SOFTWARE DESIGN SPECIFICATION TEMPLATE

1.0 Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for the software.

1.1 Goals and objectives

The objective of the software is to complete the following requirements:

i. A graphical system to select seats, where the sold seats must be marked in red and the available seats must be marked in green.
ii. The system must display all plays and their show times and dates within a year.
iii. A customer must be able to select multiple seats from one or many shows.
iv. A customer must be able to add a seat(s) to the shopping cart.
v. The system needs to have a customer registered to purchase seats.
vi. The shopping cart should be editable for the customer.
vii. The system must inform the user of a successful transaction showing the seats specific to a play.
viii. The system must always be online.
ix. A theater admin can create, update, and delete seat prices for a specific play.
x. A theater admin must be able to generate reports of sales for a specific play.

1.2 Statement of scope

This project entails creating a system that will allow a customer to purchase tickets from the theater Los Portales. The system will need to be hosted over the internet to allow purchase of tickets 24 hours a day. Likewise, any theater admin needs to have access with an administrator account to update, create and delete seat prices for shows. They also need the ability to generate reports on ticket sales for any show.

The expected time frame to complete the functioning system is estimated to be six to seven weeks to ensure all objectives in section 1.1 are met and completed. Any additional requirements could potentially add one week per extra requirement.

1.3 Software context

The software is intended to work as ticketing system for customers so that they may purchase tickets for the show before they arrive. The theater admin will be able to create shows, set seat prices, and generate reports for each show that will report the number of seats sold, seats left, and revenue generated. This software only provides what is listed in the requirements in section 1.1 and is intended function as a sales platform for tickets and nothing further than those requirements in section 1.1.

1.4 Major constraints

Major constraints to the system are expected to be the following:

- Database creation and setup with the system
- Testing the database with the system to ensure proper create, read, update, and delete is function properly
- Ensuring the frontend GUI is visualized neatly

2.0 Data design

The Los Portales Theater is a C# based Model View Controller (MVC) web application divided into two parts: client-side and server-side.

2.1 Internal software data structure

On the client side, user interaction with the data is managed by various Views presented via web forms. The client side will be implemented using a combination of HyperText Markup Language (HTML) and JavaScript. Data is organized in various tables stored in a MySQL database, a breakdown the various tables and columns are described later within this Design Section (*Database description*). The data will be retrieved from the database via HTTP Get requests from the Server upon webpage initialization.

The data structure on the server will be implemented using C#. Data to be sent to the MySQL database is exchanged with embedded SQL statements (i.e. Merge, Delete, Select, etc.).

2.2 Global data structure

The key data components for this web application revolve around a centralized database. The webpage will request data from the server via various HTTP Get and Post methods.

2.3 Temporary data structure

Temporary data structures in the Los Portales Theater webpage refer to the various SQL queries and scripts which will only be run based on the source transaction.

2.4 Database description

Below are data points that will need to be housed within a database:

1. **Play (Table Structure)**

```
CREATE TABLE [dbo].[PlayHeader](

[PlayID] [int] IDENTITY(1,1) NOT NULL,

[PlayName] [varchar](128) NOT NULL,

[PlayDate] [datetime] NOT NULL,

[PlayTime] [datetime] NOT NULL,

[CreateDateTime] [datetime] NULL,

[ModifiedDateTime] [datetime] NULL,

CONSTRAINT [PK_PlayID] PRIMARY KEY CLUSTERED

(

[PlayID] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]
```

2. **Seat (Table structure)**

```
CREATE TABLE [dbo].[PlaySeats](

[SeatID] [int] IDENTITY(1,1) NOT NULL,

[PlayID] [int] NOT NULL,

[Price] [float] NOT NULL,

[CustomerID] [int] NULL,

[IsSold] [int] NULL,

[ModifiedDateTime] [datetime] NULL,
CONSTRAINT [PK_SeatID] PRIMARY KEY CLUSTERED

(

[SeatID] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]
```

3. **Customer (Table structure)**

```
CREATE TABLE [dbo].[Customer](

[CustomerID] [int] IDENTITY(1,1) NOT NULL,

[FirstName] [varchar](128) NULL,

[LastName] [varchar](128) NULL,

[Street] [varchar](100) NULL,

[State] [varchar](128) NULL,

[Zip] [int] NULL,

[EmailAddress] [varchar](200) NULL,

[UserName] [varchar](128) NOT NULL,

[Password] [varchar](128) NOT NULL,

[ActiveFlag] [int] NULL,

[CreateDateTime] [datetime] NULL,

[ModifiedDateTime] [datetime] NULL,

CONSTRAINT [PK_Customer] PRIMARY KEY CLUSTERED

(

[CustomerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]
```

4. **<u>Transaction/TicketSales (Table structure)</u>**

```
CREATE TABLE [dbo].[TicketSales](

[TransactionID] [int] IDENTITY(1,1) NOT NULL,

[PlayID] [int] NOT NULL,

[CustomerID] [int] NOT NULL,

[TotalSale] [float] NOT NULL,

[IsRefunded] [int] NULL,

[RefundDateTime] [datetime] NULL,

CONSTRAINT [PK_TransactionID] PRIMARY KEY CLUSTERED

(

[TransactionID] ASC

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]

) ON [PRIMARY
```

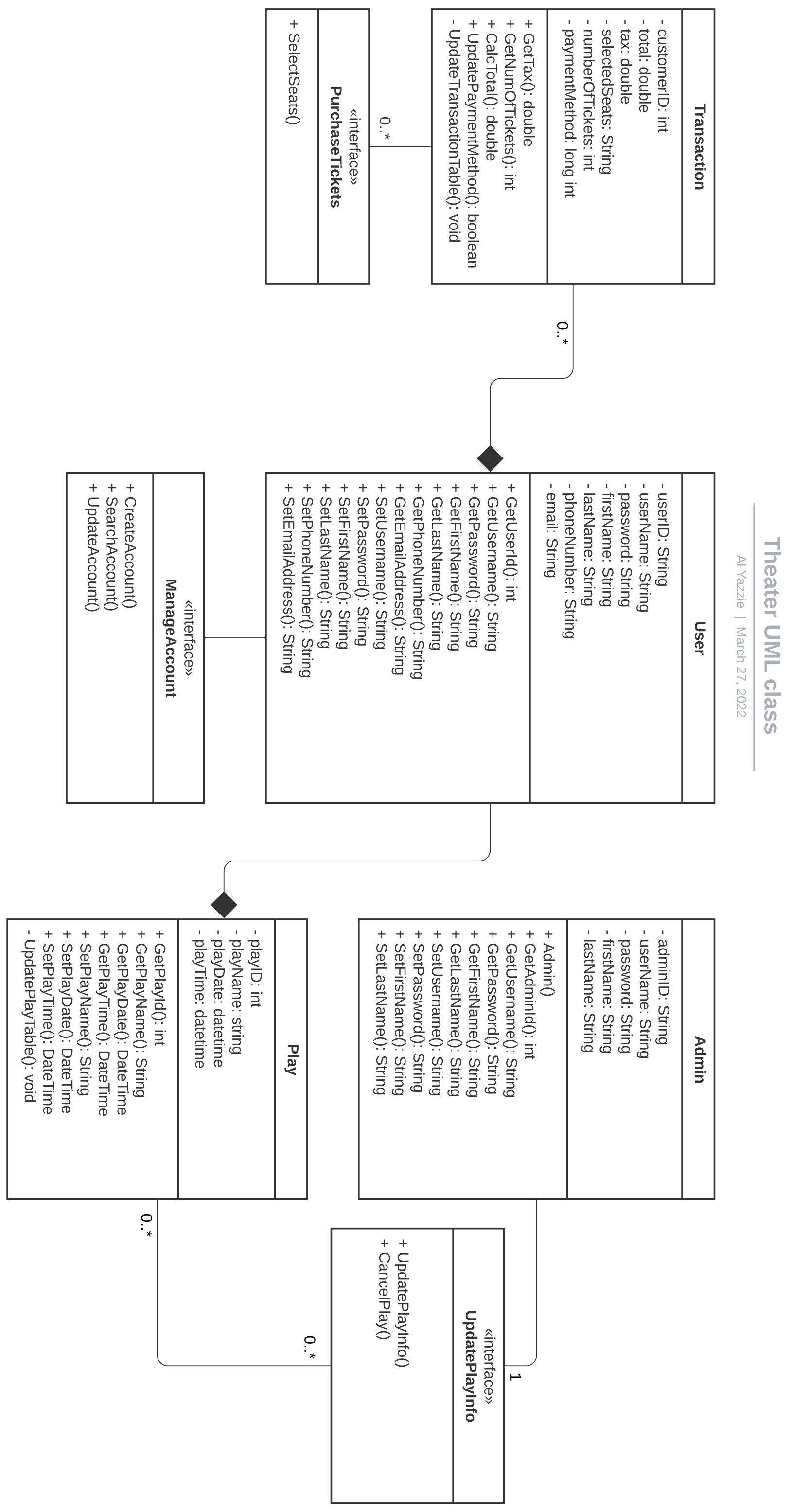
The above tables are crucial data points that the system will need to store for customers to purchase tickets and for theater admins to generate reports and create plays to assign seat prices too.

3.0 Architectural and component-level design

<u>3.1 System Structure</u>

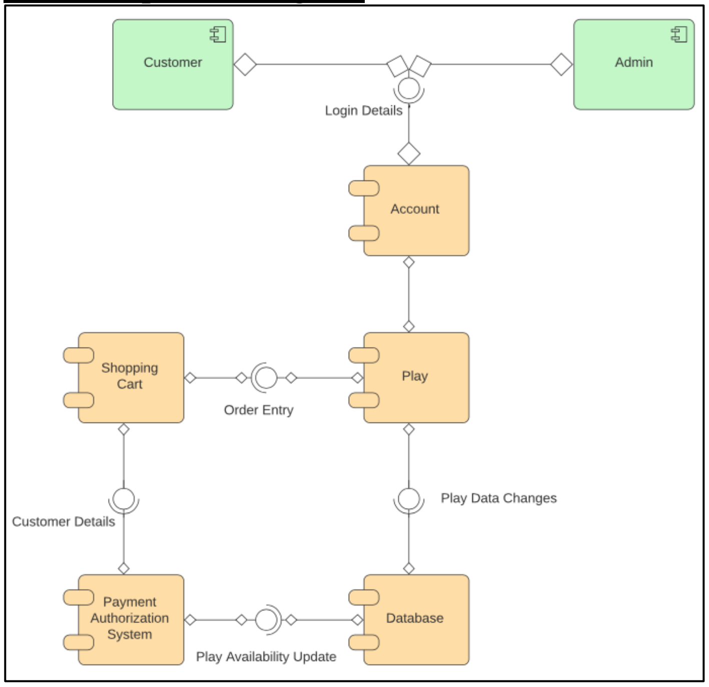
The Los Portales Theater web application is split into two components: an HTML client-side application and server-side C# application and MySQL database.

3.1.1 Architecture Diagram



Project Leader: Al Yazzie

3.1.2 Component Diagram



3.2 Play Class

The Play Class represents the Los Portales theater play details. The class is comprised of several data points including a unique identifier for each Play, the Play Name, Play Date, Play Time, when the admin created the record and if/when a play is modified.

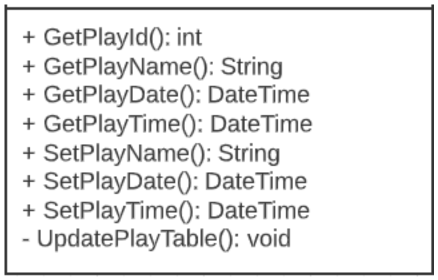
3.2.1 Processing narrative

When the Admin adds a new play to the system, a Play object is created. This object is responsible for storing information unique to the Play. This includes the following:

- PlayID
- PlayName
- PlayDate
- PlayTime

Any time information related to a Play is required, the Play object is called. For example, the Admin needs to create or modify Play information, each are identified using its unique PlayID. When a new Play is created/modified, the current date and time are captured and written to the database to the CreateDateTime and/or ModifiedDateTime column for tracking changes to the database.

3.2.2 Play interface description



```
+ GetPlayId(): int
+ GetPlayName(): String
+ GetPlayDate(): DateTime
+ GetPlayTime(): DateTime
+ SetPlayName(): String
+ SetPlayDate(): DateTime
+ SetPlayTime(): DateTime
- UpdatePlayTable(): void
```

3.2.3 Processing detail

The only methods in the class are accessors, mutators, and a method to interact with the database.

3.2.3.1 Design Class hierarchy

The Play class has no parent or child classes.

3.2.3.2 Restrictions/limitations

There are no restrictions.

3.2.3.3 Performance issues

At any one time, there is only one Play object created or modified by the admin therefore no performance issues are present. The only issue that could arise would occur if the database server is being serviced thus, temporarily halting incoming and outgoing connections until the server is operational again.

3.2.3.4 Design constraints

The major constraint for this class is each new Play object will need to include a Play Name, Play Date and a Play Time. The table on the database does not allow null values in these fields.

3.2.3.5 Processing detail for each operation

- UpdatePlayTable() : void
    - Method to write to the database based on the operation from the mutator methods
- Accessors/mutators
    - These methods are used to get and set Play object information as needed. All fields can be modified except the PlayID which is a primary key on the table.

3.3 User Class

The User Class represents the Los Portales customers. The class is comprised of several data points including a unique identifier for each User, a Username, Password, First Name, Last Name, Phone Number and Email address when a User object is created or when a User object is modified. This class is utilizing built in MVC Identity functionality for the creation of Roles within the application

3.3.1 Processing narrative

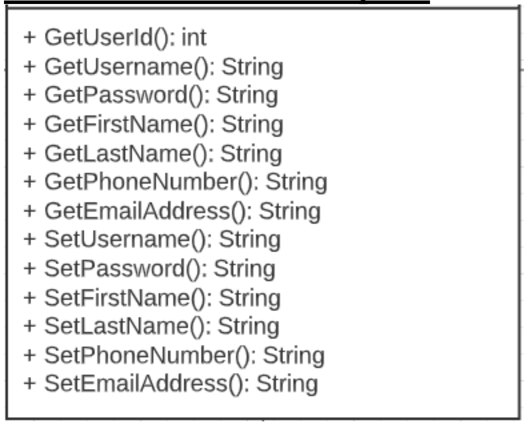
When a new Account is created a new User object is created. This object is responsible for storing information unique to each User. This includes the following:

- UserID
- UserName
- Password
- FirstName

- LastName
- PhoneNumber
- Email

Any time information related to a User is required, the User object is called and identified using its unique UserID. When a new Account is created/modified, the current date and time are captured and written to the database to the CreateDateTime and/or ModifiedDateTime column for tracking changes to the database.

3.3.2 User interface description



```
+ GetUserId(): int
+ GetUsername(): String
+ GetPassword(): String
+ GetFirstName(): String
+ GetLastName(): String
+ GetPhoneNumber(): String
+ GetEmailAddress(): String
+ SetUsername(): String
+ SetPassword(): String
+ SetFirstName(): String
+ SetLastName(): String
+ SetPhoneNumber(): String
+ SetEmailAddress(): String
```

3.3.3 Processing detail

Several methods within this class are accessors or mutators.

3.3.3.1 Design Class hierarchy

The User class has no parent or child classes.

3.3.3.2 Restrictions/limitations

There are no restrictions.

3.3.3.3 Performance issues

Several User accounts may be created from several sources which could potentially decrease available resources, nonetheless the UserID primary key on the table would prevent duplicate accounts. Like the Play class, the only issue that could arise would occur if the database server is being serviced thus, temporarily halting incoming and outgoing connections until the server is operational again.

3.3.3.4 Design constraints

The major constraint for this class is each new User object will need to include a Username and a Password. The table on the database does not allow null values in these fields.

3.3.3.5 Processing detail for each operation

- Accessors/mutators
  - These methods are used to get and set User object information as needed. All fields can be modified except the UserID which is a primary key on the table.

3.4 Transaction Class

The Transaction Class represents the Shopping Cart and a Finalized processed payment within the Los Portales webpage. The class is comprised of several data points including a unique identifier for each User, the Number of Tickets, Selected Seats, the Payment Method, Tax, and a Total when a User purchases a ticket.

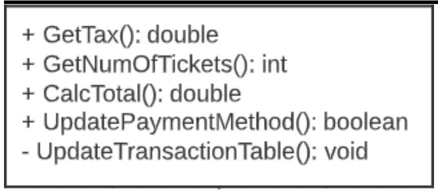
3.4.1 Processing narrative

When a purchase is about to occur, this object is responsible for storing information unique to each Transaction. This includes the following:

- CustomerID
- Total
- Tax
- SelectedSeats
- NumberOfTickets
- PaymentMethod

Before a Transaction is completed, each Transaction is labeled with a unique TransactionID.

3.4.2 Transaction interface description



```
+ GetTax(): double
+ GetNumOfTickets(): int
+ CalcTotal(): double
+ UpdatePaymentMethod(): boolean
- UpdateTransactionTable(): void
```

3.4.3 Processing detail

There are two accessor methods which work with Calculating the Total Sale. The last is a method to interact with the database.

3.4.3.1 Design Class hierarchy

The Play class has no parent or child classes.

3.4.3.2 Restrictions/limitations

There are no restrictions.

3.4.3.3 Performance issues

Several Transactions can be performed from several sources which could potentially decrease available resources, nonetheless the TransactionID primary key on the table would still separate each transaction with their own identifier. Like the Play class, the only issue that could arise would occur if the database server is being serviced thus, temporarily halting incoming and outgoing connections until the server is operational again.

3.4.3.4 Design constraints

The major constraint for this class is each new Transaction will need to include the PlayID, the UserID and the Total. The table on the database does not allow null values in these fields. This is also to link which user purchased each ticket.

3.4.3.5 Processing detail for each operation

- CalcTotal() : double
  - Method used to calculate the total sale based on the number of tickets and tax.
- UpdatePaymentMethod() : Boolean
  - Method to update stored Payment method
- UpdateTransactionTable() : void
  - Method to write to the database based on the operation from the mutator methods

- Accessors/mutators
    - These methods are used to get Play information as needed. All fields can be modified except the TransactionID which is a primary key on the table.

3.5 Admin Class

The Admin Class represents the Los Portales customers and the theater admin. The class is comprised of several data points including a unique identifier for each Admin, a Username, Password, First Name and Last Name and when an Admin object is created or when an Admin object is modified. This class is utilizing built in MVC Identity functionality for the creation of Roles within the application.

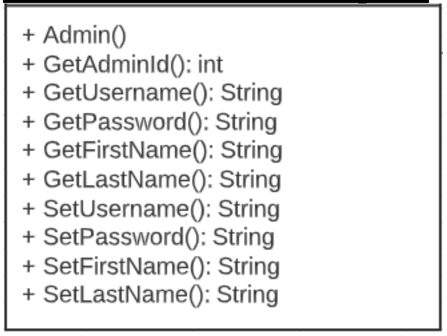
3.5.1 Processing narrative

When a new Admin is created a new Admin object is created. This object is responsible for storing information unique to each User. This includes the following:

- AdminID
- UserName
- Password
- FirstName
- LastName

Any time information related to a User is required, the User object is called and identified using its unique UserID. When a new Account is created/modified, the current date and time are captured and written to the database to the CreateDateTime and/or ModifiedDateTime column for tracking changes to the database.

3.5.2 User interface description



```
+ Admin()
+ GetAdminId(): int
+ GetUsername(): String
+ GetPassword(): String
+ GetFirstName(): String
+ GetLastName(): String
+ SetUsername(): String
+ SetPassword(): String
+ SetFirstName(): String
+ SetLastName(): String
```

3.5.3 Processing detail

Several methods within this class are accessors or mutators. There is one default constructor method in this class.

3.5.3.1 Design Class hierarchy

The Admin class has no parent or child classes.

3.5.3.2 Restrictions/limitations

There are no restrictions.

3.4.3.3 Performance issues

Admin account creations are uncommon, unlike User accounts, nonetheless the AdminID primary key on the table would prevent duplicate accounts. Like the Play class, the only issue that could arise would occur if the database server is being serviced thus, temporarily halting incoming and outgoing connections until the server is operational again.

3.3.3.4 Design constraints

The major constraint for this class is each new Admin object will need to include a Username and a Password. The table on the database does not allow null values in these fields.

3.3.3.5 Processing detail for each operation

- Default Admin constructer
- Accessors/mutators
  - These methods are used to get and set Admin object information as needed. All fields can be modified except the AdminID which is a primary key on the table.

4.0 User interface design

A user interface is crucial to this software as it will help make the process of purchasing a ticket much easier for the customer. As of right now we have identified 8 users interfaces that are critical to ensure the task of purchasing a ticket and ensuring the theater admin can complete their job.

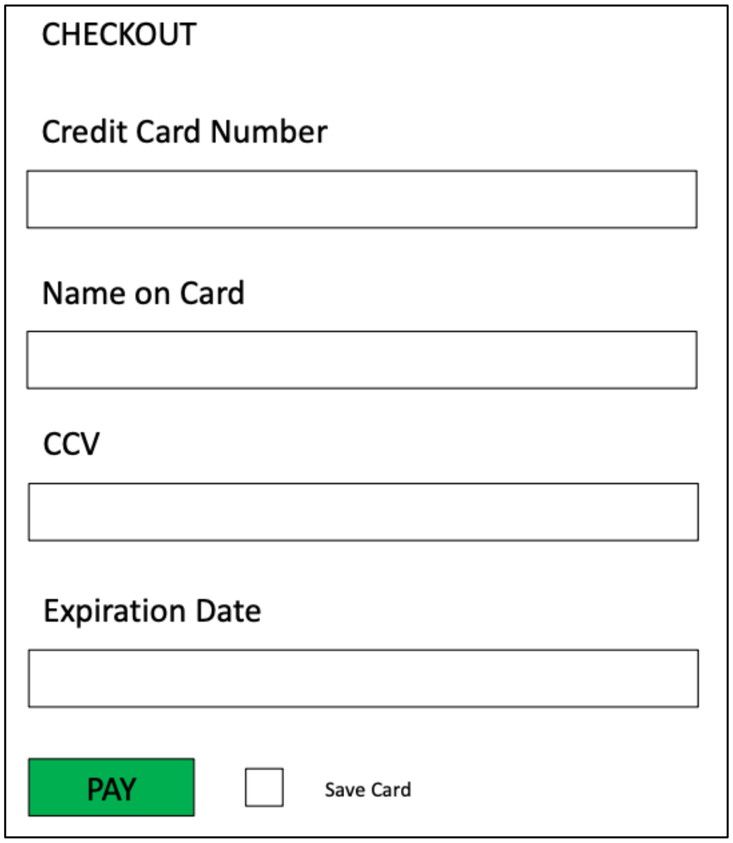
4.1 Description of the user interface with Objects and Actions

The user interface will display the following:

1. Log in page for the user
   a. Username
   b. Password
2. Register page where user can create an account
   a. Password
   b. Name
   c. Date of birth
   d. Email address
   e. Address
3. A list of plays for the user which they can click on to view the seating chart
4. Seating chart for each play where each seat has the price listed and an available seat is green and taken seat is red.
5. A check out page
6. A shopping cart for tickets must be editable and the user can delete items
7. A sales report page for the theater admin can select a play from a drop-down menu. The interface will also ask the admin which fields the admin wants to know about in a check box: seats sold, seats available, total revenue.
8. A page for plays in which the theater admin can Create, Delete, Update plays.
   a. Create page – Admin can enter in the name, date, and time of play

b. Delete page - Admin can delete the play
c. Update Play - Admin can update the name, date, time, and seating prices

4.1.1 Screen images - Los Portales Theater GUI



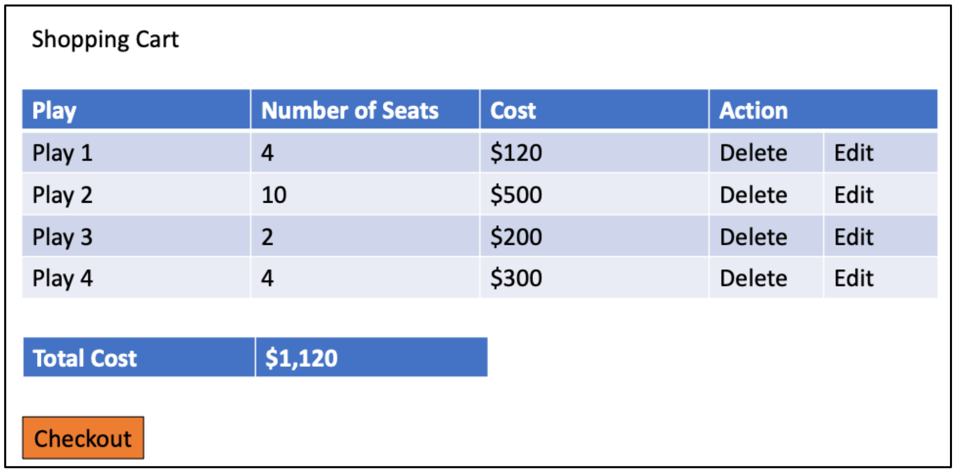
CHECKOUT

Credit Card Number

Name on Card

CCV

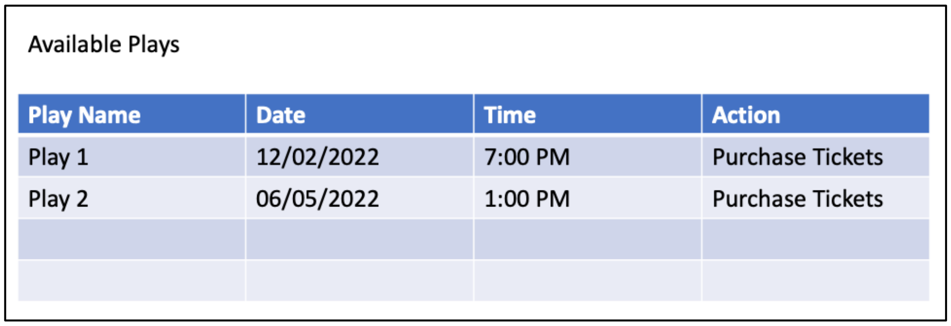
Expiration Date



Shopping Cart

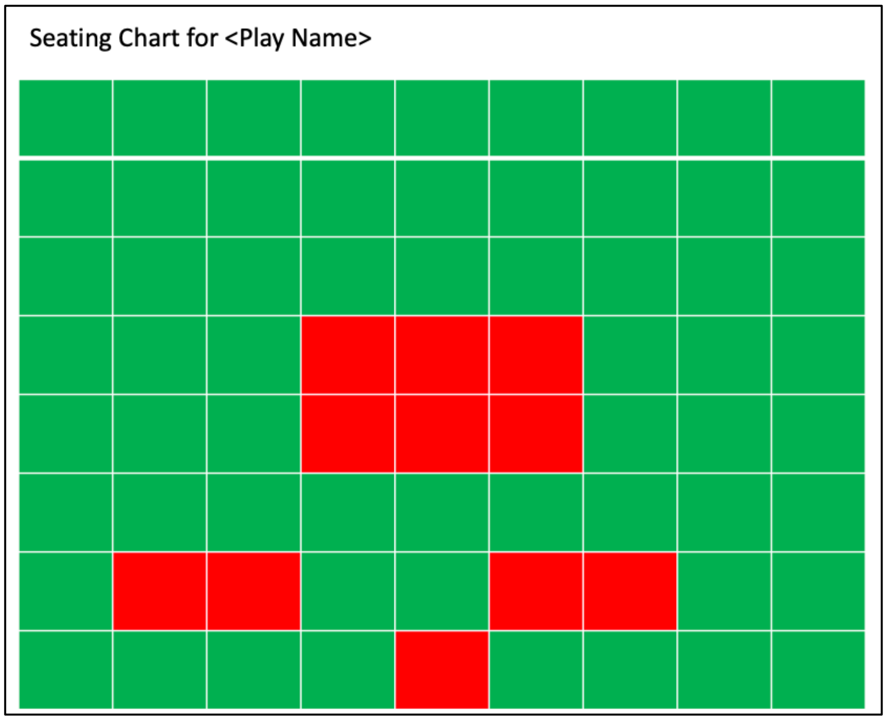
Play	Number of Seats	Cost	Action	
Play 1	4	$120	Delete	Edit
Play 2	10	$500	Delete	Edit
Play 3	2	$200	Delete	Edit
Play 4	4	$300	Delete	Edit

Total Cost	$1,120



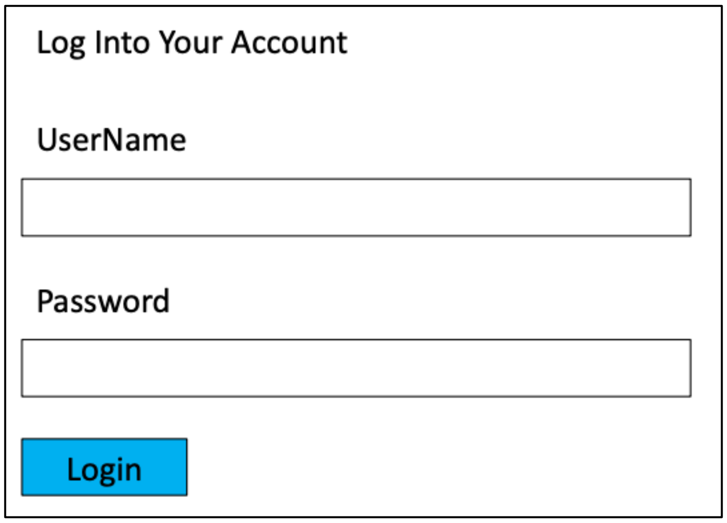
Available Plays

Play Name	Date	Time	Action
Play 1	12/02/2022	7:00 PM	Purchase Tickets
Play 2	06/05/2022	1:00 PM	Purchase Tickets



Seating Chart for <Play Name>

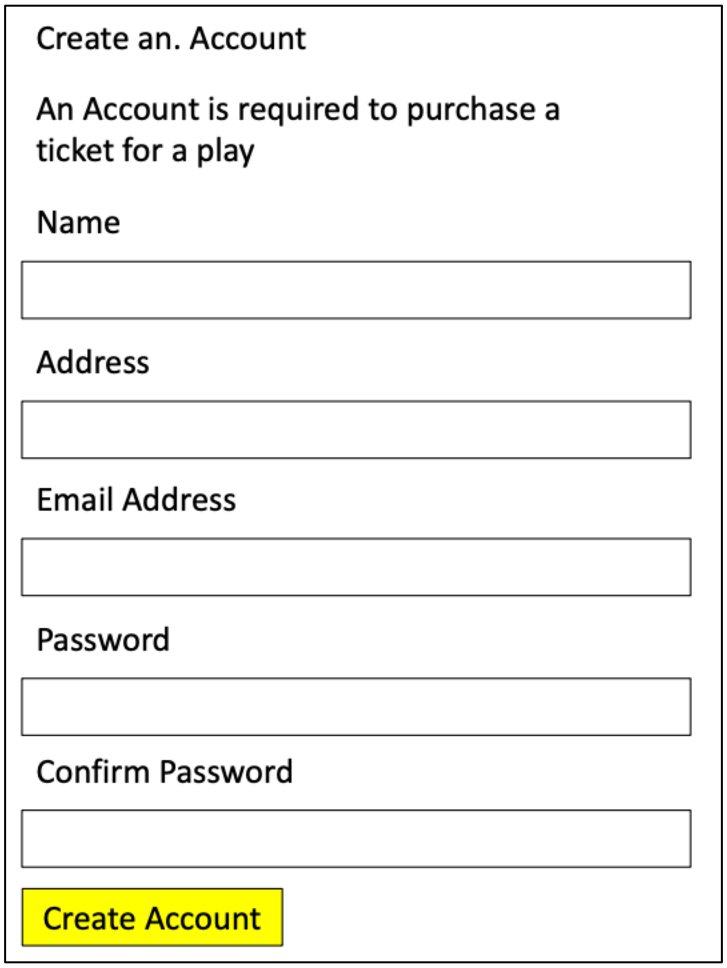
🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩
🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟩	🟩
🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
🟩	🟥	🟥	🟩	🟩	🟥	🟥	🟩	🟩
🟩	🟩	🟩	🟩	🟥	🟩	🟩	🟩	🟩



Log Into Your Account

UserName

Password



Create an. Account

An Account is required to purchase a ticket for a play

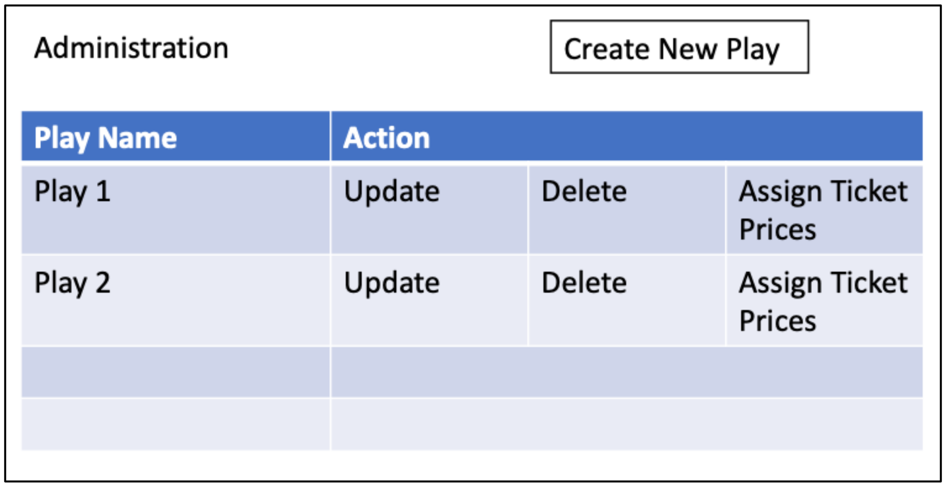
Name

Address

Email Address

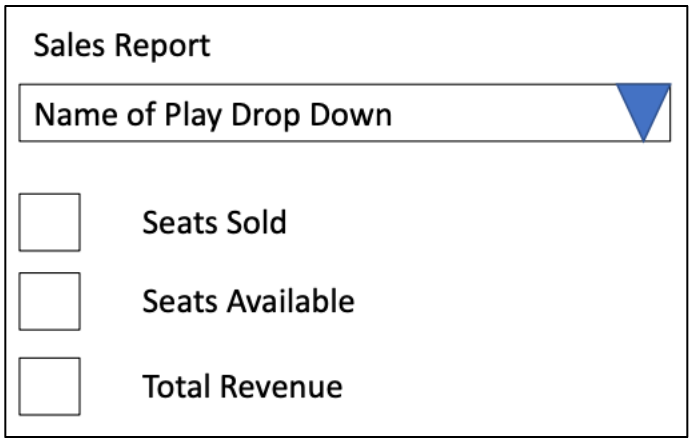
Password

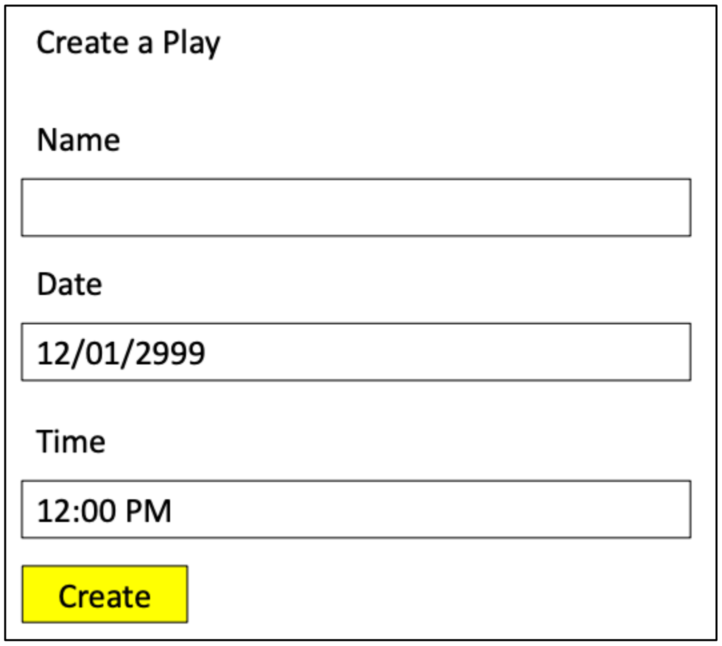
Confirm Password



Administration

Play Name	Action		
Play 1	Update	Delete	Assign Ticket Prices
Play 2	Update	Delete	Assign Ticket Prices





4.2 Interface design rules

The user interface shall follow the rules below:

1. Consistency
2. Use of shortcuts
3. Provide feedback to the user
4. Design dialog to yield closure
5. Simplify to prevent errors
6. Easily reverse actions
7. Give users control
8. Reduce short-term memory load

4.3 Components available

Listed below are GUI component that will be implemented in HTML.

1. Textbox – allows a user to input data to certain data that is being asked from them.
2. Label – Helps label certain elements on the GUI to guide users.
3. List Box – Displays a list of items in which the user can select an option from.
4. Table – Ensures data is presented neatly
5. Checkbox – Allows the user to check a box when input is needed to perform an operation
6. Button – Allows a user to click and then performs an operation after inputting data to a checkbox, textbox, or list box
7. Link – Has a hyperlink that can help user easily navigate to certain areas of the app by embedded a URL

5.0 Restrictions, limitations, and constraints

Time

Due to the project being a 15-week semester for a semester of school, the team has very limited time capacity to build, test and implement a working software to the client. The development phase needs to be extremely monitored as the team should not waste time during this phase to build a working software.

6.0 Testing Issues

To ensure the software executes properly, we need to incorporate testing into our development phase to ensure the software is bug free. The testing should ensure proper input is taken and the proper output is outputted.

6.1 Classes of tests

Here, we describe how we will test the software as it is being developed.

Unit Testing

We will use white box testing and write unit test to ensure each component is functioning correctly. This will be important for the components that interact with the database. Ensuring that data is written correctly to the database is critical to ensure the app functions how the client wants expects it to respond.

Likewise, the development team will write unit cases for actions of registering a user, creating a play, updating a play, removing a play, and completing a transaction. This data is stored in the database, and we must ensure that each action preforms as excepted.

Component Testing

With many different components within this software, it is critical we test each component as well and ensure it functions correctly. For example, we shall test the graphical user interface within ana action that requires a database action. We can perform a unit test for this and generate a script to test the GUI to ensure that it only accepts the correct format for input. Next, we shall validate the information to whatever backend function that needs to execute.

Use-Case testing

We should also test our software by examining our use cases. We should ensure that the use case and the results listed out should match how the system is interacting. This type of testing can ensure the front-end and back-end of the application are communicating properly and the expect results are seen.

User Testing

When the software is functioning as expected and all the above testing is complete, we shall consult the client to test the software. Having the user test the software before releasing it will help guide the team into what the client needs, and the requirements are met. Having a clean and user-friendly interface will help the user navigate the software and the client should have final say in how they want the software to look so that they may use the software without distractions.

6.2 Expected software response

There should be expected results from the software while testing. While testing actions that connect to the database, we should expect these methods will write when a write action is performed, a read action indeed reads from the database, and a delete method deletes the proper data from the database. There should also be ways the software ensures a failed action doesn't not edit the database and informs a user that an action failed and how they should correct it.

6.3 Identification of critical components

Listed below are critical components of the software:

1. **Transaction**
   The transaction component is the backbone of this application. During the test phase this component needs to be tested to ensure it functions properly. That is, we need to make sure a successful transaction is successful, and that failed

transaction doesn't purchase the tickets and does not pull money from a user's card.

2. **Play Creation**
   Ensure the play creation component of the administrator side of the application is critical to the application. If the theater admin cannot create a play, the user cannot purchase tickets for a play. Likewise, this also entails the theater admin can assign prices to the seats so that the user may see them to purchase.
3. **Account Creation**
   Creating an account is another critical component to the software. If a customer cannot create an account, they will have no way to purchase tickets. This module should ensure that a user will always have a way to create an account and give proper feedback if an account cannot be generated and refer the user on how set an account up.