# Ticketing Booking System

**Members:**

**Sufiyan K20-0386**

**Abdeali K20-0122**

**Nausherwan K20-0428**

# Introduction:

**Ticket Booking System** is a program which will help users to book/buy tickets of the movies that are currently available. Users will first have to be registered in order to Book their tickets Projects like these bring great impact in fluidity ad bring economic benefit by making it an ease for users to do taxing work of waiting in lines for hours and driving the sales of the Company.

# Background (Research and Evaluation):

After lengthy talks between many senior members of Fast NUCES and hours spent researching online, we noticed a **common trend of using Databases,** and SQL servers to make this Project (Ticketing Booking System). Databases were mainly used in storing information and passing the information through different forms, Managing and organizing data in which various data relationships exist.

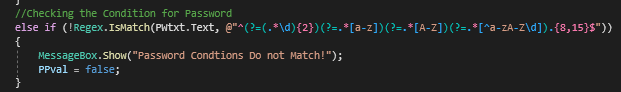
We wanted to overcome this trend and thought of designing this whole project **by using the Features we learn in Object Oriented Programming Course CS217 and using features off course.** This Project was Done on **C# using Window forms**. Key Feature which deviates from C++ Class Feature, is **Partial Classes**. it provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a partial keyword.

# Problem Analysis

In today’s world digitization and technology is recommended in every aspect of life. By taking a look around we can find many important and fundamental issues which are still not addressed the way it should be. When it comes to booking your tickets online, the old methodology is waiting in long lines and trying to buy your ticket, and the biggest uncertainty is that not knowing if tickets are already booked or not, booking online will let you know the availability and can also plan ahead on when you want to watch and book it for that specific day. This project also allows you to select different venues for your comfortability, this removes the need to visit different venues hoping for an availability of your seat.

# Implementation:

When user enters the Program, he is shown **two Options**: Log in and Sign up. New Users would have to go through the **Sign-up stage** which consist of fields user has to fill in order to process to the Next Page. Each Field has **validation** check which makes sure if Character length meets its requirement for certain fields such as Username, and checks if Password Strength meets the conditions set by the Program. All this Validation is done by using a **class called Regex** also known as Regular Expression, it is a kind of ‘Grammar’ which is used to parse text (strings). **The regex equation** is set in such a way that program reads the equation and tries to **find patterns in the text you have written.** It has strict grammar you need to follow in order to make sure program reads the pattern correctly. **File handling** has also been implemented to verify Username used. *(Explained in the Log-in Section)*



**Regex is a class** we are using and it has a member function called **IsMatch**() which accepts two arguments, first one is the string you want, here we are just taking it from the Textbox where user as typed its password, and the Second argument takes the expression used to parse the text.   
***An overview of what this Expression is currently doing:***

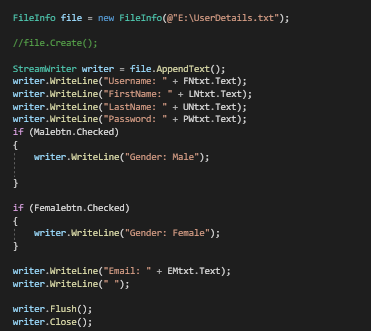
* The **“@”** before the String Apostrophes tells the compiler that to not consider Escape sequences as Escape sequences.
* **The Caret Symbol (^)** Tells the compiler to start the match at the beginning of the string or line.
* **This Symbol (?=)** is a **positive look** **ahead** symbol, which tells the compiler that this condition written within the parenthesis must be matched and only then start comparing further condition stated in the expression. You can see each condition has a positive Look ahead.
* **The curly braces {…}** tells the computer to repeat the preceding character (or set of characters) for as many times as the value inside this bracket.
* **Character Classes:** Have many specifies in place, such as, **/d**; matches any digit in the text, First expression asks user to input Digits and the character Limit set for it is at least **{2}**. Another important one is **/w**; which Matches any word character.
* **.\*[a-z]** - states that user can use any of the characters from a to z any number of times, stated by the **Quantifier or the Asterisk sign \*** which tells that *preceding characters is to be used 0 or more times,*
* We have already stated a limit **{8,15}** that there should be at least 8 characters till 15 characters used in your password which include the 2 numbers, Letters and also wants you to input special symbols. Which is stated by **(?=.\*[^a-zA-z/d])** There is a **negation sign(^)** which tells the compiler **to use anything except** the group of characters mentioned inside.
* **The Dollar ($) sign** at the end tells the compiler to Check the **{8, 15}** character condition at the end of the String;

The User is finally in the **log-in section**. The User only needs to enter his Username and Password. **System does verify** to make sure that the password entered matches the username entered during the sign-up stage. All of this process involves **file handling**. The program first reads the file which contains usernames of various users *(username.txt)* *(sign-up implemented the feature to make sure that username selected is Unique, hence no two users can have the same username),* The system is **actively reading** the file as username is being entered to give user a real time feedback (through Text\_changing () function which is part of an Event handler function). The System then **cross checks the user’s password** by checking on the same (*int Line No)* on Password’s file(*Password.txt*) to confirm if it really exist or not. If there is an **Error,** User is given a **Prompt message** that his Username or password **doesn’t match the database** when user clicks on the Log-in Button (Password verification is only done when this Event has occurred). Below is a snippet of the code in action.



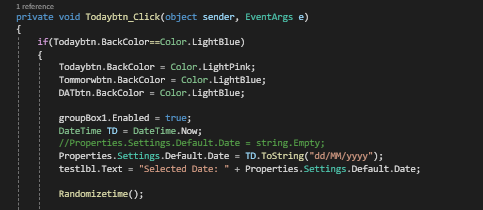
**Stream Reader** is a class and we have created an object *Recieveusername* and *ReceivePassword*, and through its **constructer** we passed down the **File path**. Functionality on how it works is mentioned above. And we have used the method **ReadLine()** to read each line, and used a **for loop to read until it reaches the end of file**, If It does find the data **in between**, then there is **no need** to complete reading the file till the end.

There is also Class known as **Stream Writer** which is used to **write to file.** *Snippet shown below* shows **UserDetails** being saved once the user is **done signing up**. **The Open Mode is Append,** because every user that has visited this Program, their details are stored.  
The **File Info class** provides **properties** and methods that are **used** to create, delete and read file, its mode is also set as well as shown below.



We then Reach the **Home Screen**, where user is given options to navigate to different page where user is given options to navigate different pages for example view their receipt from the homepage and logout back to the login/signup screen, these two options come under a **drop down panel** named *profile*. The other option is given to visit the page where **all the movies are displayed.** The homepage is also inherited by the payment module meaning the final pay button so you will be redirected again to the home screen from there

After **Selection of the Movie** user wants to watch, we are taken to the page the **page where user can select the Venue and the Showtime.** **Features implemented** are Toggle buttons; the system make sures the user only **selects one Time slot**, and **not multiple time slots**, to select a new time slot, he would need to de-select the previously chosen timeslot. *(User is prompted on the form about this condition)*. **Another feature is**; saving User data at Runtime using System Settings



**Using Properties.Settings.Default.Date = TD.tostring(“dd/mm/yyyy”);** what this is doing is saving the Time which is generated from the **Datetime Class** which ***Gets*** the Date and Time and saves it to the object. The current time on the computer is saved.

**Getter function working at its back end**, it is a **static function** hence only occupies single memory block throughout the program, it does not create multiple copies, hence saving processing memory.

public static DateTime Now {get ;}

This functions gets the **Time stored on your computer and stores it in the Object created TD** that object is then used to store the data in **properties** so the program can easily use this information elsewhere in the program. **TD.tostring(“dd/mm/yyyy”);** is basically **converting the integer value stored in it to String**. *(Reason it to easily print it on textboxes which only output strings and no other data type).*

So Here we learnt the method to **store data** in *User settings* and we also learnt how to use *File handling* which was demonstrated in other/previous parts of the program.

User is then taken to **Seat booking system**, the whole booking is an interactive system, where he gets to **view the seating layout**, and can decide the seating position he would like to choose and the Type of Seating mode (Standard or VIP). **Each Seat Button is Hardcoded to detect weather Seat has been selected** or not, if it is selected then it determines overall **how many Standard and VIP seats are selected**, as well as the **Seat number** which is then **saved into a file where it can be retrieved later on the program**. (*Will be required when payment confirmation page is opened).* **One important** point to note is, the User can only proceed on to the next page, once he has selected 1 seat or more, or else the Proceeding (confirm seats) **button will not be enabled.** So Overall**, Validation checks** are in place to make sure an error, or a bug is not carried out further into the program, which later can cause implications.

Once User has Selected his Venue, Showtime, His seats, He is taken on **Confirmation page,** where he can **View all the Details** he has selected and can **proceed to pay** by entering his **Credit Card details**. Once that is done, **An Email is sent to Users Email**, where he can Review his *Ticket details and Show*.



### So how is the Email Sent to the User?

**Namespaces** used for Sending an Email are

*Using System.Net;*

*Using System.Net.Mail;*

We used The class **MailMessage** and create an Object for it, and in the **constructor** of *MailMessage ("COMPANY-EMAIL","USER-EMAIL"),* we **passed** Senders and Users email, Since all users data is stored in **Text files**, And it knows which username matches what username through file Handling, We can easily retrieve user’s email and **store it in the variable** *'useremail'* and use it as a **Receiving -End email.**

using the object **'mm'** made for **MailMessage**, we can call it's ***Methods*** to store the **subject and the body of the emai**l which the Company will be sending**, body will consist of the ticket details user selected during the whole process.**

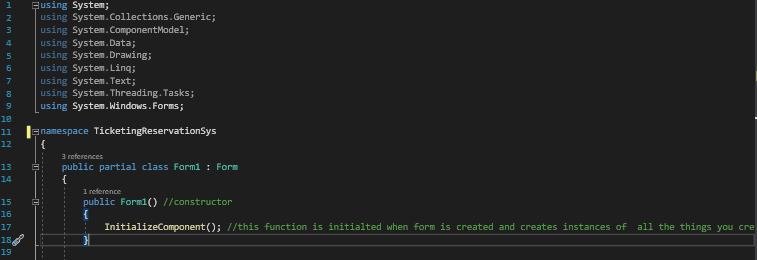
You then **create an object for SmtpClient** which also stands for Simple Mail Transfer Protocol

**To construct and send an email message by using SmtpClient, you must specify the following information:**

* The SMTP **host server** that you use to send email. You will have to assess the Host and Port properties and determine accordingly.
* **Credentials for authentication**, these credentials are your Email password that you normally use to log-in into your Email.
* The **email address of the sender.**
* **The email address or addresses of the recipients**, these are already decided when Mail Message class is made hence it is not necessary to do it here.
* **The message content** is then send using the function **stmp.Send (mm**), this passes down the whole object as its argument

The SmtpClient class has no **Finalize** method, so an application must call **Dispose ()** method to explicitly free up resources.

# Feature Analysis

**When you create a Form:** 

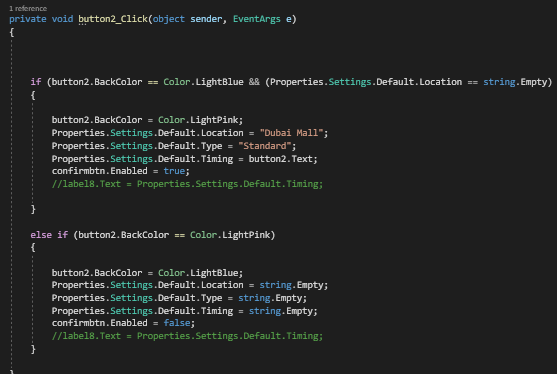
**A Partial Class** is created which is inheriting with the class Form. Form class already exist in header file System.Windows.Forms, which contains member functions which work with all the controls you create on the form.

**Public Form1() {InitializeComponent()}** acts as your **constructor** so when this Form is created, It automatically calls this **default constructor**, which runs this function called **InitializeComponent()**, and what it does it is creates all the **Controls** you placed as soon as you open the form.

What is Control?  
**Controls** are objects that are contained within **form** objects. Each type of **control** has its own set of properties, methods, and events that make it suitable for a particular purpose.

What are Event Handlers?  
**Event Handelers** are Methods in an object that are executed in when an user interacts with his GUI system, or can even occur when a certian conditon is met.

Function shown below **button1\_click(object sender, EventArgs e){ ..},** these functions are created to **carry out a Tasks** when a user clicks a button for suppose as this function handeles exactly that, this functions runs everytime this event is executed, each Control has its own events which can be programmed according to the needs. **Code Segment shown below is the functinality when a button on the form created is clicked.**

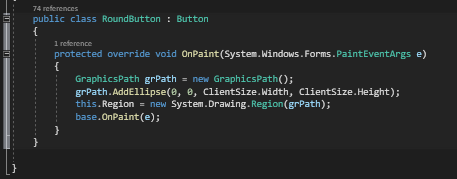


What do these Arguments mean?   
These are the **standard signatures** for every event handler that you create. **The Sender object** is that Object which raised this event at the first place. In the case of static events, this parameter will be null. **EventArgs** represents the data related to that event, Data as in suppose you want labels text to be changed, hence Text will act as the Data related to that event.

How do you Use Access Modifiers in C#?  
You have to state public, private, protected before declaring each variable data type, or return type of a function. By default it’s always private, even if you don’t mention it.

private void FNtxt\_TextChanged(object sender, EventArgs e)

**Overriding a Function** can be done in this way (Snippet below). When overriding, classes have to inherit first and parent class needs to have the function as well, we override when we need to change the functionality of that object. So here we have demonstrated that feature by changing the Design of the button by overriding the function from Button Class. The Buttons are now Round, but are only round if you use Round buttons Control, if you use Button Control, You will have the Default feature. This is quite useful when you are creating your custom controls.



# Class Diagram: