Eastern New Mexico University

Team 4

Michael Gonzales, Garry Callis, Jeru Sanders, Stephen Washington

Project: Stage 4

Final Report

May 11, 2022

Table of Contents

Table of Contents 2

Introduction 4

Requirements 5

Client Requirements 6

Functional Requirements 8

Initial Effort Estimation 11

Domain Analysis 14

Concept Definitions: 14

Association Definitions: 15

Attribute Definitions: 16

Traceability Matrix 17

Interaction Diagrams 18

Module Diagrams and Specification 20

Class Diagram 20

Diagram 1: 20

Diagram 2: 21

Diagram 3: 22

Use Case Diagram 23

Sequence Diagram 25

System Architecture and System Design 26

Components Description 29

Design Rationale 30

Database 31

Algorithms and Data Structures 33

User Interface Design and Implementation 35

Design of Tests 46

Requirements/Type/Phase 46

Testing Schedule 49

Test Reports 50

History of Work, Current Status, and Future Work 58

History of Work 58
Gantt Diagram 59
Current Status 63
Future Work 64

Introduction

This is the final report for our software engineering project “Theatre Portales.” The goal of this report is to give a general overview of the entire project. This report will include specific details relating to requirements, design, tests, test results and future tasks for our system. This report will utilize and ingrate information from other documents such as the SDD document, Test Plan etc. Should the software needed to be maintained, this document plays a critical role, allowing the reader to understand the system. By the end of this report, you will have a thorough understanding of the system.

Requirements

When developing our system, it was crucial to meet specific requirements as stated in the instructions for the “Theatre Portales” project. This section will focus on the requirements we have been given by our client. In addition to the client requirements, we also added some basic functional requirements. These functional requirements will include system, admin, customer, and checkout requirements. Many client requirements are functional requirements. Due to this, most client requirements will also be included in the functional requirements section. As mentioned earlier in the introduction, this section will reiterate requirements previously included other technical documents relating to our system. For more information, please refer to these documents.

Client Requirements

These are essential requirements provided by the client for project "Theatre Portales." This list will not include additional functional requirements created by our team.

- Available seats are green
- Booked seats are in red
- Different plays are available during the year
- Admin
  - Must ask for username and password to access admin features
  - can add plays by date and time
  - Can generate a report (seats sold for certain plays and dates)
  - Adjust individual seat price
    - Can select one or multiple seats to accomplish this
- Registered customer
  - Can buy seats for any play
  - User information must include (name, age, address, telephone number, and email address)
  - Can buy seats for any future play

  - You can remove selected seat by clicking again
  - Can add seats to a shopping cart
  - Can make changes to cart (add/remove seats)
  - Can move from play to play to purchase seats
- Checkout
  - Must present total and report of transaction
  - Report contains a consecutive number for each transaction (transaction ID), play name, date/time, and seats purchased
- System
  - Must ask for payment information (Card number)
  - Confirm and inform customer the sale was correct
  - (Optional) email receipt
  - Must be online
  - (Optional) optimized for mobile devices

Functional Requirements

Listed below are the requirements that are essential to creating a fully functioning and effective system. This includes requirements designed to improve aesthetics, ease of use and improve customer/admin/client satisfaction. This list was created by our team.

- System:
    - 8 rows and 12 column seats
    - Seats must all fit on the screen nicely
    - Cannot be too small to touch
    - Leave some border on the end
    - All buttons must be visible
    - Cannot cover buy button at the bottom
    - Available seats are green
    - Unavailable seats are red
    - Seating for future plays
    - Online and accessible with mobile devices
        - Screen rotation
        - Touch buttons
        - Virtual keyboard

        - Screen size

- Admin:
    - Add new plays (time and date)
    - They can also edit and delete plays
    - Change individual seat prices
    - By selecting one or more seats and changing them
    - Convenient to change all the seats at once to the same price
    - Can generate a report with
        - How many seats were sold for a specific play and date
        - Requires username and password to access the management area.
- Customers:
    - Be able to create an account with an email address and password.
    - Be able to edit their information (ex: name, age, address, telephone, and email)
    - Registered customers can buy available seats for any play
    - Can select and deselect multiple seats
    - Can add a list of selected seats to a shopping cart
    - Can move from play to play to buy more seats

- Checkout:
  - Total due
  - Report on transactions
  - Transaction ID (consecutive number)
  - Name of play or plays
  - Dates/Time
  - Seat numbers
  - Customer needs to make changes to shopping cart
  - The system must ask the customer for a Creditcard number and simulates the sale.
  - System must inform user of the status of the sale
  - An option to the customer, the system must send an email with purchase details (Optional)

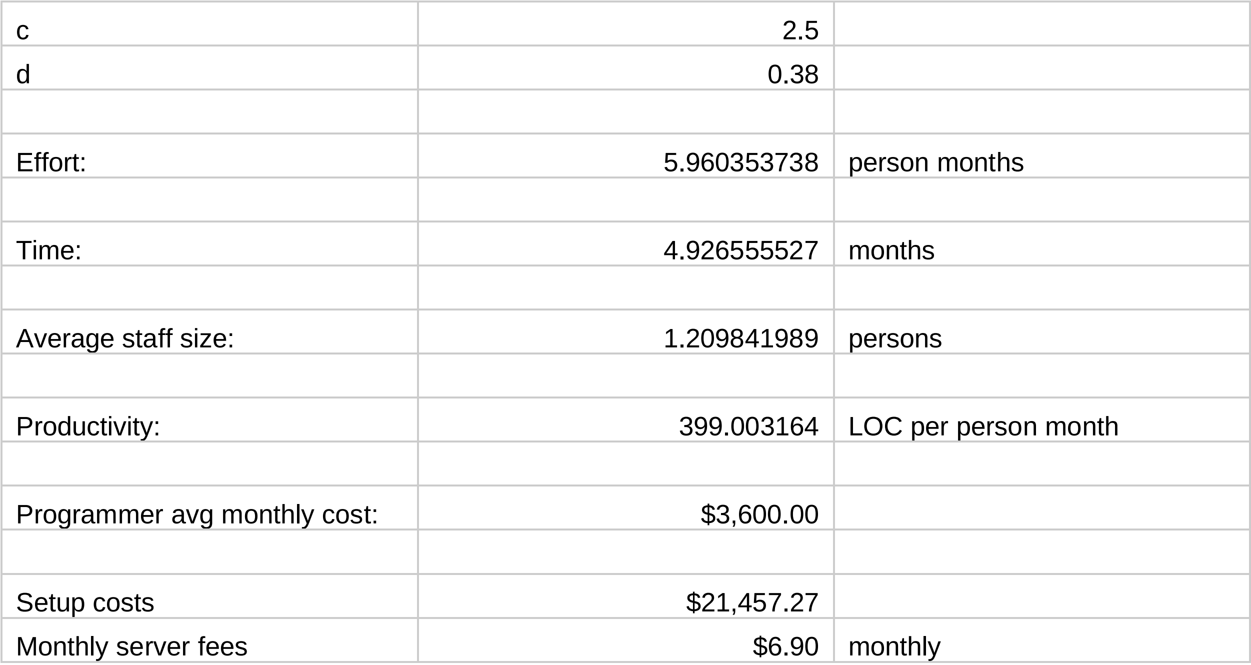
As a reminder, the client requirements provided above are essential. Functional requirements, not listed in client requirements, would be beneficial, enhancing the overall functionality of the system. Further on in this document, we will discuss software testing. This will reiterate important requirements stated in this document. In addition, we will discuss the tests we designed, to make sure the requirements listed in this section are met.

Initial Effort Estimation

Finances play a crucial role when designing a system. In this section, we have provided a graph used to estimate the amount of effort needed to design our system. This was an older estimation generated early in the semester. This graph provided the client with a general idea of what the cost might be to build the system. However, it is important to remember that this graph is an estimate. Depending on the number of variables, this could easily be more or less expensive.

Function types	Costs	
Internal Logical File	10	
External Interface File	7	
External Input	4	
External Output	5	
External Inquiry	4	
The system will keep a list of usernames and passwords required to enter the management area.	Internal Logical File	10
The system will store all showing and seat information to prevent double buying tickets.	Internal Logical File	10
The operator will provide a username and password to enter the management area.	External Inquiry	4
The operator will be able to add and remove shows.	External Input	4
The operator will be able to customize the dates, number of seats, and price for each seat per show.	External Inquiry	4
The operator will be able to view a report showing all sales info.	External Output	5

The customer will be able to view the upcoming shows and available seating.	External Output	5
The customer can select multiple seats and add them to their cart.	External Inquiry	4
The customer can enter their name, age, address, phone, email, and card info to purchase the seats at checkout.	External Input	4
The system will reserve the seat and confirm the sale to the user on screen.	External Output	5
UFP:		55
Value adjustment factors:		
Data communications	3	
Distributed data processing	3	
Performance	2	
Hevaily used configuration	0	
Transaction rate	1	
Online data entry	4	
End-user efficiency	4	
Online update	3	
Complex processing	0	
Reusability	0	
Installation ease	3	
Operational ease	3	
Multiple sites	0	
Facilitate changes	1	
VAF:	0.92	
AFP:	50.6	
5th gen language avg FP hours:	8	
Javascript avg code lines for fp:	47	
Estimated lines of code:	2378.2	
a	2.4	
b	1.05	



c	2.5	
d	0.38	
Effort:	5.960353738	person months
Time:	4.926555527	months
Average staff size:	1.209841989	persons
Productivity:	399.003164	LOC per person month
Programmer avg monthly cost:	$3,600.00	
Setup costs	$21,457.27	
Monthly server fees	$6.90	monthly

As you can see from the graph, we estimated that the system would take around 2571 lines of code. However, nearing completion of the project we surpassed this number in front-end development alone. Since we created more lines of code, our initial effort estimate was inaccurate. The expense to create the system would be much higher, requiring more time to develop the system.

Domain Analysis

Concept Definitions:

Responsibility	Concept
R1: Theatre Application	APP
R2: User Interface for application	UI
R3: Retrieve list from Shows	Show Manager
R4: Accesses stored shows	Show
R5: Access stored Seats	Seat
R6: Checks if user is an admin	ServerInterOp
R7: Adds new shows	ServerInterOp
R8: Attempt Login	ServerInterOp
R9: Reserves Seat	ServerInterOp
R10: Payment Transactions	ServerInterOp
R11: Retrieves list of seats for each show	Show Manager
R12: Selects seat to process for transaction	Show Manager
R13: Selects show to process for transactions	Show Manager
R14: Database	Database

Association Definitions:

Concept Pair	Association Description	Association Name
APP ⇔ UI	Provides user interface with the app.	AppUserInterface
APP ⇔ Show Manager	Show manager processes seat/show information for the App.	AppShowManager
APP ⇔ ServerInterOp	Query Database for login information, add/edit show/seat data. Deals with payment transactions	AppServerInterOp
Seat⇔ Show	Manages seats for each show	SeatShow
Show⇔ Show Manager	Information is consolidated for App use.	ShowManager
ServerInterOp⇔ Database	Query/Modify database	QMDatabase

Attribute Definitions:

Concept	Attribute	Description
App	UI	User Interface
	Show Manager	Show Manager
	Server	ServerInterOps
UI	User Interface	Provides a way for the user to interface with the app.
UI	ChangeScreen()	Changes screen
UI	UpdateScreen()	Updates screen
Show Manager	Shows	Provides data to the app.
Show Manager	currentShowIndex	Index of selected show
Show Manager	currentSeatIndices	Indices of selected seats.
Show Manager	selectShow	Show selected data.
Show Manager	selectSeat	Seat selected data.
Show	seats	Provide seat data for selected show.
Show	id	Show Id from database
Show	name	Name of the show
show	date	Date of show
Seat	price	Price for seat
Seat	ownerId	Owner ID set to the seat.
ServerInterOp	currentUserId	User ID
ServerInterOp	isAdmin	Admin verification
ServerInterOp	LoadShowData()	Queries database for shows
ServerInterOp	AttemptSignIn()	Queries database to verify user to login.
ServerInterOp	ReserveSeat()	Marks seat reserved in database
ServerInterOp	chargeCreditCard	Payment transactions
Database	Database	Database

Traceability Matrix

Use Case	Concept						
	App	UI	Show Manager	Show	Seat	ServerInterOp	Database
UC-1	X	X	X			X	
UC-2			X	X			
UC-3				X	X		
UC-4	X					X	X

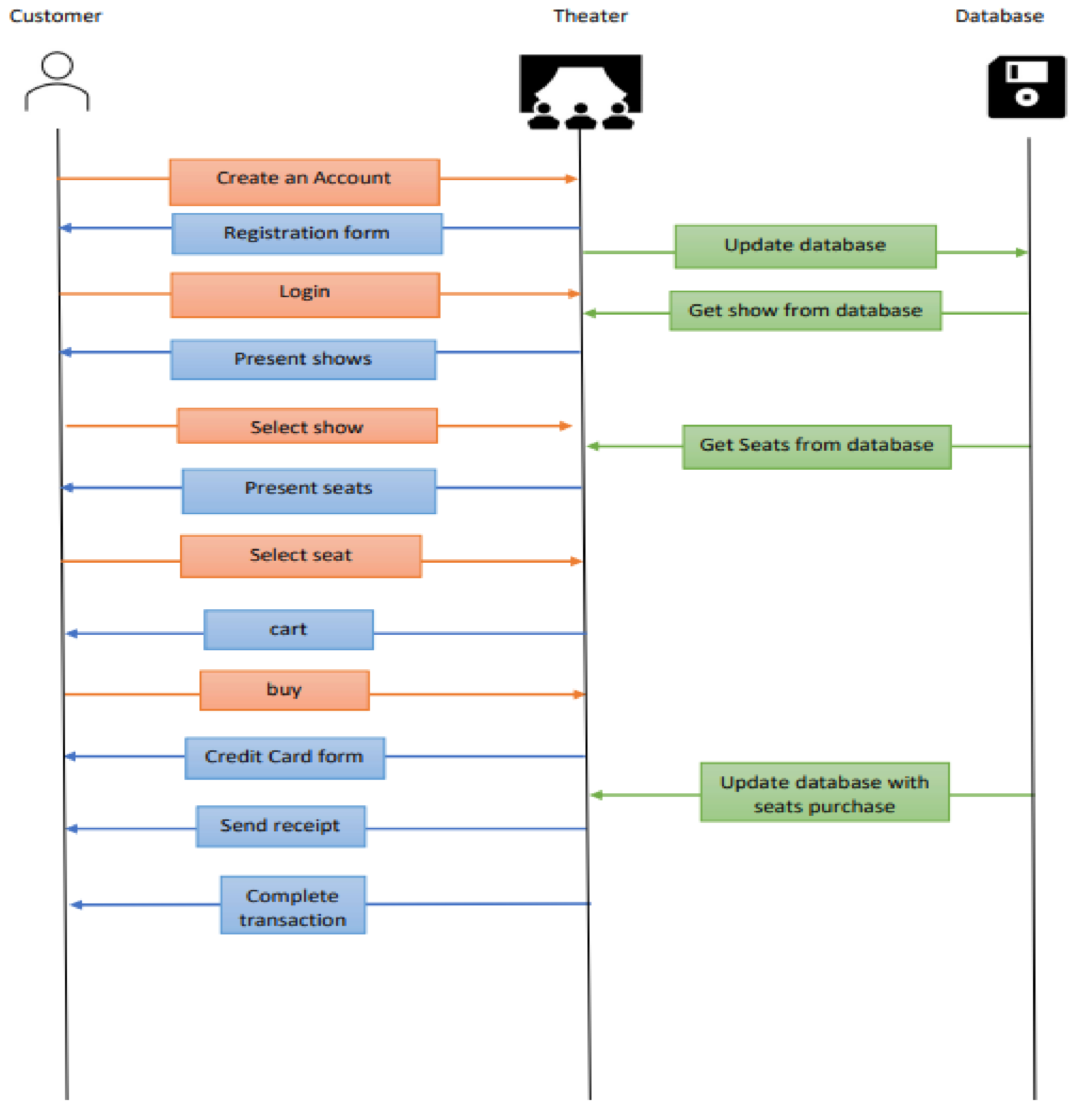
UC-1: The User Interface (UI) allows the user to interact with the app. The show manager processes show and seat data for the user to see and interact with on the app.

UC-2: The Show Manager uses show/seat data from Show to process for the app.

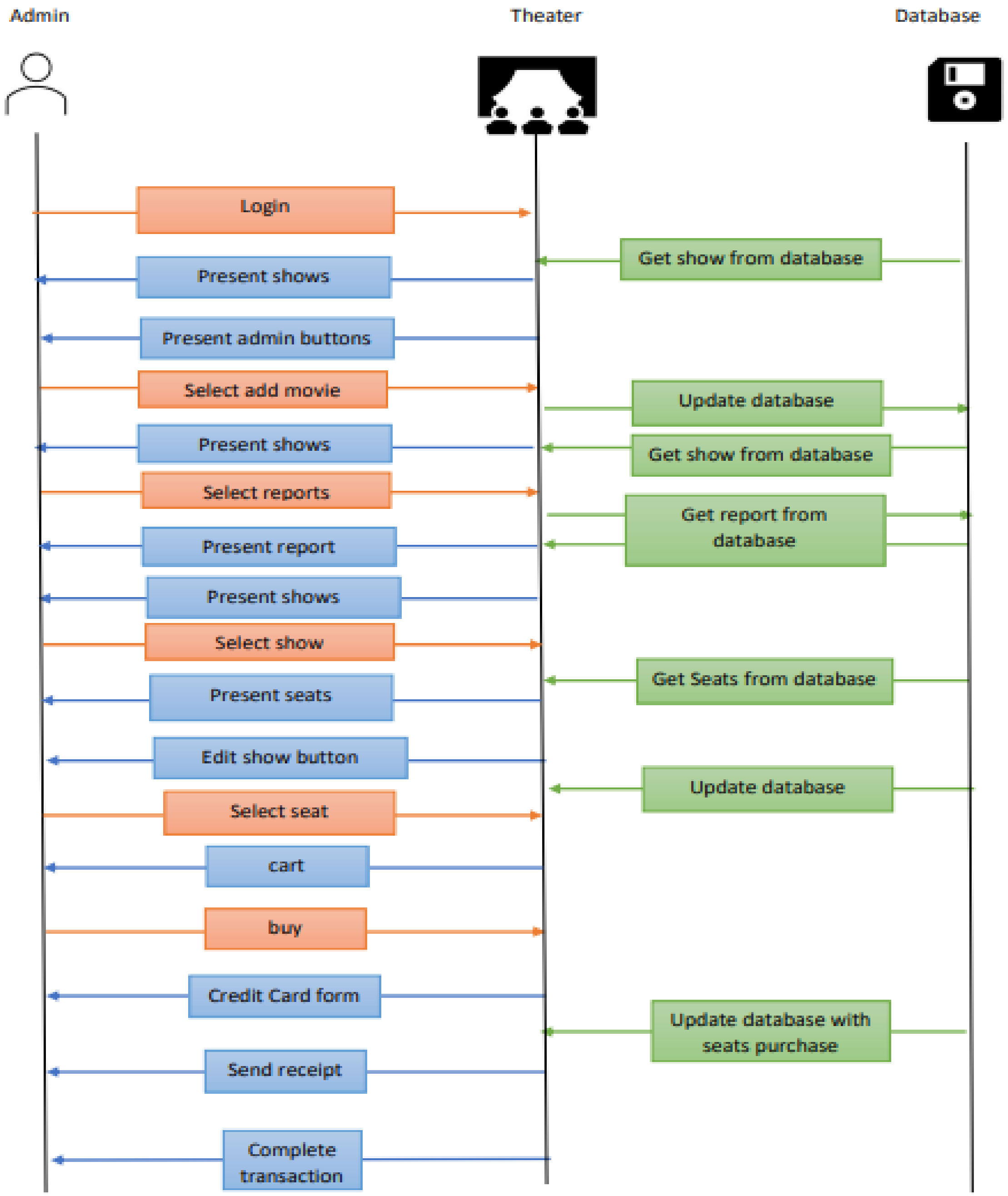
UC-3: Seat manages seat data for each show.

UC-4: The app uses ServerInterOP to gather user, admin, show, and seat information from the database. ServerInterOp uses information gathered from the database to login user, check if they are an administrator, mark/check seats reserved in the database, and process transactions.

Interaction Diagrams



The customer logs in to the theatre to present the available plays. The customer can select from the shows and will be presented with the seats. The customer will make their selection from the available seats and will be taken to the cart to review their selection. Once the customer is ready, they will complete the purchase process. Once the purchasing process is complete the database will update accordingly, and a receipt is given on screen.



The admin has the same abilities as a customer along with the ability to add, edit plays and to get the transaction history.

Module Diagrams and Specification

In this section we will provide diagrams of important modules of code within our system. We will provide diagrams to provide a visual representation of our software design. Then we will provide a brief explanation of each one. This will provide you with a better understanding of our software. Please refer to the "Domain Analysis" section for in-depth descriptions of each component mentioned.

Class Diagram

The title of this diagram is misleading. There are very few to no classes within our system. However, there are important modules of code within our system that are like classes, in that they complete important system functions. This diagram played an important role in the design of our system.

Diagram 1:

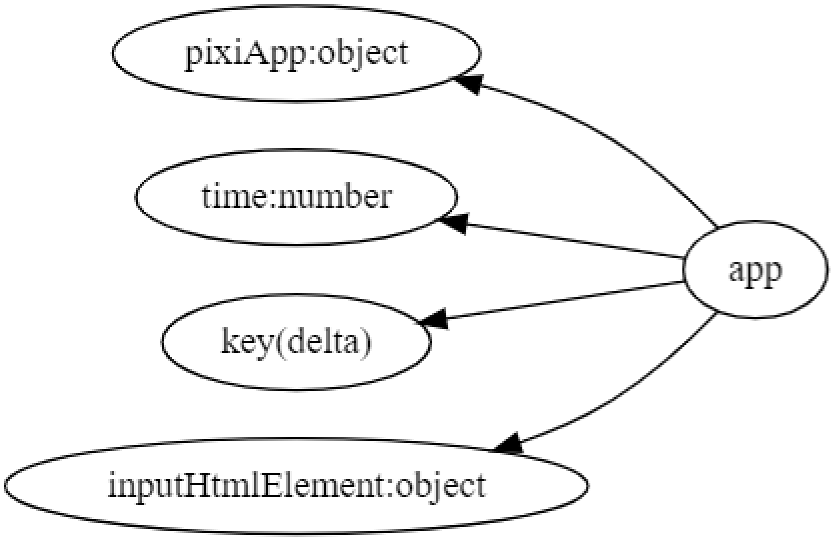


Diagram 2:

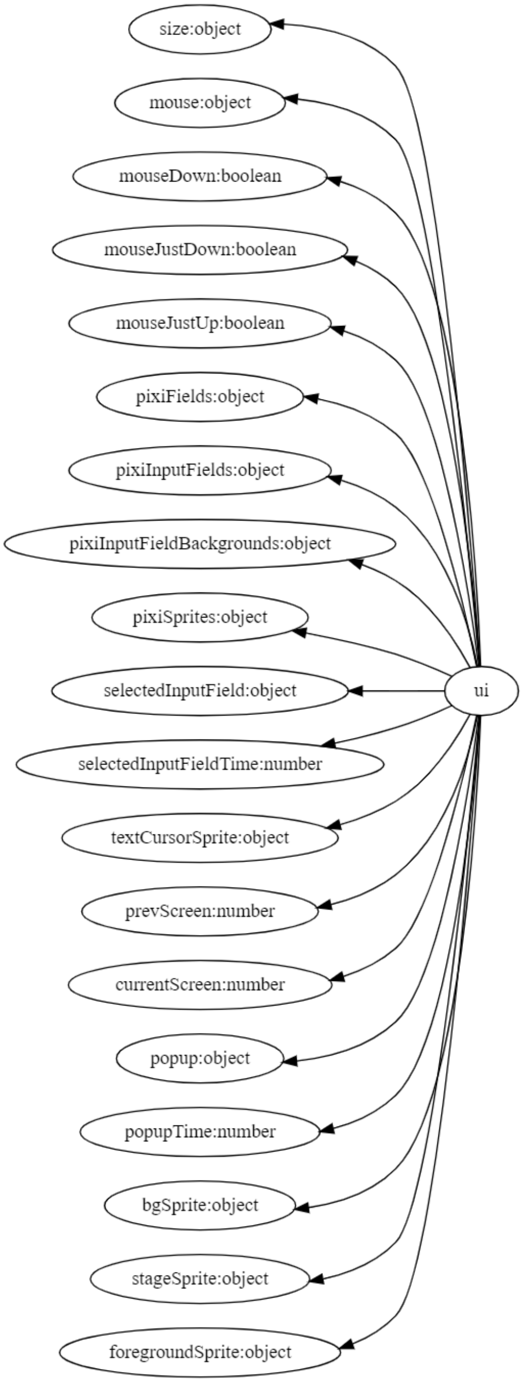
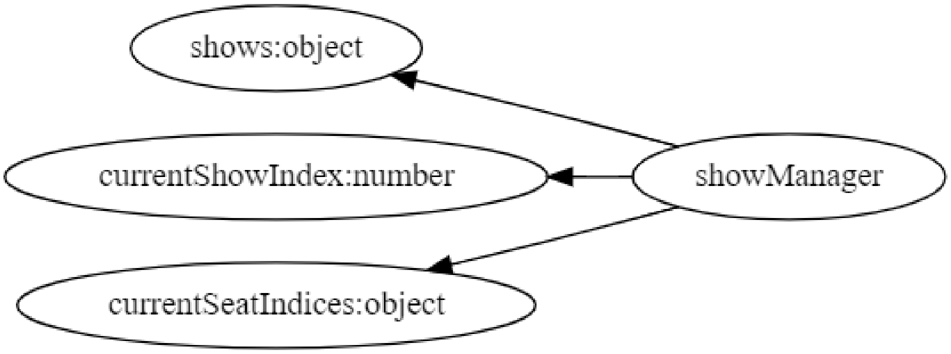
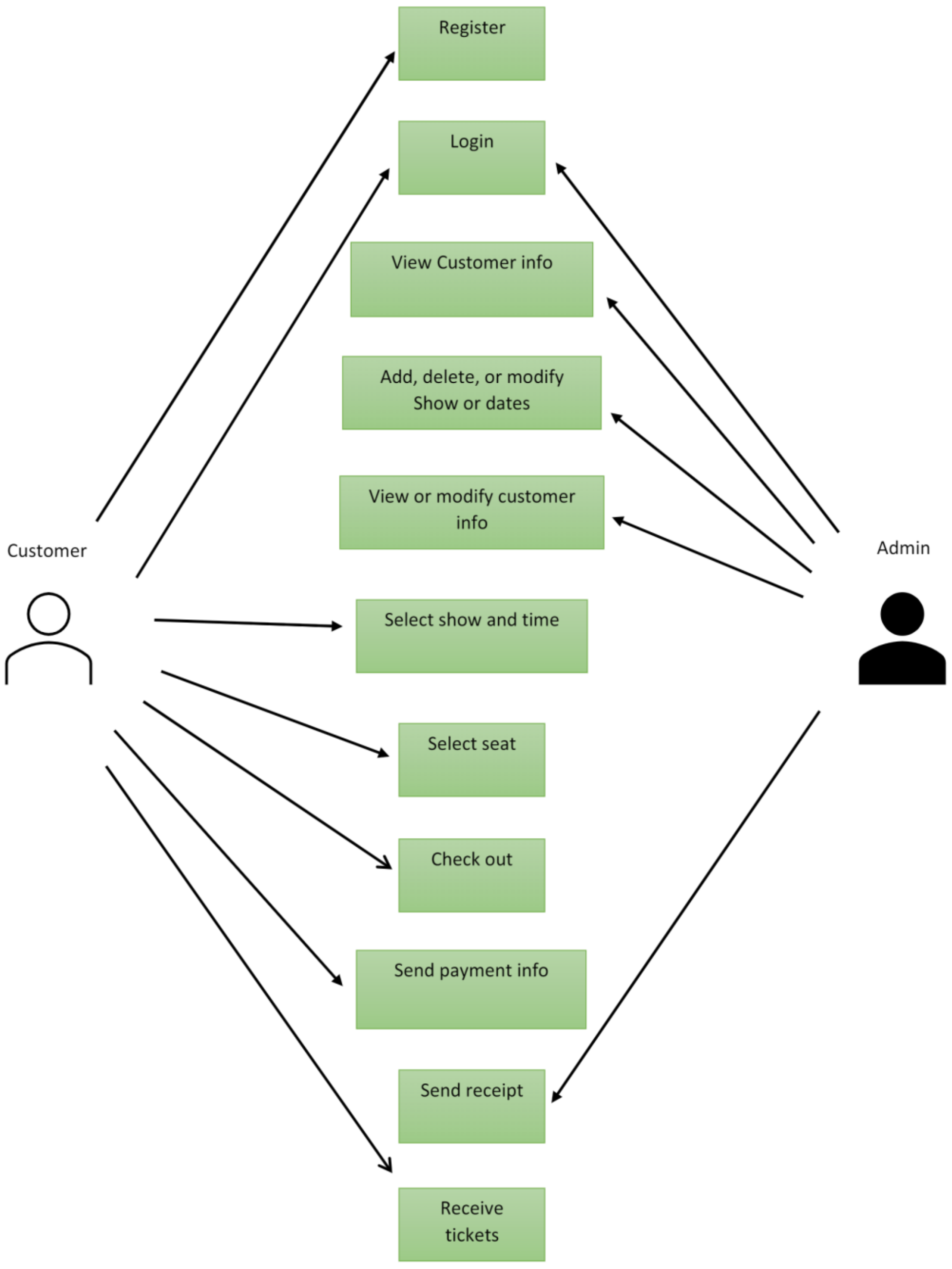

size:object
mouse:object
mouseDown:boolean
mouseJustDown:boolean
mouseJustUp:boolean
pixiFields:object
pixiInputFields:object
pixiInputFieldBackgrounds:object
pixiSprites:object
selectedInputField:object
ui
selectedInputFieldTime:number
textCursorSprite:object
prevScreen:number
currentScreen:number
popup:object
popupTime:number
bgSprite:object
stageSprite:object
foregroundSprite:object

Diagram 3:


shows:object
currentShowIndex:number
showManager
currentSeatIndices:object

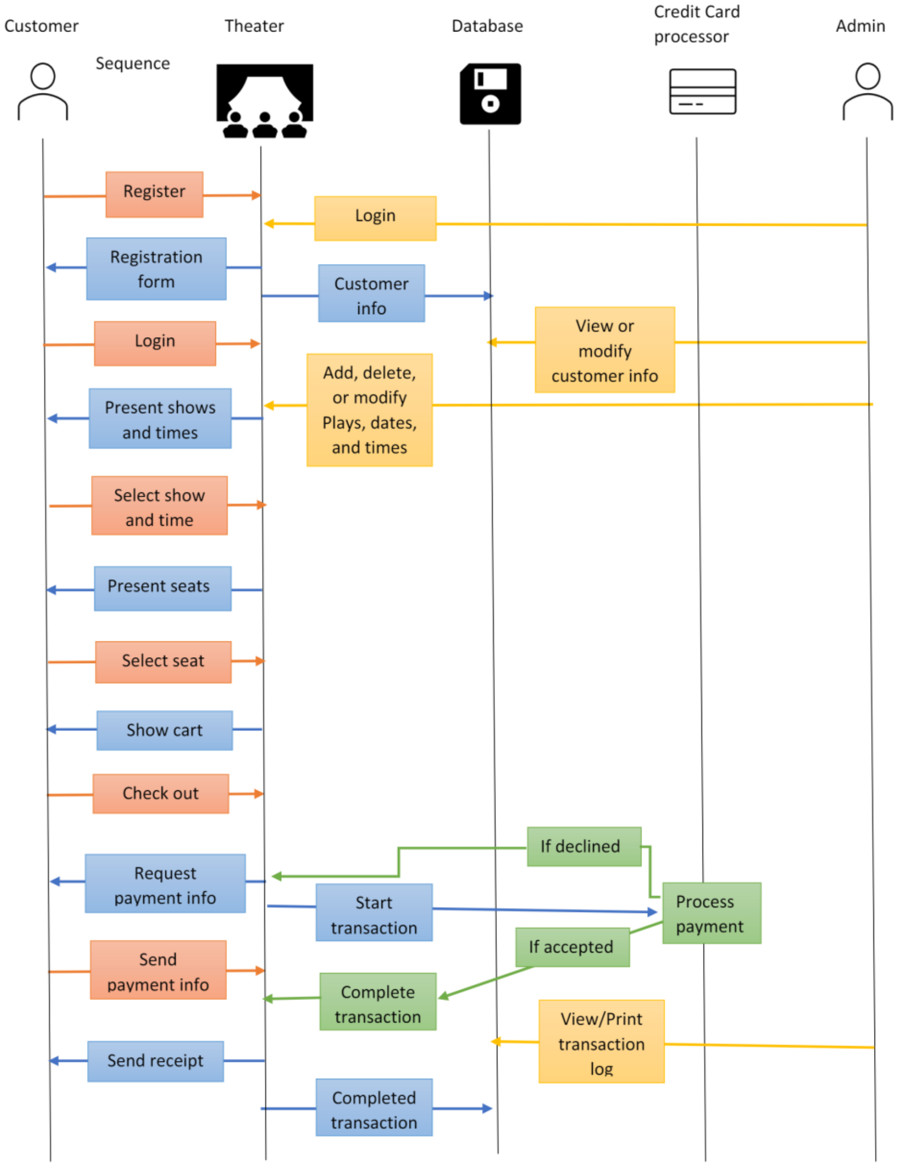
Use Case Diagram

There are two main users within our system. There is the “Admin”, who is a business owner and manager. The admin of the system will require certain functions in order to meet business needs. Then there is the “Customer”, who’s tasked with purchasing tickets for plays. The customer will require less system functionality. The Use Case Diagram provides an illustration of the functionality needed for both the client and the admin.

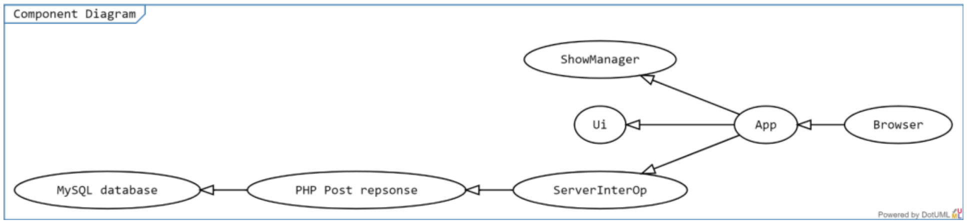

Register
Login
View Customer info
Add, delete, or modify Show or dates
View or modify customer info
Customer
Admin
Select show and time
Select seat
Check out
Send payment info
Send receipt
Receive tickets

Sequence Diagram

The Sequence Diagram provides an illustration between the users, the functions, and the database. Provided below is the sequence diagram.

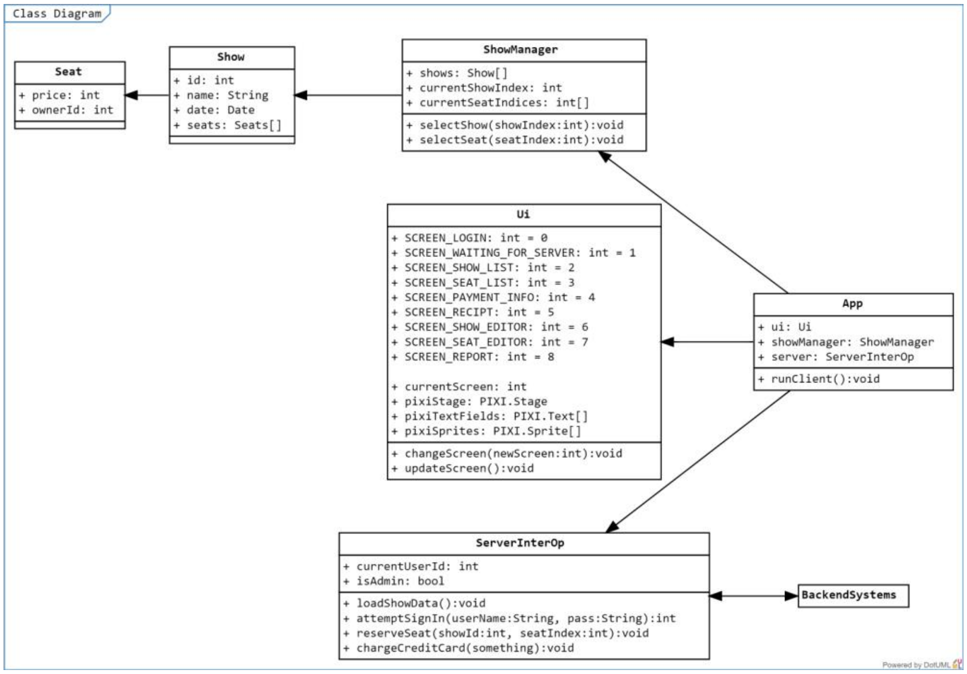


System Architecture and System Design



This is the client-side application for the Los Portales ticket sales platform. The client-side application is written in JavaScript and will run in the user's web browsers.

Both customers and admins are users. Customers will use this application to view current available shows and seats and purchase them. Admins are the owners are the venue and will log in to add and adjust shows. The client will be a single page application written in "vanilla" JS prominently using the library PixiJs.


Class Diagram
Seat
+ price: int
+ ownerId: int
Show
+ id: int
+ name: String
+ date: Date
+ seats: Seats[]
ShowManager
+ shows: Show[]
+ currentShowIndex: int
+ currentSeatIndices: int[]
+ selectShow(showIndex:int):void
+ selectSeat(seatIndex:int):void
Ui
+ SCREEN_LOGIN: int = 0
+ SCREEN_WAITING_FOR_SERVER: int = 1
+ SCREEN_SHOW_LIST: int = 2
+ SCREEN_SEAT_LIST: int = 3
+ SCREEN_PAYMENT_INFO: int = 4
+ SCREEN_RECIPT: int = 5
+ SCREEN_SHOW_EDITOR: int = 6
+ SCREEN_SEAT_EDITOR: int = 7
+ SCREEN_REPORT: int = 8
+ currentScreen: int
+ pixiStage: PIXI.Stage
+ pixiTextFields: PIXI.Text[]
+ pixiSprites: PIXI.Sprite[]
+ changeScreen(newScreen:int):void
+ updateScreen():void
App
+ ui: Ui
+ showManager: ShowManager
+ server: ServerInterOp
+ runClient():void
ServerInterOp
+ currentUserId: int
+ isAdmin: bool
+ loadShowData():void
+ attemptSignIn(userName:String, pass:String):int
+ reserveSeat(showId:int, seatIndex:int):void
+ chargeCreditCard(something):void
BackendSystems
Powered by DotUML

The following is a list of screens and their functions:

NONE	A sentinel screens
LOGIN	The screen where customers or admins log in
WAITING_FOR_SERVER	A screen for the client to wait for a server response on
SHOW_LIST	Shows all the available shows in a list (icon|title|date)
SEAT_LIST	Shows a grid of seats where the user can select multiple seats by touching/clicking them, then purchase them by clicking a buy button. Available seats are in white, unavailable ones are grey, selected are yellow
PAYMENT_INFO	Shows a screen where the user inputs their credit card, CSV, and exp date, then clicks buy. There should be visual icons of each of the major card

	providers, with all but MasterCard and VISA crossed out. (TODO: Figure how know what provider a card came from) (TODO: Figure out how to avoid two people paying at the same time for the same seat) Optionally: User can save or pick a saved card Optionally: Cash payment
RECIPT	Just show the seat and show your pay with a green check mark or something. Has a button to go back to the show list.
SHOW_EDITOR	A screen that admins get to from the seat list, that allows them to modify the show name, date, icon.
SEAT_EDITOR	A screen that admins get to from the seat list, that allows them to modify the selected seats price.
REPORT	A screen that dumps a huge filterable log of all the events

Once the user logs on or chooses to browse as a guest, the client will need to download the latest show and seat availability data. This will happen by calling *ServerInterOp.loadShowData()*, this function will make an XML HTTP Request to a page server.php?action=load, this page will reply with all show and seat data, the layout of which has not yet been determined (CSV strings).

Components Description

The app will consist of 4 global singletons.

- *Ui*: The module in charge of drawing the screen and processing user input into actionable commands. Solely in charge of Pixi.js.
- *ShowManager*: The module in holding the downloaded list of show and seats, all client-side show/seat data is held here. (This will probably be renamed to “Cart” later when that functionality is documents)
- *ServerInterOp*: The module in charge of interacting with the server, downloads the show/seat data and stores it in the *ShowManager*, sends update and verification commands to the server.
- App: Just a structure that holds all 3 of the other modules.

Each of these singletons will be initialized during an *onload()* callback within the main page's header. The relevant code for the singletons will be loaded from client.js and *runClient()* will be called.

- The *runClient()* function will simply start the update loop, further initialization will happen upon the first iteration of the update loop.

The update loop contains one giant "switch/case-like" if-statement for each entry in the SCREENS enum, the ui module contains the *currentScreen(ui.currentScreen)* that dictates which branch is taken. Each branch is responsible for the display and input processing for its screen. Just before the branch is taken a boolean *firstFrameOfScreen* is declared and set to true if *ui.screenTime==0*, *ui.screenTime* is incremented at the end of each update loop.

Design Rationale

The intention is that each screen uses *firstFrameOfScreen* being true as a signal for it to call ui creation functions like *createText(),* and *createButton()*. Then, monitor the state of the created elements for user interaction.

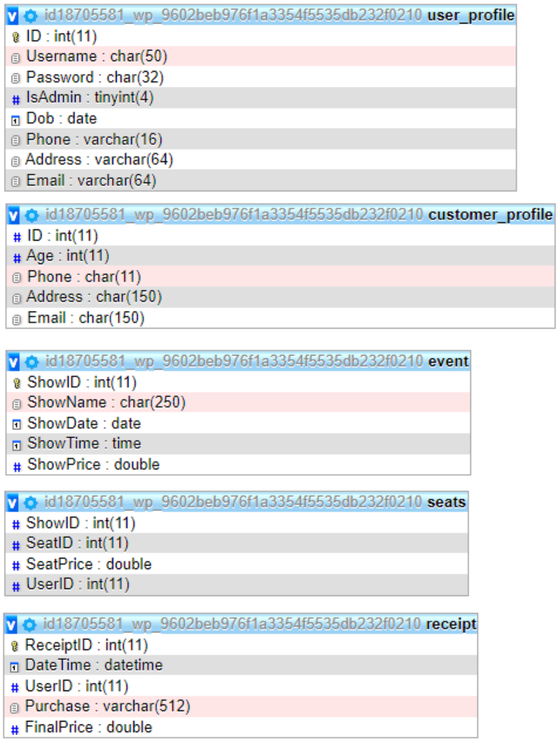
When a user action occurs, or some external event is received, *changeScreen()* can be used to change to a new screen. *Ui* elements do not need to be cleaned up, *Ui* elements will be automatically removed at the top of the update loop.

Database

A simple MariaDB database has been set up and it consists of 5 tables, as follows: *user_profile*, *customer_profile*, seats, event, and receipts. The database is set up to use IDs in different tables to connect all the data. PHP files are used to integrate the database with the application. The number of characters allowed for user inputs is set to the minimum number of characters needed for the database to function properly. This will help to prevent SQL injections into the database.

For reference, listed below are pictures of the actual database used in our system.

Actual Database:



id18705581_wp_9602beb976f1a3354f5535db232f0210 user_profile
ID : int(11)
Username : char(50)
Password : char(32)
IsAdmin : tinyint(4)
Dob : date
Phone : varchar(16)
Address : varchar(64)
Email : varchar(64)

id18705581_wp_9602beb976f1a3354f5535db232f0210 customer_profile
ID : int(11)
Age : int(11)
Phone : char(11)
Address : char(150)
Email : char(150)

id18705581_wp_9602beb976f1a3354f5535db232f0210 event
ShowID : int(11)
ShowName : char(250)
ShowDate : date
ShowTime : time
ShowPrice : double

id18705581_wp_9602beb976f1a3354f5535db232f0210 seats
ShowID : int(11)
SeatID : int(11)
SeatPrice : double
UserID : int(11)

id18705581_wp_9602beb976f1a3354f5535db232f0210 receipt
ReceiptID : int(11)
DateTime : datetime
UserID : int(11)
Purchase : varchar(512)
FinalPrice : double

ıs are made to add prepared statements to
latabase. Doing this will provide further
ﾌL injections. Relationships and keys will
next version, to increase the efficiency of
the time of some the function takes to
. This will allow the development of future
ime by making database commands more
specific, and reduce workload on system resources by organizing the database in

a way that it will easily be able to pull precise data to send to the frontend systems in the exact desired format. This will reduce the need to use some resources and reduce having to parse information from the database on the front-end systems.

Algorithms and Data Structures

Data structures: The application uses arrays and the dynamic Object type in JavaScript. More complicated structures are built ad-hoc, there are no classes. At the global scope there are 3 structs; app, *showManage*r, and ui.

- **app:** A small structure containing login id, current time, and other general app specific information.
- **ui:** Contains all the gui data in a big struct, this also is the only Object in which members are dynamically added. Update loops store temporary references to ui elements:
    - Ex) ui.addButton = createTextButtonSprite("Add");
        - In this way, “ui” functions like a hash map.
- **showManager:** Contains an array of Show's and an array of Seat's for the currently selected show.
- There is also a global array **cart** that contains CartEntry's.

Although there are no classes, there are 3 structs that are used in arrays and have a consistent layout.

- **Show:** Contains a show id, name, and date for a show.

- **Seat:** Contains the *userId* of the current owner (0 if none) and the price
- **CartEntry:** Contains the *showId* and *seatIndex* that the user wishes to purchase

Algorithms: In the client we use a linear iteration algorithm to get a show given a *showId* and to get the index of a show given a reference. For animation, we use a function called *clampMap*, that is a combination of normalization, interpolation, clamp, and ease. For easing we use Robert Penner's easing equations from his 2001 publication.

This algorithm is commonly used:

```
let col = index % COLS;

let row = Math.floor(index / COLS);
```

It derives a 2-dimensional coordinate from a 1-dimensional array.

User Interface Design and Implementation

Objective

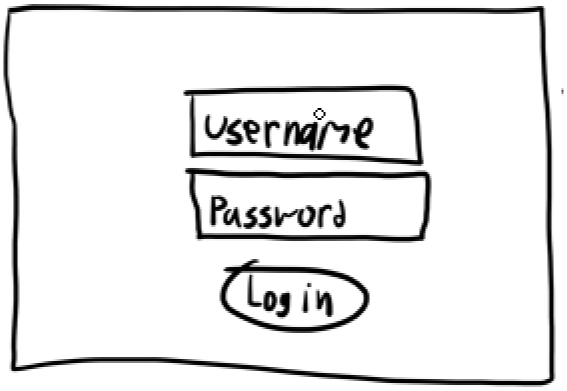
When designing the user interface, we wanted to focus on meeting all requirements previously stated by our client. In addition to our functional goals, we focused on simplicity and ease of use for our users. This aids with the overall satisfaction of the users and client. In this section we will discuss the design and layout of the user interface. While designing the software for "Theatre Portales," we spent time ensuring that the software was easy to use for the user. This was done by making a simple yet easy to understand layout. To give you a general idea of our software design, we will discuss our original design plans (included in the SDD document) and our final design plans.

Original Design

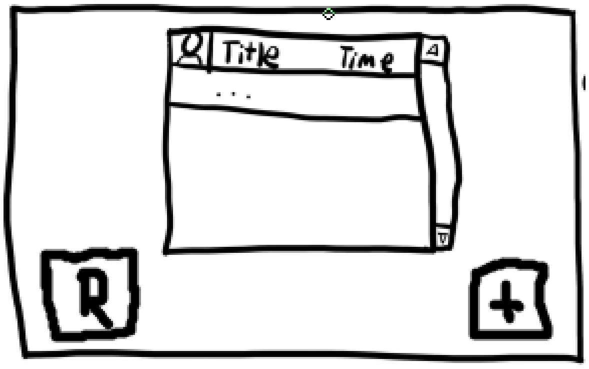
There were two main perspectives to consider when constructing the software: the admin perspective and the customers perspective. Listed below were a few essential features for each perspective needed for our software. For more original design information please refer to the SDD document.

Customer's perspective:

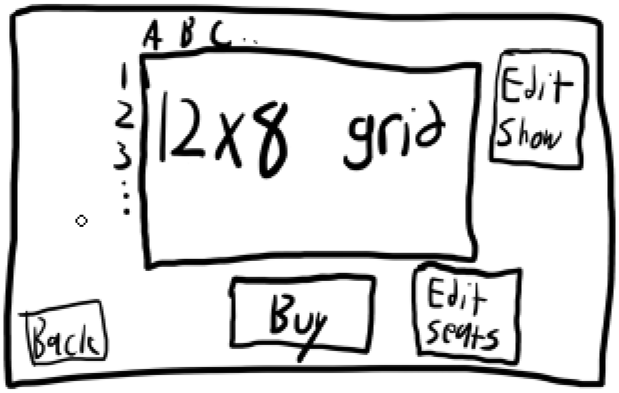
- Upon opening the app, the customer is prompted to log in to their account, they can either log in, or press a "browse as guest" button to proceed.



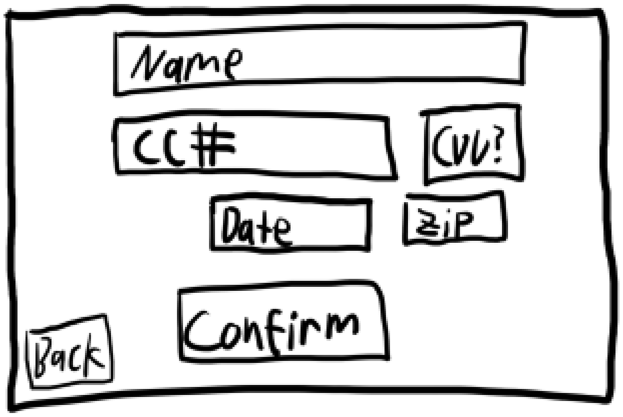
- After that the customer sees a list of all the shows that are scheduled for the future. The customer can scroll up and down this list and touch an entry to view the seat availability.



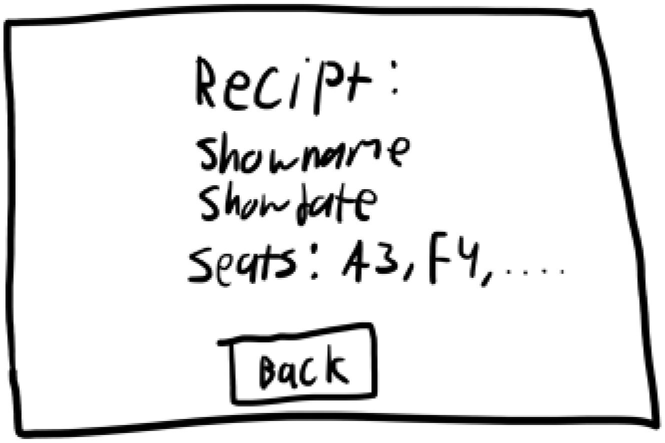
- On the seat listing page, the customer sees a grid of seats, color coded to their availability. Green=available, Gray=taken. The customer can touch a seat to select or deselect it, selected seats glow yellow, taken seats cannot be selected. Once the customer has selected 1 or more seats, a purchase button appears at the bottom of the screen.



- When the customer clicks this button, they are taken to the payment screen where they are shown their selected seat numbers, the price, and prompted to provide their credit card information. As an extra feature, we might add a "save card" feature.



- Once the customer confirms their CC information, a request is sent to the server to process the transaction and mark the seat as taken. The server relies on the client's application, with whether the transaction was successful or not.

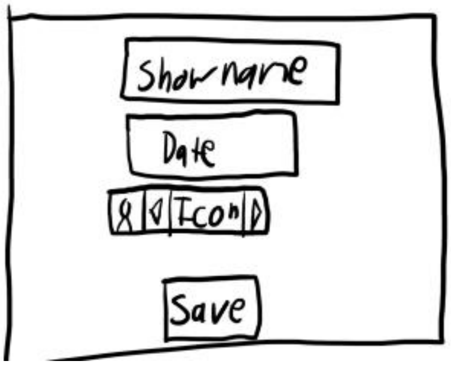


- The transaction may fail if the CC does not have the appropriate funds, or the seat was just taken.
- If the purchase was successful, the customer is taken to the receipt screen where they are shown information about their purchase.

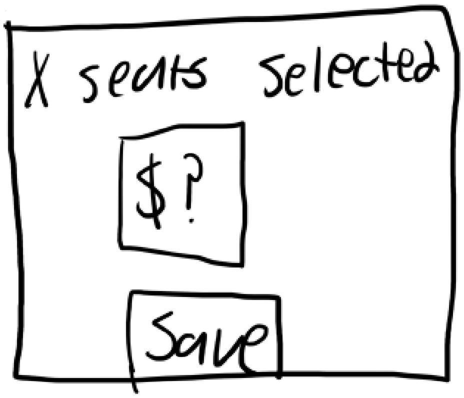
Admin perspective:

The admins perspective is mostly identical to the customers' perspective. The main difference is that on the seat grid screen, instead of a purchase button, admins see an "Edit seats" button, that takes them to a screen where they can set the price of the selected seats.

- They all see an "Edit show" button that allows them to edit the name and date of the currently selected show.



- On the show listing screen, they have an "Add show" button, that creates a new show, and moves them into the "Edit show" screen for that show.



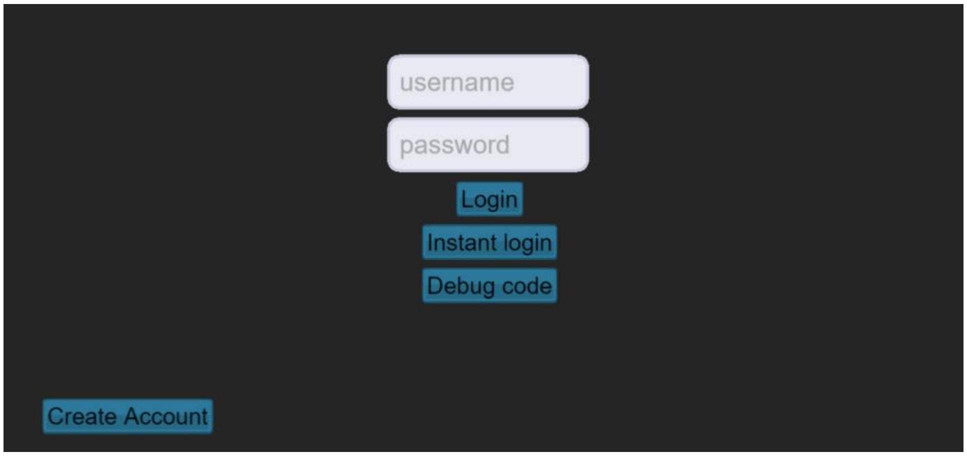
- Also on the show listing screen, they have a "Reports" button that takes them to a screen where they can see a filterable list of purchase history and information.

Report
Back

Current Design

Screen 1:

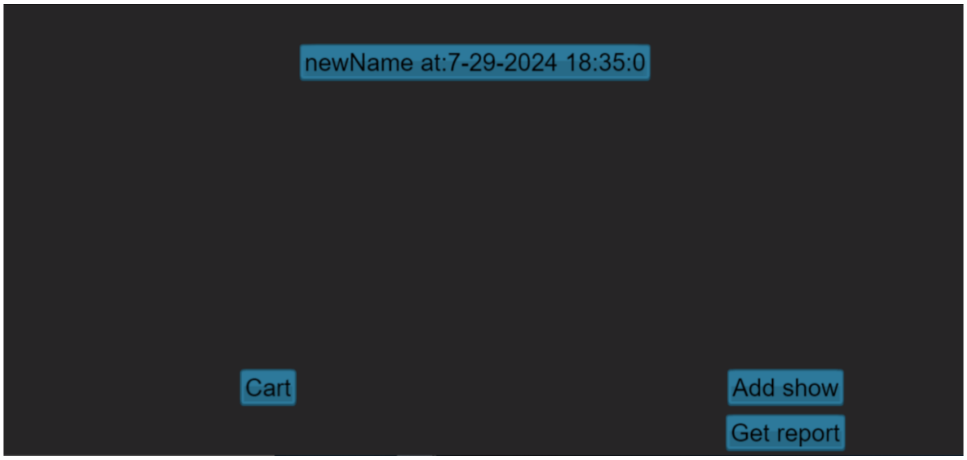
On the first screen, we have the login screen. This is where customers with an account can easily enter their username and password. Then they can simply hit the login button located right below where they entered their credentials. If they entered their credentials. If they enter incorrect information, a pop up will appear letting the user know that they entered incorrect information. If they are new customers, they can hit the create an account button. Then they will enter in the required information needed to create an account. Here is our login screen (software still in progress).



As you can see, the place to enter a password has a hint text, letting the user know what information is needed. In addition, the create button is extremely easy to locate. We placed it in the far-left corner of the screen. This enhances visibility and ease of use.

Screen 2:

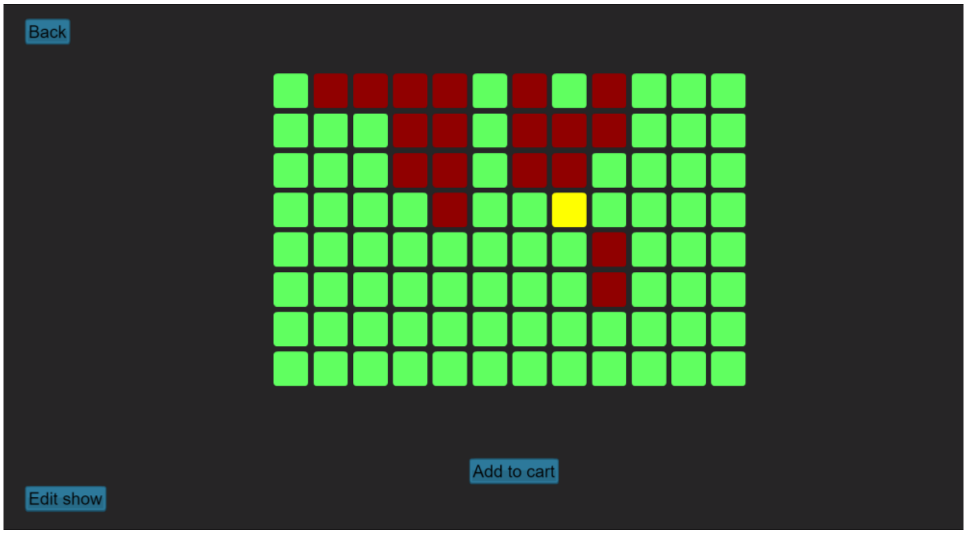
After credentials are entered, and the user proceeds to login, we come to a show screen. This screen will provide information on the shows provided by “Theatre Portales”. All plays are shown in the middle of the screen. This is to draw attention to the main purpose of the screen, which is to display the plays.



For the admin perspective, you will have access to the Add show and Get report buttons. It is also located in the button right section of the screen for maximum visibility. However, if you have entered non-admin credentials, you will not have access to these buttons. This feature aids the simplicity of the screen. You have two options as a customer, select seats in a certain show, or go to cart and proceed with checking out. It can get simpler than this.

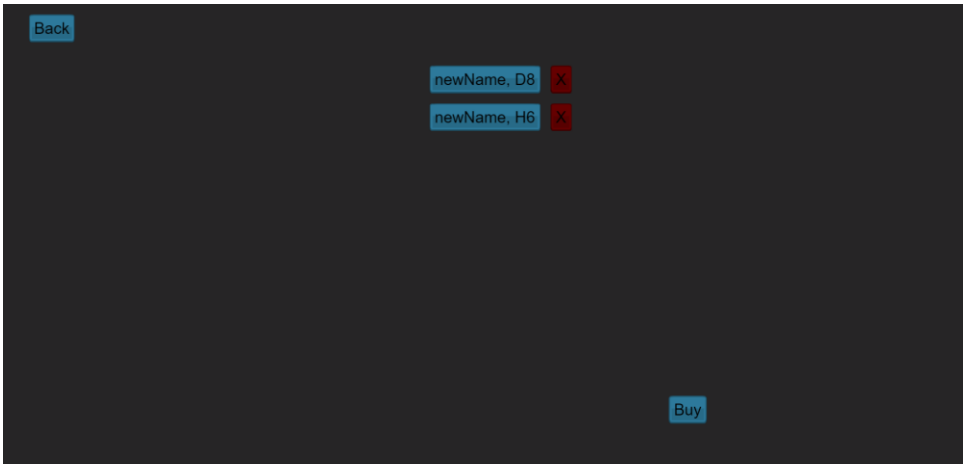
Screen 3:

The screen shown below is a basic seat selection screen. We are brought to this screen when we select the show. As you can see, available seats are marked green. This signals to the client that they are available. Red squares stand for seats that are taken. Yellow means seats have been selected. When you select a seat, you will see an “add to cart” button. This will add the selected seat (ticket) to the cart.

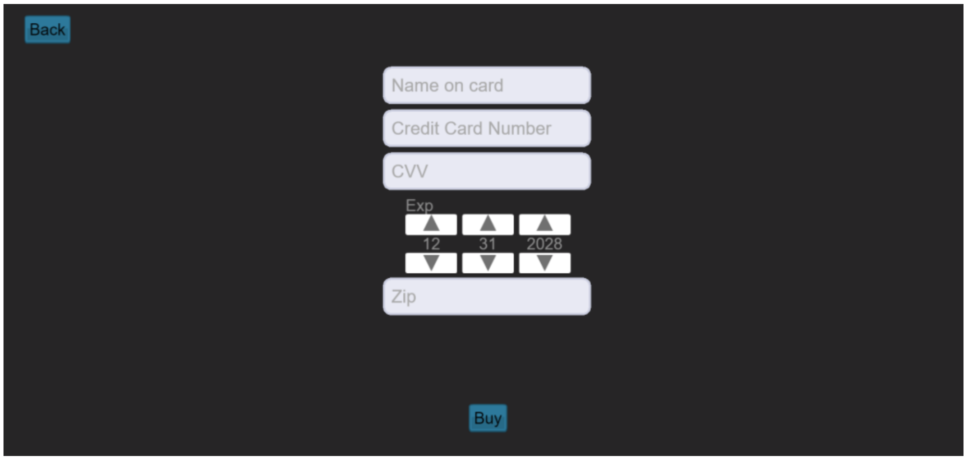


Screen 4:

This is the cart screen. As you can see, the play and seats are in the middle of the screen. Drawing the attention of the customers. On the right side of the tickets, you can easily see a red “X.” This will remove the selection from the cart. On a customer verifies that the cart is right they can proceed to “Buy.” This will proceed to the purchasing information screen.

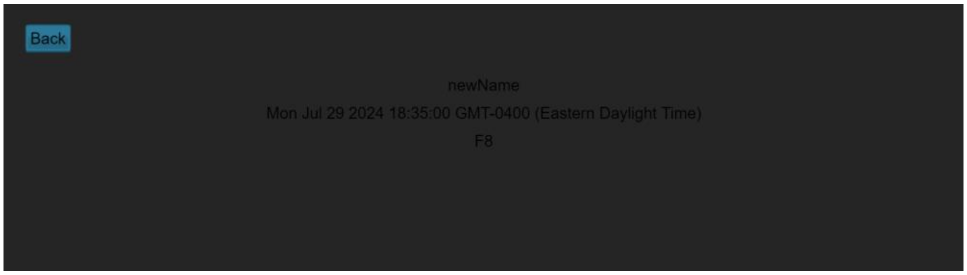


Screen 5:



On the purchasing screen, we have hinted texts. As previously mentioned earlier this provides the user with the information they need to enter in the correct information. For the card expiration we decided to create a button-based selection icon. This is useful because the client will not have to use a certain type of format (ex: 04-30-21 or 04/30/21). This feature creates a more user-friendly selection option.

Screen 6:



newName
Mon Jul 29 2024 18:35:00 GMT-0400 (Eastern Daylight Time)
F8

After you simply enter your payment information, a receipt will be generated. If you wish to purchase another ticket, you simply need to select the “back” button. The “back” button is highlighted in blue to make it easier to locate.

I hope this section has provided you with useful information about the user interface design. Both the admin perspective and customer perspective were considered. In our completed software, the admin will have access to what they need, and the customer will have access to what they need. Customers will be limited to purchasing, while admins get to make needed changes to update this system for optimal functionality. As you can see from the design description, we spent much of our time making sure that the user had a pleasant experience using our software. This was accomplished by focusing on ease of use and simplicity.

Design of Tests

Requirements/Type/Phase

During the project we had to create a Software test plan. This was a document that was used to address client requirements and functional requirements. We created a list of these requirements, that are displayed in the table below.

System Display Requirements		
Requirements	Type (due date)	Testing Phase
8 rows and 12 column seats	Visual Inspection	1
Buttons must all fit on all screens nicely with padding and not overlapping	Visual Inspection Layout Testing Mobile testing	1 2 3
Available, selected, and taken seats are color coded (green, yellow, red)	Visual Inspection	1-4
Mobile devices are supported	Visual Inspection Mobile testing	2 2
Touch screen works for buttons	Visual Inspection Mobile testing	2 2
Virtual keyboard and input text works	Visual Inspection Mobile testing	3 3
Screen rotation works	Visual Inspection Mobile testing	3 3
Shows seating for future plays	Visual Inspection	1
Customer Requirements		
Users can create accounts with an email address and password	Integration testing Unit testing	3-4 4
Users can log in to their accounts	Visual inspection Integration testing Unit testing	2 2 4

Customers can edit their profile details (name, age, address, tele, email)	Integration testing Unit testing	3-4 4
Customers can select and deselect seats. You can add a list of seats to the cart.	Visual inspection Integration testing	1 1
Customers can browse and add tickets to the cart for checkout	Visual inspection Integration testing Unit testing	1 3 4
Customers can add tickets to multiple plays to the cart and check them out in one transaction	Integration testing Unit testing	3 3-4
Users can edit tickets and view the current total in the cart	Visual inspection Integration testing	3-4 3-4
Users see the tickets and shows they purchase tickets for after checkout	Visual inspection Integration testing	2-4 4
Admin		
Can create, edit, and delete plays	Integration testing Unit testing	4 4
Can set the prices of seats individually and by selecting multiple seats	Integration testing Unit testing	4 4
Can generate a report of purchases	Integration testing	3-4
Checkout		
The system requires customers to put in credit card information and simulate a sale. Informing user of the status of the sale (ex) purchase is successful).	Visual inspection Integration testing Unit testing	1 2-3 4
Generate a receipt/report of purchase	Visual inspection Integration testing	1 2-4
Receipt/report displays correct information (transaction ID, Name of play, Date/Time, Seat numbers)	Visual inspection Integration testing	1 2-4

Optional email with receipt information (Ask professor for to clarify).	Integration testing Unit testing	4 4

Phase 1: Initial planning and requirement understanding phase

Phase 2: Design specification phase

Phase 3: Risk analysis and testing phase

Phase 4: Code and demo phase

Testing Schedule

As mentioned in the table above, our requirements were put into phases. These phases were time estimates on when the tests should be completed. Listed below is the graph from the software test plan, providing information about the testing dates.

Phase	Date to Start	Date to Complete
1	Jan 30th	Feb 27th
2	Feb 27th	March 27th
3	March 27th	April 17th
4	April 17th	May 11th

Test Reports

During the testing section our goal is to describe the tests that need to be performed, following the phases listed in the above graph. Below is a document that describes all requirements that will need to be tested. This includes display requirements, customer requirements, admin requirements, checkout requirements, and backend requirements. Unlike the other graphs listed in this section, this graph will provide more detail on pass/fail criteria, testing procedures, and what we expect the result to be. For more detailed information on requirements, please refer to the Software Test Plan document.

Display Requirements Testing		
Testing Requirement: 8 rows and 12 column seats Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Visual testing.	Pass: When selecting seats, each play has 8x12 seats.	Fail: A play or plays would not contain 8x12 seats.
Result	Pass	
Testing Requirement: Buttons must all fit on all screens nicely with padding and not overlapping Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Go to each screen with a variety of window sizes and observe the layout.	Pass: All buttons are spaced and sized in a convenient way for users.	Fail: Button overlaps, are placed partially or fully off screen, or are misaligned.
Result	Pass	

Testing Requirement: Available, selected, and taken seats are color coded (green, yellow, red) Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Log in, select some seats, observe the color change, purchase the seat, observe another color change.	Pass: The seat color correctly reflects its status.	Fail: The seat's color does not respond to its status.
Result	Pass	
Testing Requirement: Mobile devices are supported Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Test on Android and iOS.	Pass: The app should work and have a reasonable layout.	Fail: The app screen does not have a reasonable layout.
Result	Pass	
Testing Requirement: Touch screen works for buttons Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Touch buttons on the phone.	Pass: The software buttons work when used on phone.	Fail: buttons do not work when pressed on phone.
Result	Pass	
Testing Requirement: Virtual keyboard and input text works Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Touch a text field to see if the on-screen keyboard appears and is functional.	Pass: The keyboard is functional in all ways.	Fail: Keyboard does not appear, text is too small, cursor bar does not appear, or layout changes abruptly.
Result	Pass	
Testing Requirement: Screen rotation works Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	

Rotate the screen suddenly and continuously.	Pass: The screen rotates and the layout of the app changes to a convenient horizontal layout.	Fail: Any layout issue, or a non-response.
Result	Fail (landscape does not work on low res devices)	
Testing Requirement: Shows seating for future plays Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Log in as and admin, create a show, go to the show list, observe the new show on the list.	Pass: The show appears as expected.	Fail: The show does not appear ever, or only appear upon refresh.
Result	Pass	
Customer Requirements Testing		
Testing Requirement: Users can create accounts with an email address and password Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Start the app, create a user, then sign in with the user.	Pass: You can sign in.	Fail: You cannot create the user, or cannot log in, or are an admin.
Result	Pass	
Testing Requirement: Users can log in to their accounts Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Enter in password and username for a registered customer	Pass: After information is entered, the customer will proceed to select a play/seat.	Fail: After information is entered, the customer will not be able to proceed.
Result	Pass	
Testing Requirement: Customers can edit their profile details (name, age, address, tele, email) Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	

Log in, edit your details, refresh the app, log in again, then make sure the changes were made.	Pass: The details change.	Fail: Details do not change, or feature does not exist.
Result:	Fail (Profile editor not implemented)	
Testing Requirement: You can add a list of seats to the cart. Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Log in, select a show, select seats, add them to the cart, visit the cart.	Pass: The seats selected appear as cart items.	Fail: Items are missing or duplicated.
Result	Pass	
Testing Requirement: Customers can checkout items from the cart Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Add items to your cart, then checkout.	Pass: You can check out the items.	Fail: The items fail to be purchased or succeed and are not updated in the database.
Result	Pass	
Testing Requirement: Customers can add tickets to multiple plays to the cart and check them out in one transaction Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Add tickets from multiple plays to the cart and check them out.	Pass: Then items are checked out as expected.	Fail: Some or all items do not check out.
Result	Pass	
Testing Requirement: Users can edit tickets and view the current total in the cart Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Go to the cart and remove a ticket and watch the total.	Pass: Tickets are removable, and the total reflects the dollar amount correctly.	Fail: Tickets are not removable, or the wrong tickets are removed, or the total is inaccurate.

Result	Pass	
Testing Requirement: Users see the tickets and shows they purchase tickets for after checkout Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Purchase tickets and observe the receipt screen.	Pass: The tickets that were purchased are visible.	Fail: Some or all the tickets are missing.
Result	Pass	
Admin Requirements Testing		
Testing Requirement: Admins create, edit, and delete plays Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Sign in as an admin and attempt to create, edit, or delete a play.	Pass: Operations work as expected.	Fail: Operations fail, appear to fail, and succeed. Or a non-admin can perform operations.
Result	Pass	
Testing Requirement: Can set the prices of seats individually and by selecting multiple seats Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Sign in as an admin and attempt to change the price of some seats.	Pass: The prices are changed, and the screen reflects the correct new price.	Fail: The price does not change.
Result	Pass	
Testing Requirement: Can generate a report of purchases Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Sign in as an admin and click the get report button.	Pass: A nice looking report is generated and displayed on screen.	Fail: The report is wrong, or badly formatted.
Result	Pass	

Checkout Requirement Testing		
Testing Requirement: The system requires customers to put in credit card information and simulate a sale. Informing user of the status of the sale (ex) purchase is successful). Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Enter information on payment screen	Pass: If information is entered then the purchase is processed. Receipt is provided.	Fail: If information is missing, an error will occur.
Result	Pass	
Testing Requirement: Generate a receipt/report of purchase Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Admin will be able to generate a report.	Pass: Admin can generate a report.	Fail: Customer can generate a report, or no one can generate a report.
Result	Pass	
Testing Requirement: Receipt/report displays correct information (transaction ID, Name of play, Date/Time, Seat numbers) Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Generate a report by logging in as an admin.	Pass: Information listed above is inaccurate.	Fail: information listed above is accurate.
Result	Pass	
Testing Requirement: Optional email with receipt information (Ask professor for to clarify). Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
The receipt screen will present an option to email a receipt to the customer.	Pass: The button will send a copy of the receipt to the email provided.	Fail: The button will not send a copy of the receipt to the email provided.
Result	Fail	

Backend Requirements Testing		
Testing Requirement: Connection testing Pass/Fail Criteria: Able to connect to the web host server		
Testing Procedure:	Expected Result:	
Execute any PHP file.	Pass: $conn is initialized correctly and the request succeeds.	Fail: $conn is null and the request die()’s.
Result	Pass	
Testing Requirement: PHP files have results Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Test all functions of the system that use PHP files.	Pass: If no errors occur when running a system function.	Fail: If errors occur when running a system function.
Result	Pass	
Testing Requirement: PHP files have error messages Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Test errors and check the popup	Pass: The popups appear for each error and are displayed in a user-friendly way.	Fail: An error occurs and the app freezes or fails invisibly. Or error message contains user hostile information.
Result	Fail; Error message frequently contain sql_error().	
Testing Requirement: PHP files have success messages to verify execution. Pass/Fail Criteria:		
Testing Procedure:	Expected Result:	
Test PHP files separately and examine the response.	Pass: All PHP files respond in the affirmative upon success.	Fail: PHP files are silent or display errors on success.
Result	Fail; PHP files are often silent upon success; the client will crash otherwise.	
Testing Requirement: PHP files have results, error messages, and success messages verify execution. Pass/Fail Criteria:		

Testing Procedure:	Expected Result:	
Run system functions that utilize PHP files.	Pass: If an error message presents itself during fail. Or no errors occur within the system.	Fail: If no error message occurs, when there is a failure.
Result	Pass; no way to guarantee.	

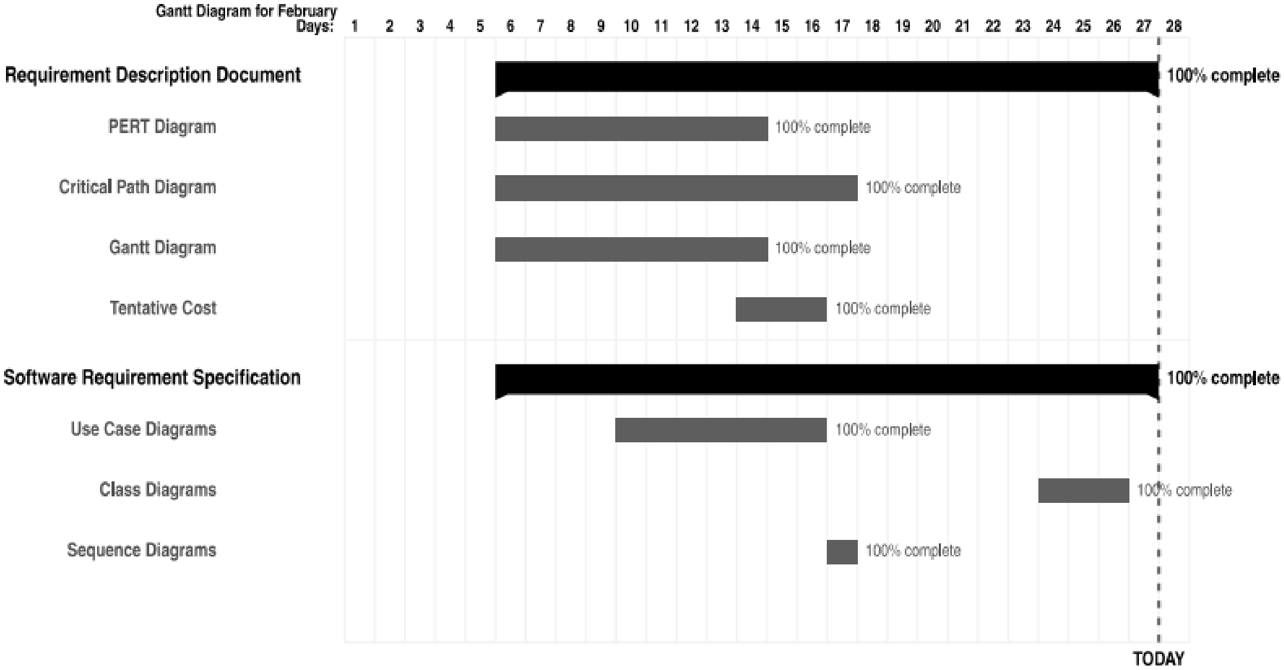
History of Work, Current Status, and Future Work

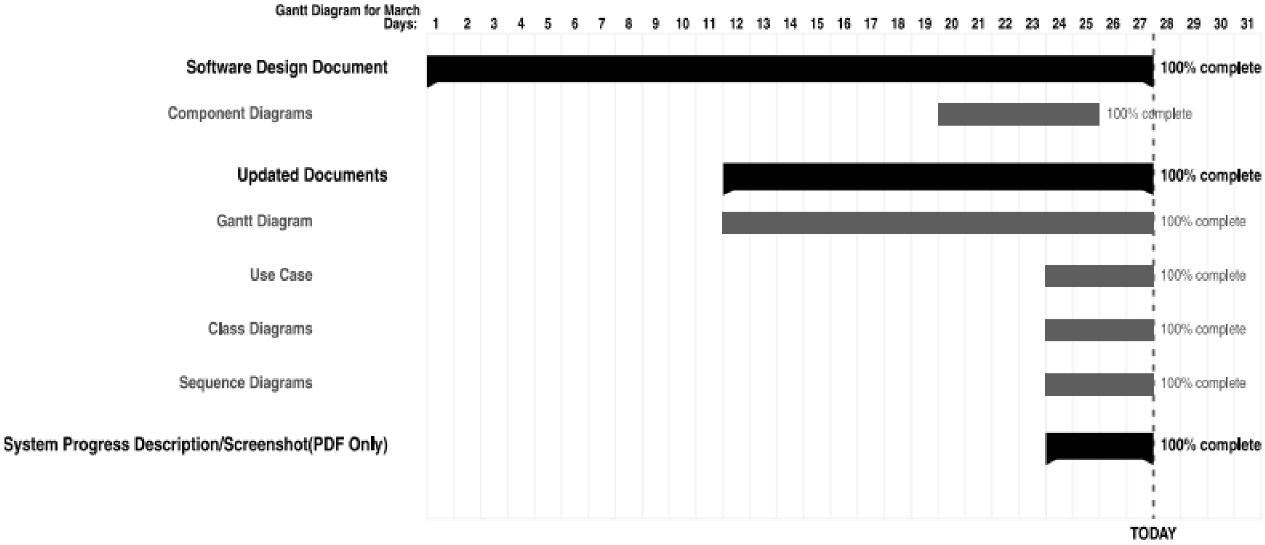
During this section we will provide the reader with a history of the work. This will provide a timeline of what and when certain tasks were completed. Provided below is a Gantt Diagram. This provides that timeline. In addition, we will complete the work by stage later in this section. Following that we will provide our current status. Finally, we will discuss future work. As we have learned throughout this course, an important aspect of software engineering is Maintenace. While most of the requirements provided have been met during this project, continued maintenance and improvement is crucial to nearly all business. Therefore, the system must be constantly improved to meet the needs of the business. Furthermore, the subsection “Future Work”, will cover what could be improved.

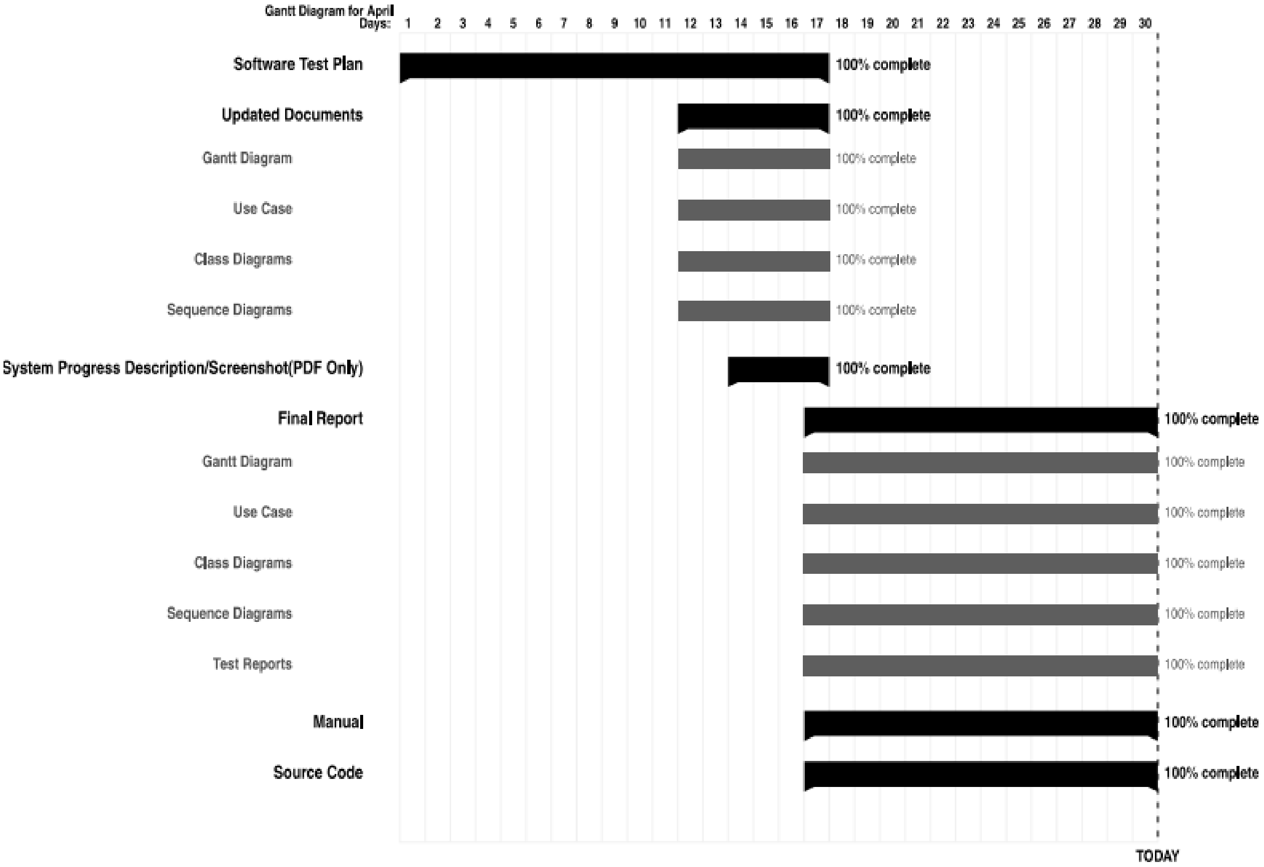
History of Work

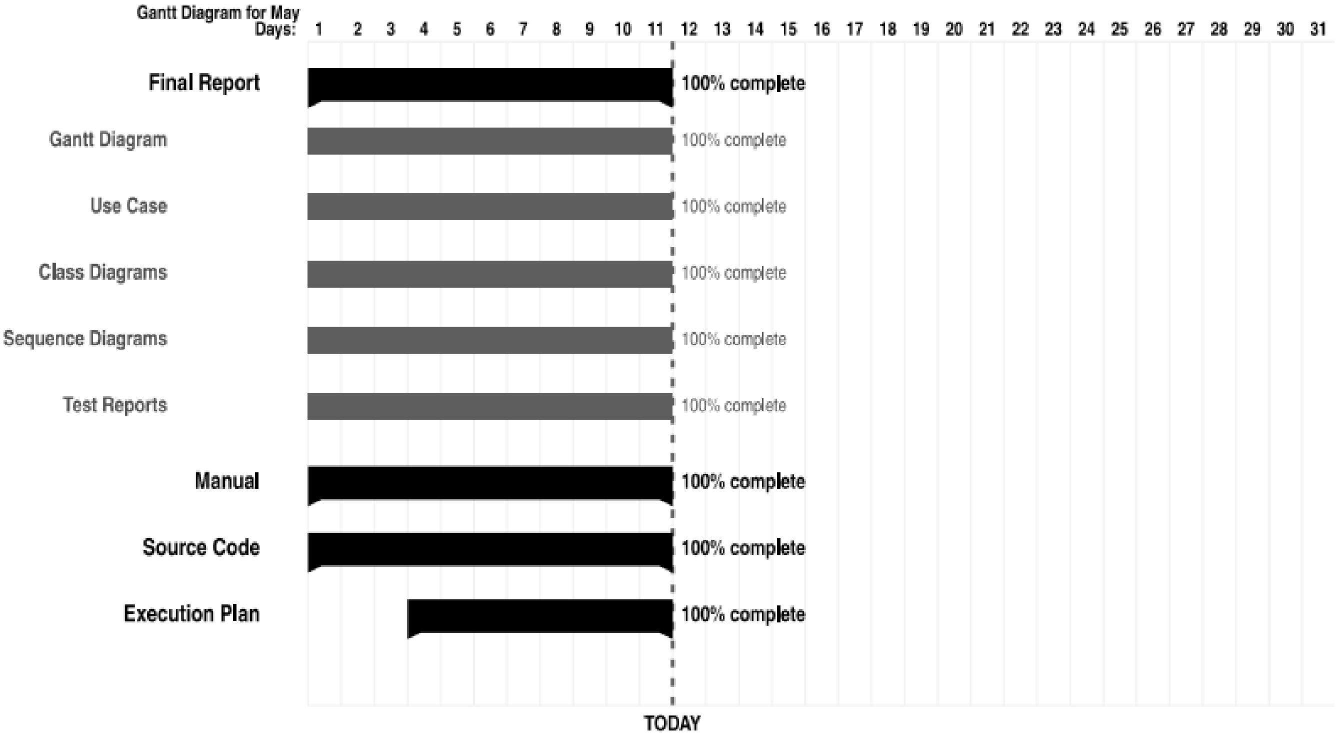
Provided below is a Gantt Diagram. As you can see, the Gantt diagram provides the tasks that need to be completed for the project. In addition, we can see when the task was completed and what percentage of the task needs to be completed.

Gantt Diagram






Gantt Diagram for April
Days: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Software Test Plan
100% complete
Updated Documents
100% complete
Gantt Diagram
100% complete
Use Case
100% complete
Class Diagrams
100% complete
Sequence Diagrams
100% complete
System Progress Description/Screenshot(PDF Only)
100% complete
Final Report
100% complete
Gantt Diagram
100% complete
Use Case
100% complete
Class Diagrams
100% complete
Sequence Diagrams
100% complete
Test Reports
100% complete
Manual
100% complete
Source Code
100% complete
TODAY


Gantt Diagram for May
Days: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Final Report
100% complete
Gantt Diagram
100% complete
Use Case
100% complete
Class Diagrams
100% complete
Sequence Diagrams
100% complete
Test Reports
100% complete
Manual
100% complete
Source Code
100% complete
Execution Plan
100% complete
TODAY

Stage 0 (Start Date-February 6, 2022):

During this stage we were able to email and communicate with classmates to form our group of four. During this stage we were able to evaluate each other's knowledge of coding, databases, and other relevant skills. From this we were able to decide which programming languages, database type, and server type to use to complete this project. This ensured we were all comfortable with the tools we would be utilizing to complete the project "Theatre Portales."

Stage 1 (February 6, 2022-February 27, 2022):

Stephen Washington was able to lead this stage and able to distribute work among the group. We were able to design graphs such as the Pert, Critical path, Gantt, Use case, Class, and Sequence diagrams. In addition, we were able to create documents such as Tentative costs and the Requirement Description Document, also known as RDD. Unfortunately, due to team misunderstandings, we were unable to complete the Statement of Work Document and Requirements document by the deadline provided.

Stage 2 (February 27, 2022- March 27, 2022):

This stage was led by Jeru Sanders. During this stage we were able to structure a Software Design Document, also known as the "SDD" or "SDP", which stands for software test plan. This was done in collaboration, using the Microsoft 365 application OneDrive to share the document and edit the document simultaneously. This sped up the process of constructing the document in a timely manner. We also updated previously generated diagrams such as the Use case, Class, Sequence, and Gantt diagrams. Screenshots were taken and provided with a concise description of our software's progress. In addition, we decided to split into groups of two. One group worked on front-end software development and the other on the back-end software development. During this stage we were able to make noteworthy progress with the system.

Stage 3 (March 27, 2022-April 17, 2022):

This stage was led by Garry Callis. Using similar applications in Stage 2 we were able to collaborate and construct a Software Test Plan, also known as "STP". This document was used to address clients and functional requirements needed in the software. During this stage we continued to develop our system "Theatre Portales." During this stage we started to integrate more front-end and back-end components. After a significant amount of integration between front-end and back-end software, we were able to address components in the software that would need to be tested. This document was designed to ensure all requirements are begin test, by doing so we can guarantee the system will meet all requirements specified.

Current Status

Stage 4 has been included in this subsection, due to this stage being the final stage of the project prior to the presentation and submission. This subsection will provide you with information about our current status and progress with the project “Theatre Portales”.

Stage 4 (April 17, 2022- May 11, 2022):

Michael Gonzales is currently leading this stage. So far, we have developed a mostly complete system. We are using Microsoft 365 applications (OneDrive), to collaborate on documents such as the final report and the software manual. The final report includes updated Gantt, Use case, Class, and Sequence diagrams. In addition, the final report will have test reports. The test report is different from the test plan. It includes specific tests for certain components. The test report explains what the requirements and components are, what is expected from the system, and what happened while testing. After that, we also created a “User Manual,” this provides the user with brief instructions on how to use the final software.

Future Work

On the 11th of May, Michael will present the completed system to the class for evaluation. Although some other teams in Team 4 are likely to assist with the presentation. Depending on the evaluation received from the class or client (professor), corrections to the software may be required. If the system is in fact in use by a business, continued maintenance and improvement may be needed. However, these future requirements are not mandatory, as the course ends on the 11th, but can be used as a practice opportunity for Team 4

If we were to continue the project after the course here are some future tasks we could complete:

- Creating an email receipt option
- Adding additional Cybersecurity procedures
- Creating an actual process to ringing up a transaction using actual credit cards