Requirements	There should be orders in the queue prior to pressing cancel

Name: TimeOut

Location: MainTicketUI

Input: The startClock() method is called from test driver startClock with the argument “10:00” passed to initialize the starting value for test purposes.

Oracle: Because ten minutes is its boundary, startClock() should call displayLogin(), which, in the test driver should just set loginDisplayed = true.

Log: The test will return the value of loginDisplayed at its conclusion. True indicates a successful test; false a failed test.

Test Case Identifier	LogOutEmployee
Test Location	Main UI
Feature to be tested	Logging out employee from main UI

Feature Pass/Fail Criteria	The test passes if the user is returned to the employee login screen after pressing log-out.
Means of Control	1. A “logout” button on the Main TicketVendorUI
Data	2. The Session object currently holding the “empId” from the previously logged in employee is cleared out.
Test Procedure	The test begins by pressing the Log-Out button the main ticket vendor UI. The user should immediately be logged out and returned to the Employee Login screen. This should be instantaneous. Time To Execute < 2 seconds.
Special Requirements	An employee should be logged in, with a non-null value in the employeeID field of the session object.

Test Case Identifier	LogOutTheater
Test Location	EmpLoginUI
Feature to be tested	Logging out currently connected TheaterDatabase
Feature Pass/Fail Criteria	The test passes if the test returns the user to the TheaterLogin UI, disconnecting from the database. The test fails if the screen does not change from the EmpLogin to TheaterLogin, and also fails if the Theater Database Connector’s info is not set to null.
Means of Control	1. A “log out of theater” button on the EmployeeLogin screen controls the logging out of the Theater database.
Data	2. The session currently holding the TheaterInfo is reset to no value (Null). This will require the user to connect to the theater database again.
Test Procedure	Test begins by user pressing the “Log-Out of Theater” button on the Employee Login screen. The system should immediately disconnect the database and return the user to the Theater Login screen. Execution time < 2 seconds.
Special Requirements	A theater should be currently logged in and connected via JDBC. (currently no database test stub, just use production database)

Work Share Document: (11/6/2016)

<u>Kris Hermstad:</u>
Team Lead, Database & Application Layer Development

<u>Lindsay Stone:</u>
User, UI Layer Development

<u>Emily Horton:</u>
Client, UI Layer Development

<u>Kirtan Patel:</u>
Technical Writer, Application Layer Development

<u>Jack Martin:</u>
Product Support, Database Layer Development

MESS: GANTT for 11/6/2016

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Object Design	Function Point Cost,Interface design ,Object relationship, Updates to the task and role charts	14	10/2	10/16	YES
Project Rationale	All project rationale, CIDs, system architecture used, etc	7	10/16	10/23	YES
Test Document	All test cases, rationale	14	10/23	11/6	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delivered by End Date?
Kris Hermstad (Team Lead)	Test Cases, organize deliverable	14	10/23	11/6	Yes
Jack Martin	Test Case	14	10/23	11/6	Yes
Emily Horton	Test Case, UI Design	14	10/23	11/6	Yes
Lindsay Stone	Test Case, UI Design	14	10/23	11/6	Yes
Kirtan Patel	Test Case	14	10/23	11/6	Yes

Next Deliverable:	Due:				
Final Project	11/29/2016	23	11/6	11/29	

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

Rationale for Test Cases:

TheaterLogin
This test case is necessary to ensure the Theater Database is properly connected to, via JDBC.   Because the login method only returns a boolean value to provide access to the user, the test case only needs to check the value of the returned boolean value. Multiple types of inputs need to be tested to ensure the form properly processes and parses the input data.

EmployeeLogin
This test case is necessary to ensure the EmployeeLogin is properly authorized. The test case is necessary to check that only valid input has been submitted. This test case will ensure that proper, secure login for employees is in place. Because the input is only limited to the numeric pad, the user will not be able to enter via a keyboard. This should allow the possible inputs to be lower than if a keyboard input as available.

BeginNewTransaction
This test case is necessary to ensure a new Transaction is created after [a] an employee has just logged in or [b] a transaction was just completed or canceled. This test passes when the previous transaction is processed and added to the database, and a new BLANK transaction is instantiated in its place. The test will fail if the previous transaction remains after it was supposed to have been processed.

AddATicket
This is a simple test case to see that when a certain ticket is added to the transaction, the correct information is accurately reflected in the currentTransaction. The test case is necessary to determine whether the proper tickets are being recorded during each transaction.

RemoveATicket
Like AddATicket above, this is a simple test case to see that a specific ticket can be removed from the current transaction window. The test passes when a ticket is successfully removed from the currentTranscation object and the currentTransaction UI window. The test fails if data remains after an attempted removal.

CancelTransaction

This test case ensures that the current Transactions data is properly cleared and a new transaction object is instantiated. The test assumes the system already has a non-null currentTransaction. If the test case is initialized without having a null transaction, it should not negatively impact the functionality of the application. The only real fail state is when a transaction is cancelled, but the transactions state is still set within the currentTransaction object.

LogOutEmployee

Test case is needed to ensure that the employee is properly logged out and the employeeID of the current session is set to null, or 0. The fail state is when the user clicks log-out but the scene does not change. As well, the test case will also fail if the user clicks log-out, and while the scene does change, the Employee info is still stored in the current session. The test only passes when the employee info is removed, AND the scene has been set to the original TheaterLoginUI.

LogOutTheater

This test case ensures that the TheaterDb connection is properly severed, and the session object is cleared of the current Theater Info. The test fails when the user clicks log-out of theater, but the scene does not change, and the session info still contains the previous theaters info. The test will only pass when upon clicking “Log-Out of Theater” the scene is changed to the TheaterDB LoginUI, and the current session object is reinstantiated with a basic empty Session. If there is not a new session, and the session data isn’t clear, the test fails.

Note: The Test Cases used are Integration Tests. While each module is being unit tested by the developers, the Test Cases presented are a composite of the different “units” functioning together as a system. Each component within each test case is being tested as well. jUnit is being used to help automate the tests of the smaller modules, while the Integration tests are being performed by the QA team members.

UPDATED FUNCTION POINT COST ANALYSIS

Weighting Factor Estimate: (UFP)

Measurement Parameters							
	COUNT		**SIMPLE**	**AVERAG E**	**COMPLEX**		**TOTAL**
Number of User Inputs	27	X	3	4	6	=	116
Number of User Outputs	10	X	4	5	7	=	57
Number of User Inquiries	6	X	3	4	6	=	28
Number of Internal Files	5	X	7	10	15	=	86
Number of External Interface of Files	1	X	5	7	10	=	10
GRAND TOTAL (FP)	UFP					=	**297**

Calculation Table:

Function Type	**Amount**	**Functional Complexity**		**Complexity Totals**	**Function Type Totals**
ILFs	**3**	**Low**	**X 7 =**	**21**	
	2	**Average**	**X 10 =**	**20**	

	3	High	X 15 =	45	
					86

Function Type	Amount	Functional Complexity		Complexity Totals	Function Type Totals
ELFs		Low	X 5 =		
		Average	X 7 =		
	1	High	X 10 =		
					10

Function Type	Amount	Functional Complexity		Complexity Totals	Function Type Totals
EIs	22	Low	X 3 =	66	
	5	Average	X 4 =	20	
	5	High	X 6 =	30	
					116

Function Type	Amount	Functional Complexity		Complexity Totals	Function Type Totals
EOs	3	Low	X 4 =	12	
	2	Average	X 5 =	10	
	5	High	X 7 =	35	
					57

Function Type	Amount	Functional Complexity		Complexity Totals	Function Type Totals
EQs	2	Low	X 3 =	6	
	4	Average	X 4 =	16	
	1	High	X 6 =	6	
					28

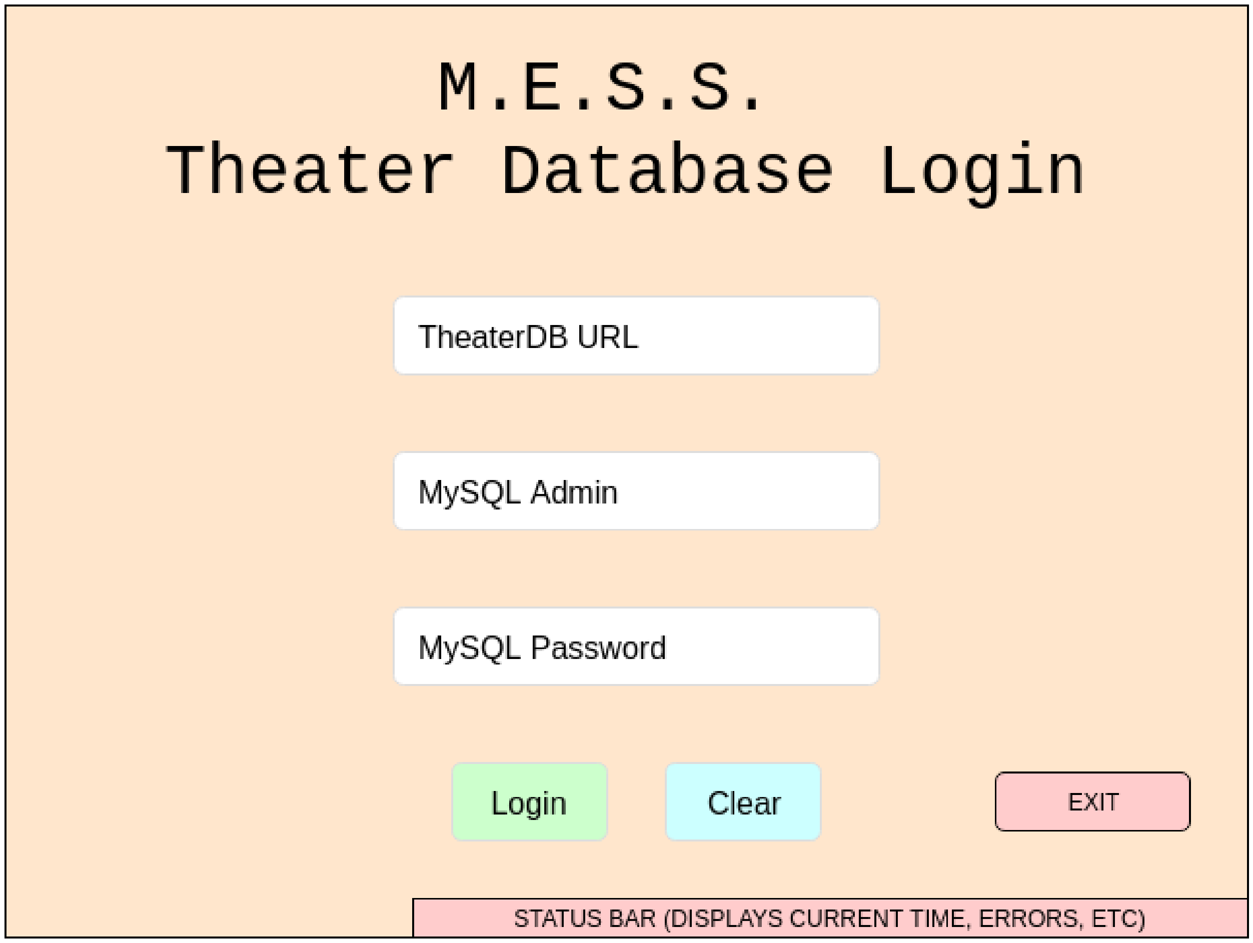
Category Ratings:

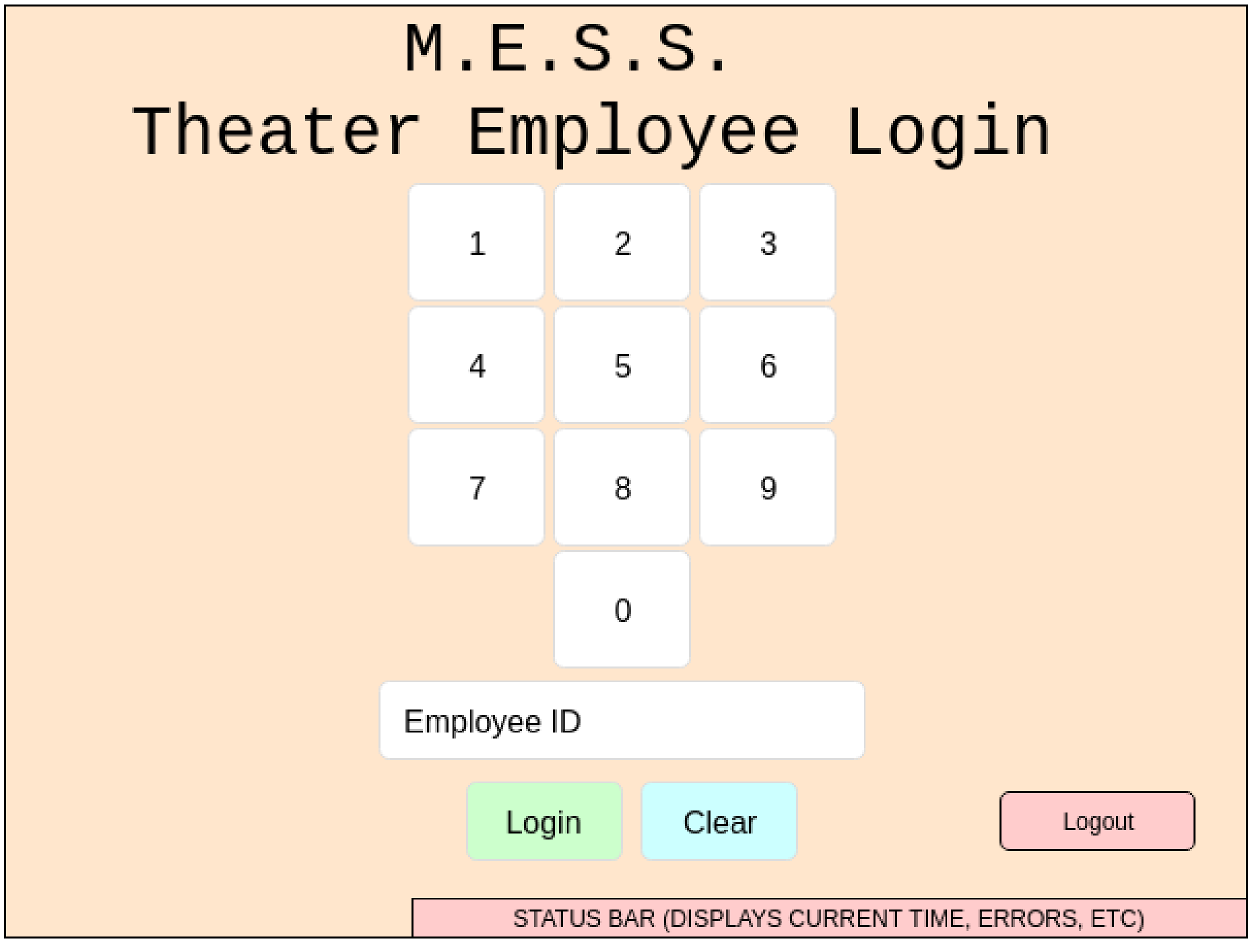
Category	Rating
1. Does the system require reliable backup and recovery?	1
2. Are data communications required?	2
3. Are there distributed processing functions?	2
4. Is performance critical?	4
5. Will the system run in an existing heavily utilized operational environment?	5
6. Does the system require on-line data entry?	4
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?	3
8. Are the master files updated on-line?	5
9. Are the inputs, outputs, files or inquiries complex?	3
10. Is the internal processing complex?	2
11. Is the code designed to be reusable?	4
12. Are conversion and installation included in the design?	5
13. Is the system designed for multiple installations in different organizations?	5
14. Is the application designed to facilitate change and ease of use by the user?	3
TOTAL SUM OF ALL CATEGORY RATINGS:	**48**

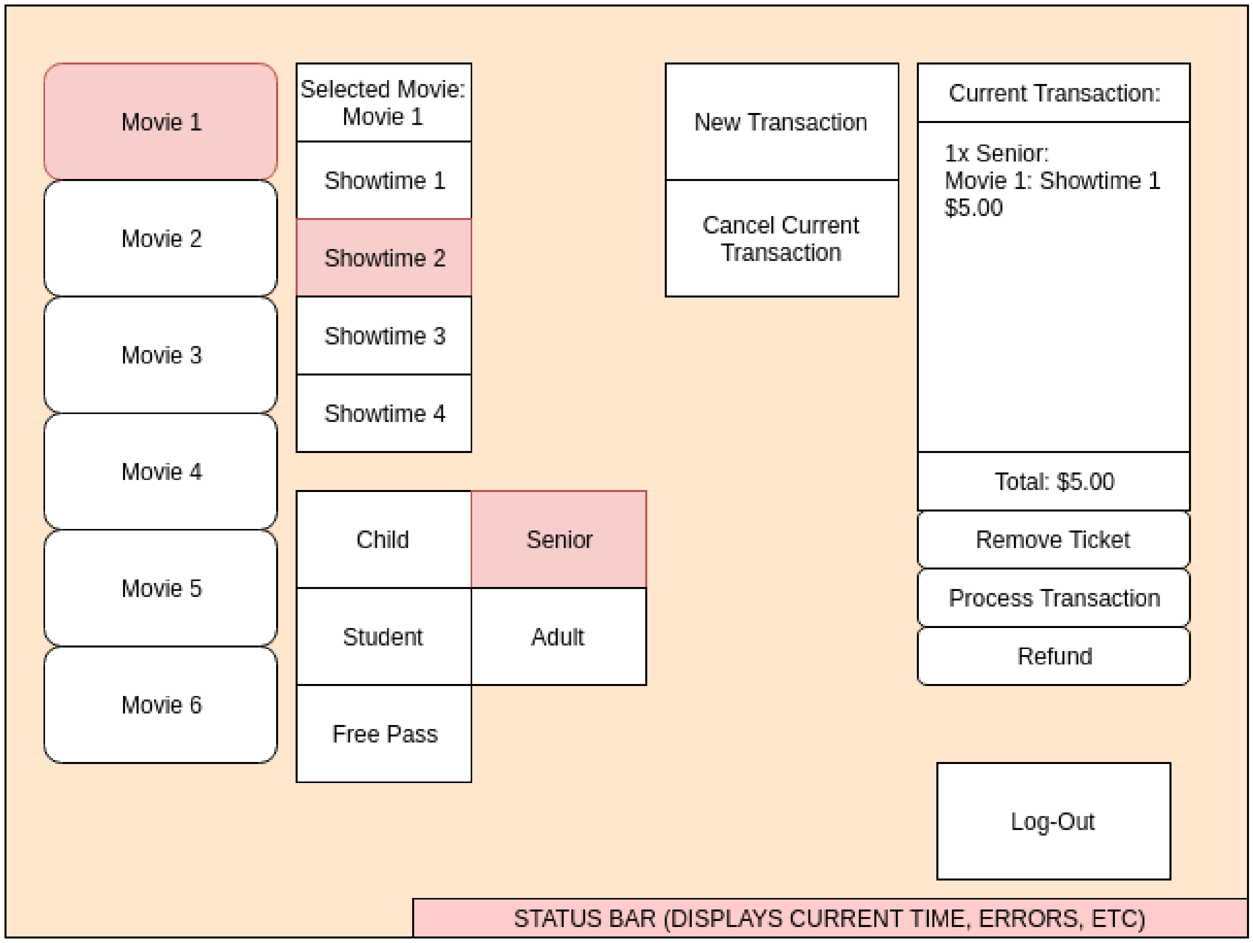
VAF = [0.65 + 0.01 * (48)]
VAF = 1.13
FPC = UFP * VAF
FPC = 297 * 1.13

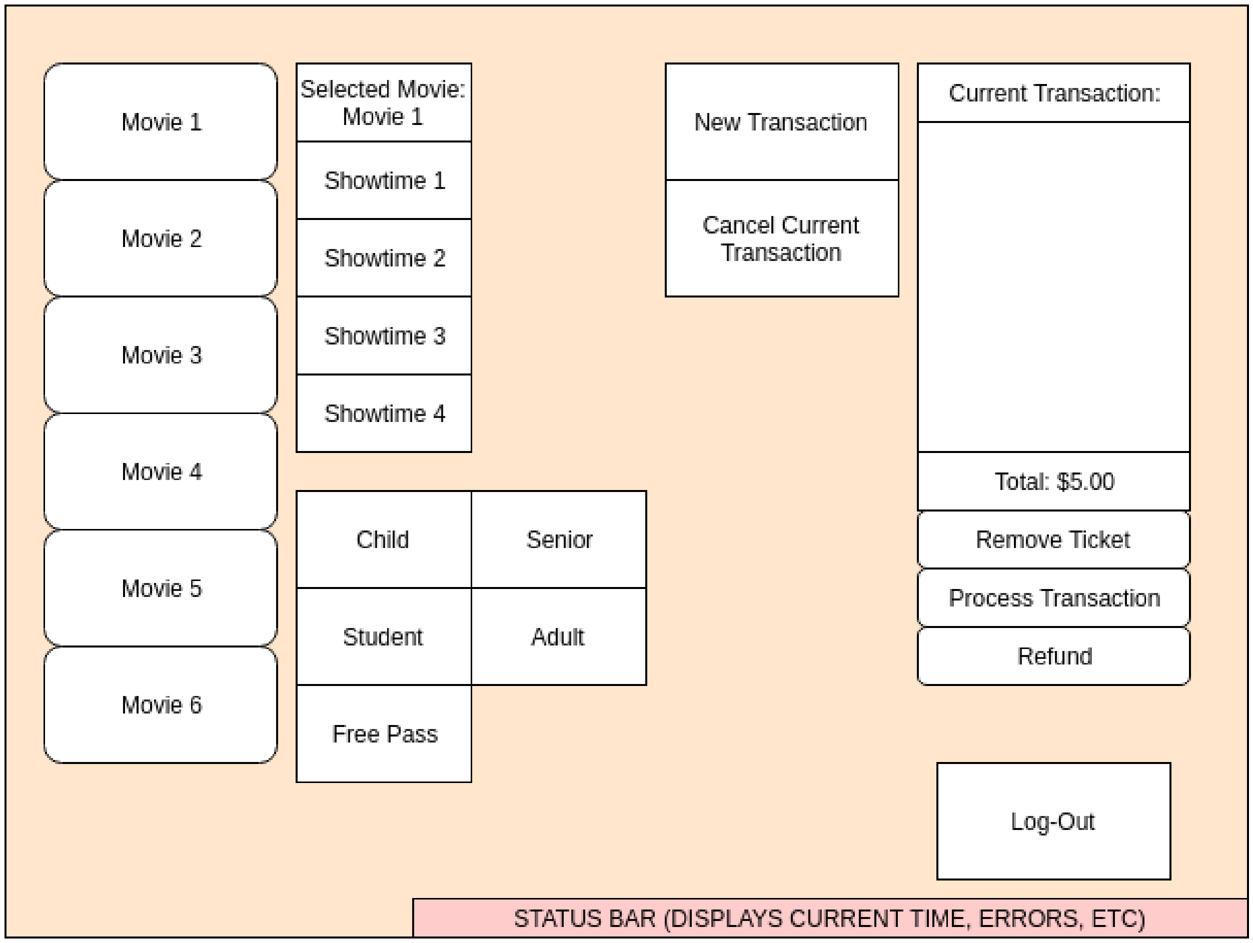
FPC = 335.61

Original Prototype




M.E.S.S.
Theater Employee Login
1
2
3
4
5
6
7
8
9
0
Employee ID
Login
Clear
Logout
STATUS BAR (DISPLAYS CURRENT TIME, ERRORS, ETC)


Movie 1
Movie 2
Movie 3
Movie 4
Movie 5
Movie 6
Selected Movie: Movie 1
Showtime 1
Showtime 2
Showtime 3
Showtime 4
Child
Senior
Student
Adult
Free Pass
New Transaction
Cancel Current Transaction
Current Transaction:
1x Senior:
Movie 1: Showtime 1
$5.00
Total: $5.00
Remove Ticket
Process Transaction
Refund
Log-Out
STATUS BAR (DISPLAYS CURRENT TIME, ERRORS, ETC)


Movie 1
Movie 2
Movie 3
Movie 4
Movie 5
Movie 6
Selected Movie: Movie 1
Showtime 1
Showtime 2
Showtime 3
Showtime 4
Child
Senior
Student
Adult
Free Pass
New Transaction
Cancel Current Transaction
Current Transaction:
Total: $5.00
Remove Ticket
Process Transaction
Refund
Log-Out
STATUS BAR (DISPLAYS CURRENT TIME, ERRORS, ETC)

Project Legacy

Development of MESS going forward should aim to incorporate many features that the development team was unable to implement for their initial RTM but are integral to the overall functionality of the product.

- Implement Ticket Sales for Future Showings beyond the Current Day.
- Track the current register's contents to inform managers of how much cash should be expected in the register at the end of a shift.
- Implement Debit/Credit payment transactions (currently only cash)
- Expand UI to be more dynamic. Currently, theaters can only load 6 auditoriums at a time. The UI should load as many buttons necessary to reflect the proper amount of auditoriums the Theater contains.
- Implement a safer way for the system to keep a user from adding a ticket from a showtime that is currently sold out.
- Refactor any heavily repeated code.
- Remove unused variables/methods.
- Continue development of modern UI design.
- Implement an administrator panel.

Above all, the project should always ensure it never steers far from its original purpose: An Intuitive, Fast, No-Nonsense Box Office Management Software.

Final Work Share Document: (11/29/2016)

Kris Hermstad:
Team Lead, Developer

Lindsay Stone:
User, UI Design and Development

Emily Horton:
Client, UI Design and Development

Kirtan Patel:
Developer, Technical Writer,

Jack Martin:
Database Layer Development

FINAL GANTT for 11/29/2016

Deliverables					
Task	Task Description	Days to Complete	Start Date	End Date	Delivered by End Date?
Project Contract	Team Lead to sign the overall document. Group members and the roles.	8	8/22	9/1	Yes
Requirement Elicitation	Requirements for the selected project,RTM,Task allocation chart, Explanation of terms	16	9/2	9/18	Yes
System Analysis and Design	Horizontal Prototype, updated RTM, Use Cases, and ID, Database used, WSD, updated Gantt chart, dictionary, rational use cases.	14	9/18	10/2	YES
Object Design	Function Point Cost,Interface design ,Object relationship, Updates to the task and role charts	14	10/2	10/16	YES
Project Rationale	All project rationale, CIDs, system architecture used, etc	7	10/16	10/23	YES
Test Document	All test cases, rationale	14	10/23	11/6	YES
Final Report	Final Report	24	11/6	11/29	YES
Team Member's Current Tasks					
Team Member	Task	Days To Complete	Start Date	End Date	Delivered by End Date?
Kris Hermstad (Team Lead)	Organize Final Deliverable, presentation	14	11/5	11/29	Yes
Jack Martin	Test Software	14	11/5	11/29	Yes

Emily Horton	Finish UI design, test	14	11/5	11/29	Yes
Lindsay Stone	Finish UI design, test	14	11/5	11/29	Yes
Kirtan Patel	Test, debug codebase, prepare deliverable	14	11/5	11/29	Yes

MySQL: The chosen Database solution for this project. MySQL is a database that will be cloud based, and will reflect the state of the client's theater.

Java: The programming language for which the entire M.E.S.S. will be written and developed in.

JavaFX: The UI library used within Java to create M.E.S.S.'s GUI.

JDBC: Database Driver to connect java program to MySQL database.

Point-of-Sale (POS): The physical computer/kiosk on which the M.E.S.S. software transactions occur

Employee: Employee refers to any "Employee" user who is interfacing with M.E.S.S. This would be the employee's of the Theater/Venue using M.E.S.S.

Manager: Manager is a form of Employee with specially authorized access to certain functions within M.E.S.S.

Show-times: These refer to specific instances of movies within an auditorium with certain start and end time properties. Each showtime is linked to an auditorium, a movie, and the amount of seats within the auditorium.

Auditorium: The specific locations within the Theater that each showtime takes place in. Each auditorium has a unique ID within the Theater's database.

Sales: Each ticket transaction is equivalent to a "sale" within the Theater's database. The sales are all logged in monitored within M.E.S.S.

As of 11/29/2016

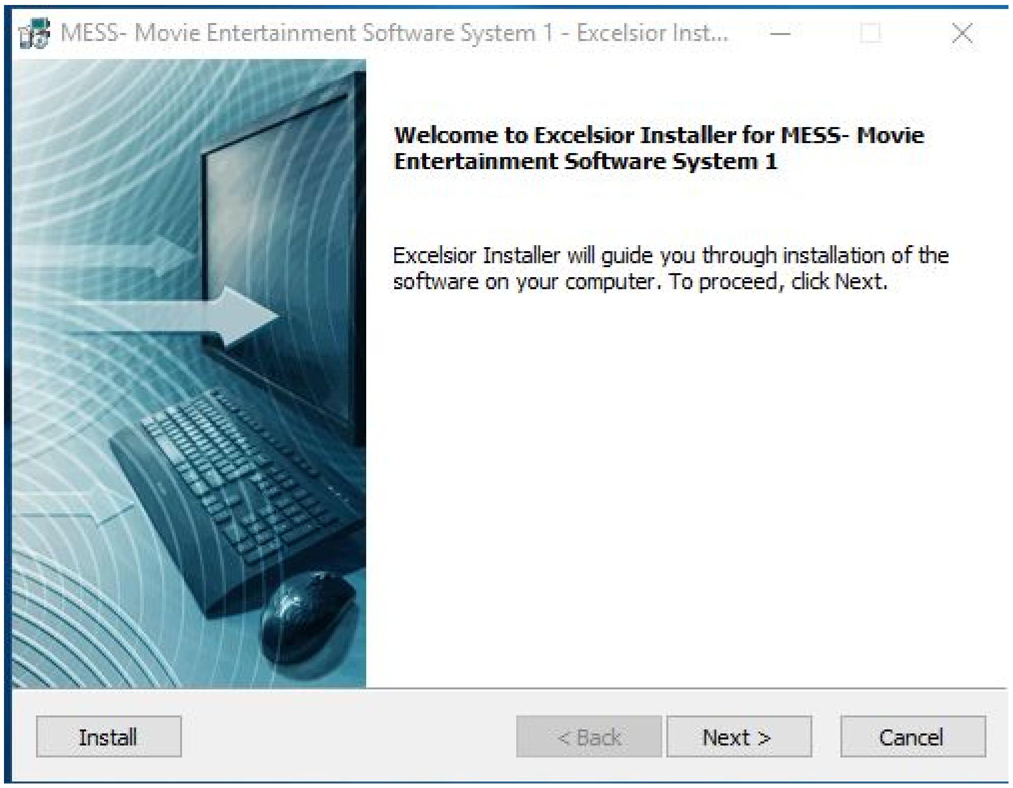
JEKKL

MESS User Guide

11/29/2016

1. Installation & Setup

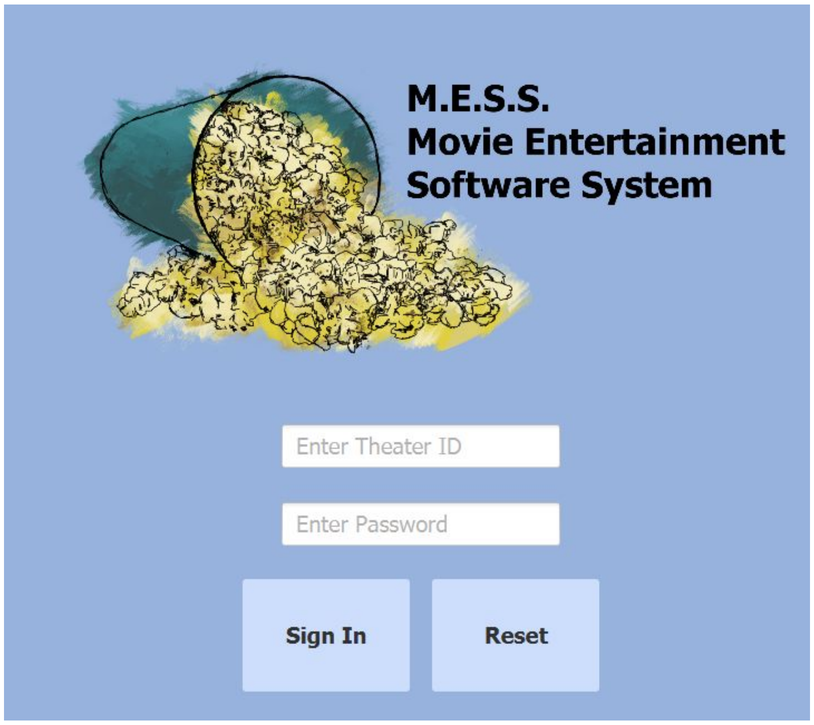
1. Run mess-setup.exe and Install to desired directory. (Default installs to Program Files/JEKKL/MESS)


MESS- Movie Entertainment Software System 1 - Excelsior Inst...
Install
< Back
Next >
Cancel

Welcome to Excelsior Installer for MESS- Movie Entertainment Software System 1

Excelsior Installer will guide you through installation of the software on your computer. To proceed, click Next.

2. Theater Login

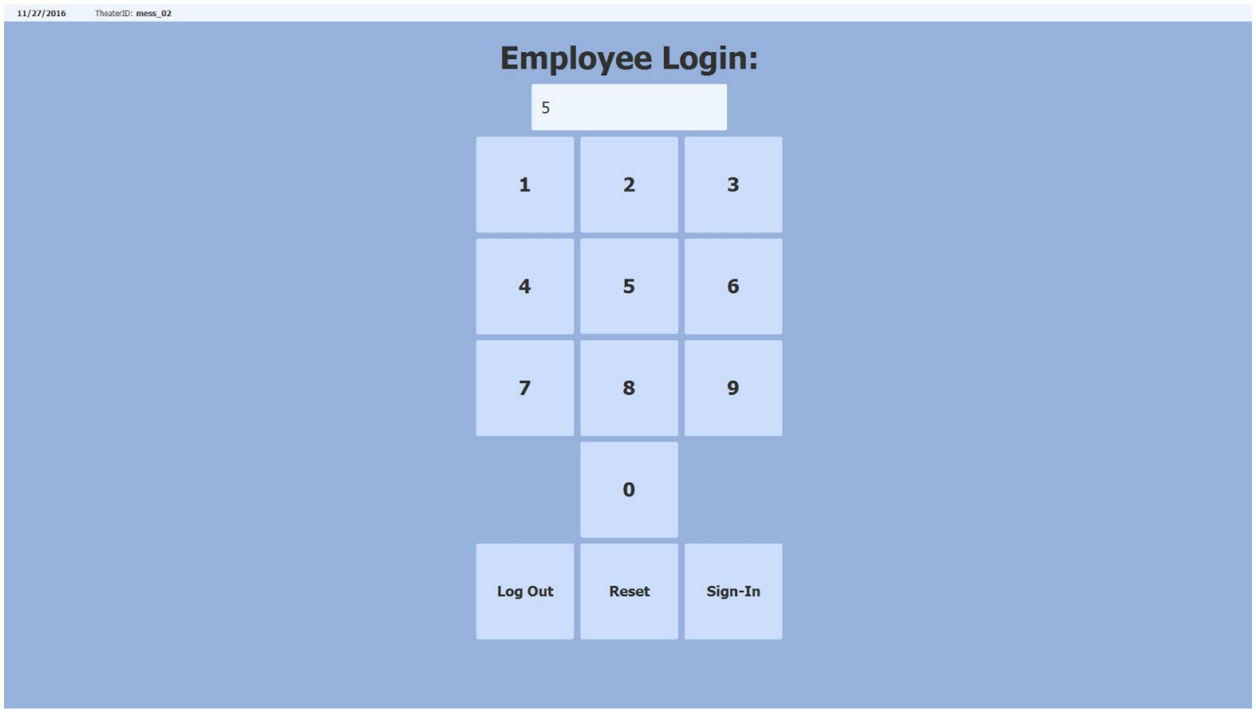


1. Enter unique Theater ID and Password into respective fields.
2. Press Sign-In to connect to Theaters Database.

Trouble Connecting?

- Check that Theater ID and Password are correct
- Check Internet Connection

3. Employee Login

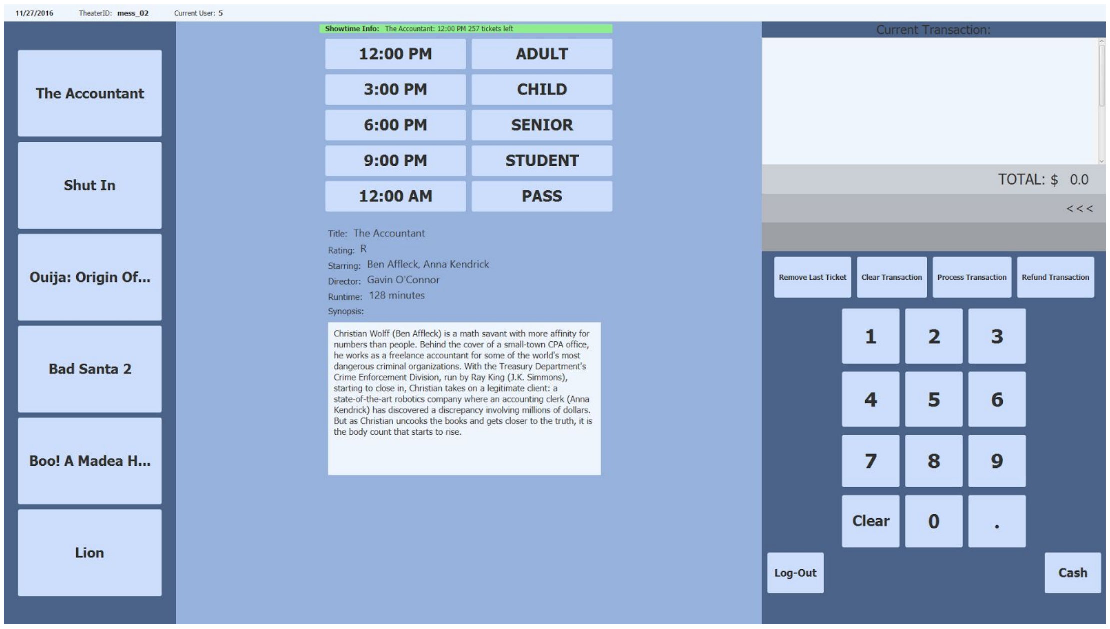


1. Enter Valid Employee ID
2. Press Sign-In

Trouble connecting?

- Verify Employee ID entered was valid.
- Check Internet Connection

4. Main Vendor Screen

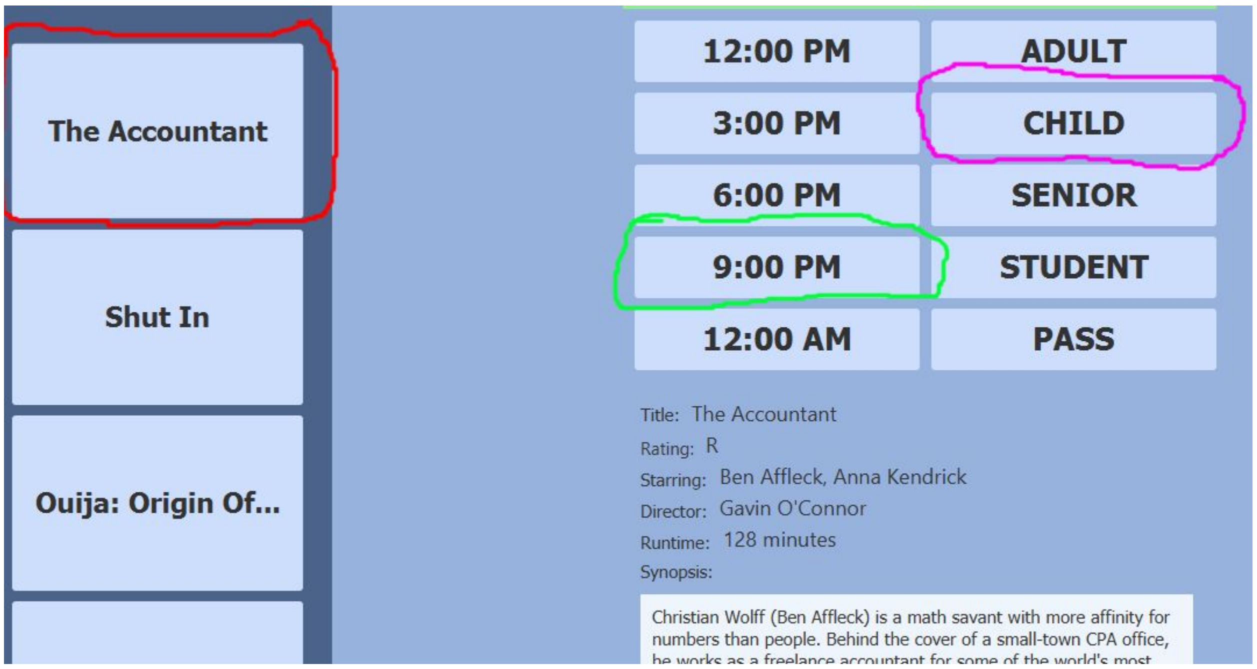


Title: The Accountant
Rating: R
Starring: Ben Affleck, Anna Kendrick
Director: Gavin O'Connor
Runtime: 128 minutes
Synopsis:
Christian Wolff (Ben Affleck) is a math savant with more affinity for numbers than people. Behind the cover of a small-town CPA office, he works as a freelance accountant for some of the world's most dangerous criminal organizations. With the Treasury Department's Crime Enforcement Division, run by Ray King (J.K. Simmons), starting to close in, Christian takes on a legitimate client: a state-of-the-art robotics company where an accounting clerk (Anna Kendrick) has discovered a discrepancy involving millions of dollars. But as Christian uncooks the books and gets closer to the truth, it is the body count that starts to rise.

1. Upon loading, the theater’s current state should be properly displayed in the UI.
2. The Left Panel contains all movies available under this Theater.
3. The Middle Panel provides information for specific movies.
4. The Middle Panel contains the controls to add tickets based on showtimes and ticket types.
5. The Right Panel is the “Register” where all transactions take place.
6. A status bar on top displays the current theater and employee logged in.

5. Add & Remove Tickets From Transaction:

Add Ticket

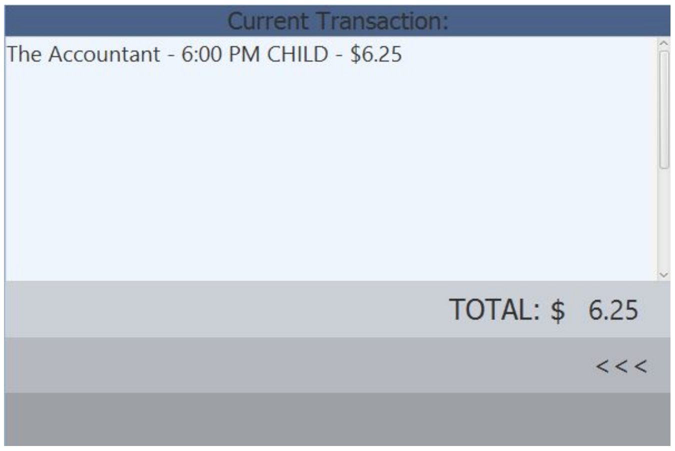


Title: The Accountant
Rating: R
Starring: Ben Affleck, Anna Kendrick
Director: Gavin O'Connor
Runtime: 128 minutes
Synopsis:

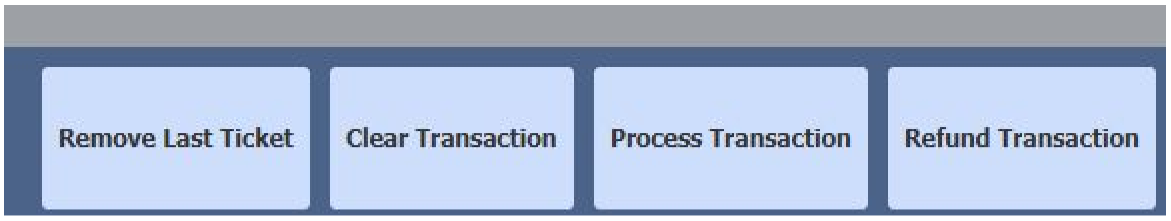
Christian Wolff (Ben Affleck) is a math savant with more affinity for numbers than people. Behind the cover of a small-town CPA office, he works as a freelance accountant for some of the world's most

1. Select a Movie
2. Select A Showtime
3. Select Ticket Type

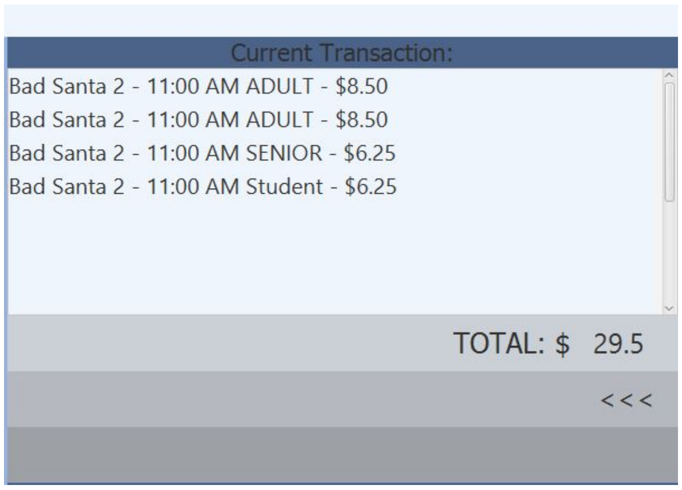
Ticket will then be added to Transaction Window


Current Transaction:
The Accountant - 6:00 PM CHILD - $6.25
TOTAL: $ 6.25

To Remove Last Ticket in Transaction, press Remove Last Ticket in Right Control Panel



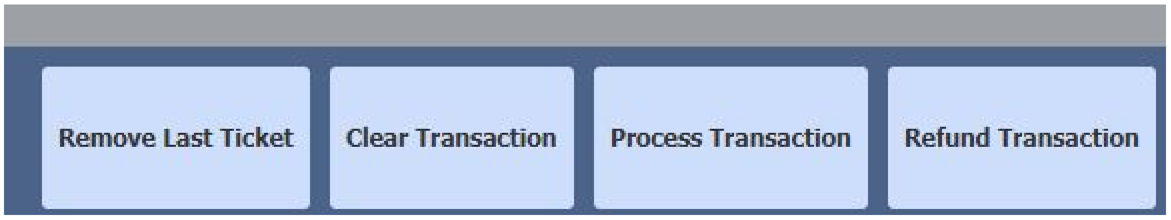
6. Process Transaction



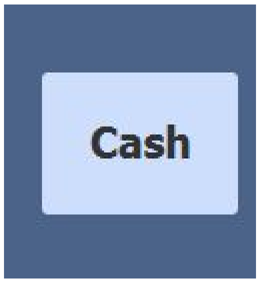
Bad Santa 2 - 11:00 AM ADULT - $8.50
Bad Santa 2 - 11:00 AM ADULT - $8.50
Bad Santa 2 - 11:00 AM SENIOR - $6.25
Bad Santa 2 - 11:00 AM Student - $6.25

TOTAL: $ 29.5

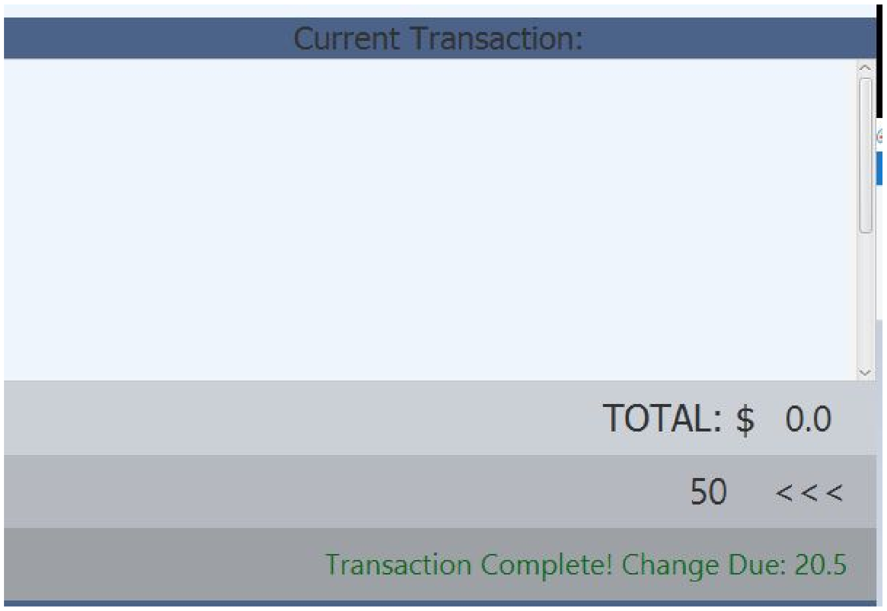
When a Transaction is paid with exact change user presses Process Transaction.



If paid with cash, user can enter the cash given in the Right Panel and press Cash.



This will process the transaction and give the user the Change Due.



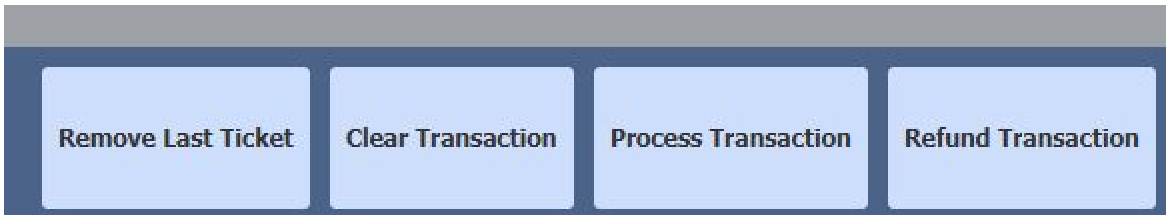
TOTAL: $ 0.0

50 <<<

Transaction Complete! Change Due: 20.5

7. Cancel Transaction

To cancel a currently pending transaction, press Clear Transaction in the Register Control Panel.



8. Refund Transaction

To refund a transaction, enter the tickets purchased as if you are going to process a transaction, but instead press Refund Transaction. The user will be prompted with the change due to the customer.

9. Logging Out

To logout of the Main Vendor screen, press the Log Out button in the Register Control Panel. This will return the user to the Employee Login Screen, remaining logged into the Theater.

To logout of the Employee screen, press the Log Out button in the Employee Login control panel. The user will be returned to the theater login screen, disconnecting from theater the system was logged in to.