Shawn Snyder

Reservation Application

Date Presented: Feb 21, 2020

A little about me:

Even during high-school I had a conscientious activity to accomplish something by learning computer programming. After I graduated, I was taken in by a company called Kanone Inc. who taught me front end web development. After that I got a part time job as a cook and have been there since. While working I decided to go to college for computer science. After the first semester I learned of Centriq and thought... I keep my college education available and get to learn what I know I want to do. It's a win-win for me.

Things I learned from this project:

By developing this project, I learned how important planning is (database wise), and why developing one you have got to make sure you got everything tied together correctly. I learned to spare no small detail. This experience also made me more familiar with accessing related columns in different tables to display them in the front end. Also… just using DateTime instead of military time in an attempt to save database space, saves from a headache later.

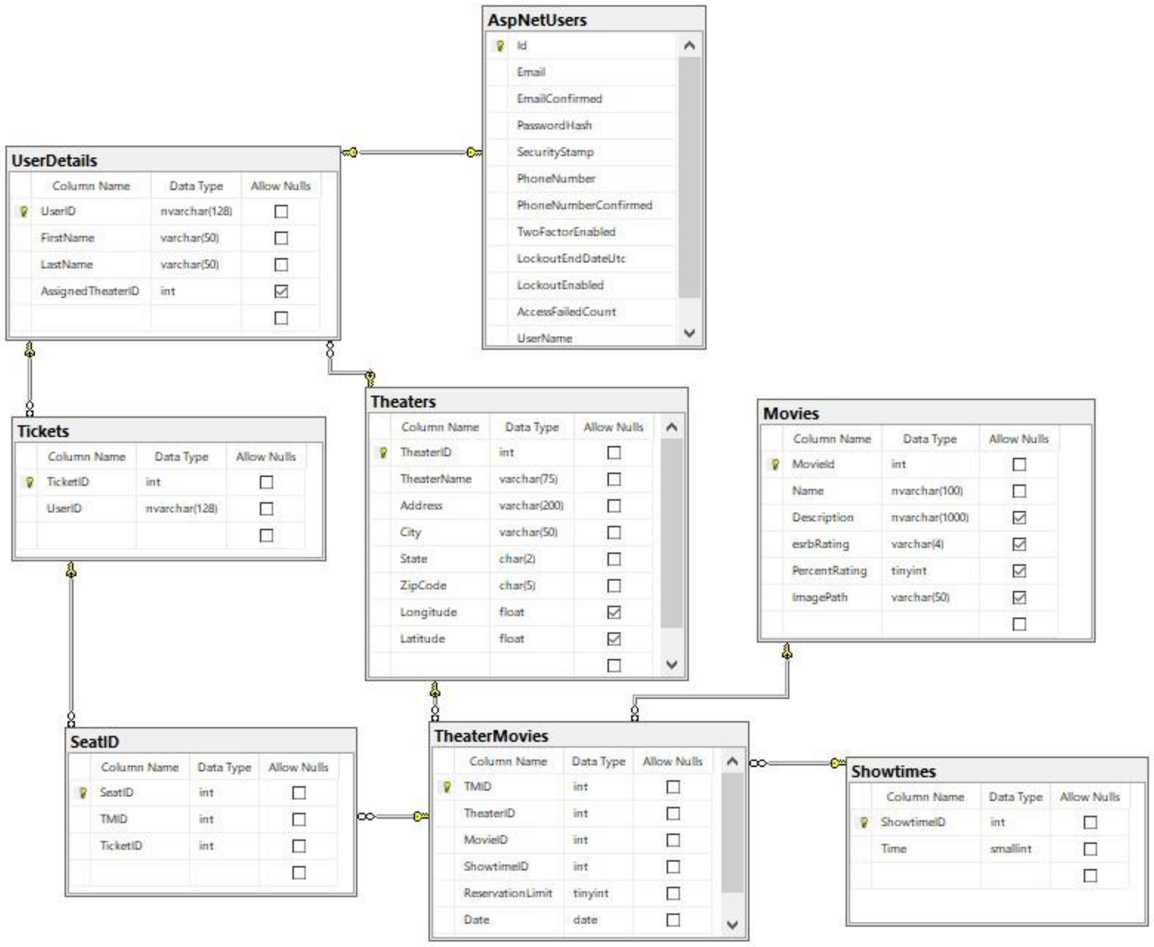
Why choose this project type:

Soon before starting this project I had gone to see about getting some movie tickets one weekend. By ordering them online I thought… “Can I make a webpage that was even more streamlined and easier to do?”.

REQUIREMENTS FOR THE PROJECT:

- Application must have at least 3 roles
- App must work on a mobile device
- Admins must have full CRUD functionality
- Employees must be able to see reservations but NOT edit
- Sensitive information must be shielded from non-authorized users

Database Diagram:



AspNetUsers

Id
Email
EmailConfirmed
PasswordHash
SecurityStamp
PhoneNumber
PhoneNumberConfirmed
TwoFactorEnabled
LockoutEndDateUtc
LockoutEnabled
AccessFailedCount
UserName

UserDetails

Column Name	Data Type	Allow Nulls
UserID	nvarchar(128)	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
AssignedTheaterID	int	☑
		☐

Tickets

Column Name	Data Type	Allow Nulls
TicketID	int	☐
UserID	nvarchar(128)	☐
		☐

Theaters

Column Name	Data Type	Allow Nulls
TheaterID	int	☐
TheaterName	varchar(75)	☐
Address	varchar(200)	☐
City	varchar(50)	☐
State	char(2)	☐
ZipCode	char(5)	☐
Longitude	float	☑
Latitude	float	☑
		☐

Movies

Column Name	Data Type	Allow Nulls
MovieId	int	☐
Name	nvarchar(100)	☐
Description	nvarchar(1000)	☑
esrbRating	varchar(4)	☑
PercentRating	tinyint	☑
ImagePath	varchar(50)	☑
		☐

SeatID

Column Name	Data Type	Allow Nulls
SeatID	int	☐
TMID	int	☐
TicketID	int	☐
		☐

TheaterMovies

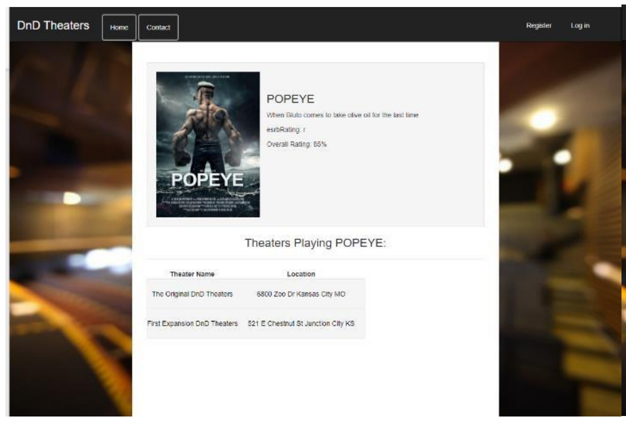
Column Name	Data Type	Allow Nulls
TMID	int	☐
TheaterID	int	☐
MovieID	int	☐
ShowtimeID	int	☐
ReservationLimit	tinyint	☐
Date	date	☐

Showtimes

Column Name	Data Type	Allow Nulls
ShowtimeID	int	☐
Time	smallint	☐
		☐

The Pages Worth Making Note Of:

Selecting a Theater



POPEYE

When Bluto comes to take olive oil for the last time

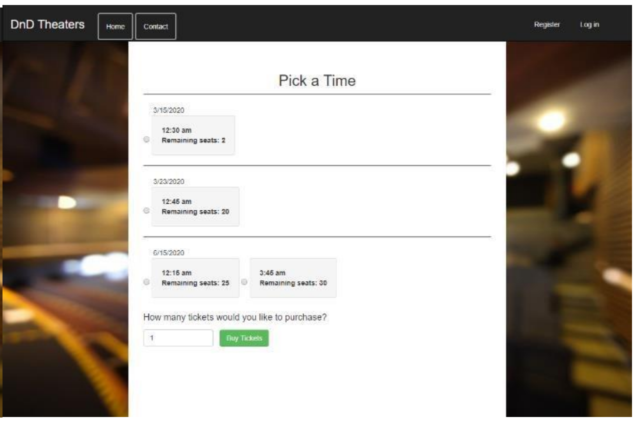
esrbRating: r

Overall Rating: 85%

Theaters Playing POPEYE:

Theater Name	Location
The Original DnD Theaters	6800 Zoo Dr Kansas City MO
First Expansion DnD Theaters	521 E Chestnut St Junction City KS

Selecting the number of tickets



Pick a Time

3/15/2020

12:30 am
Remaining seats: 2

3/23/2020

12:45 am
Remaining seats: 20

6/15/2020

12:15 am
Remaining seats: 25

3:45 am
Remaining seats: 30

How many tickets would you like to purchase?

1

These 2 pages are worth bringing up because this is where the magic happens. On the selecting a theater page, 2 things are accomplished. It shows you the information on the movie that you would want to know about as well as offer locations that are currently playing that movie.

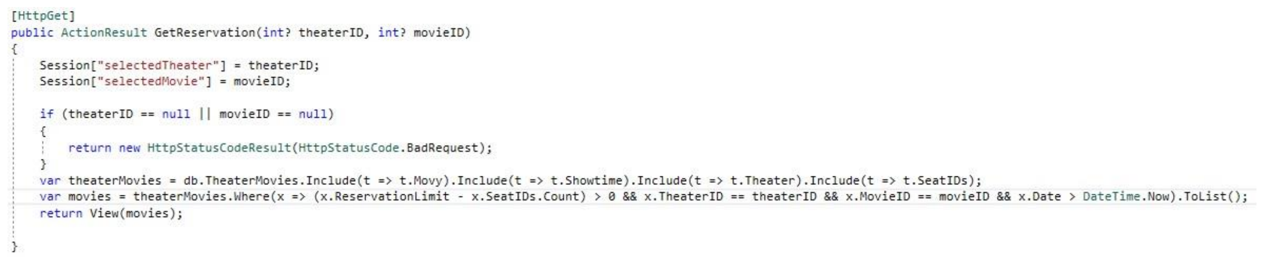
On the second page (selecting the number of tickets), shows only showtimes for that movie at that theater grouped by day and if:

1) The show hasn't already started
2) There is available seating at any particular showtime

On the second page there are several checks to make sure that the user is prompted if they did something wrong and to keep them from doing something that would cause an error. It keeps them from getting more tickets than what is allowed, and they can't get tickets where there is no showtime.

After the user finishes selecting the time and place, the program will take care of everything else in the background. It gives them a ticket and assigns those tickets to the corresponding showtime and the remaining seats get updated accordingly.

This code was made with the intention that it would have to deal with a worse case user scenario and it would still make the pages previously listed work:

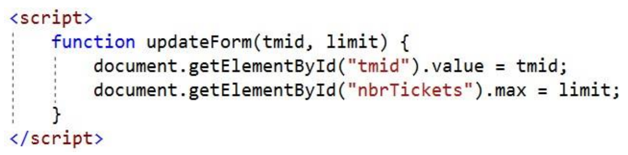


```
[HttpGet]
public ActionResult GetReservation(int? theaterID, int? movieID)
{
    Session["selectedTheater"] = theaterID;
    Session["selectedMovie"] = movieID;

    if (theaterID == null || movieID == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    var theaterMovies = db.TheaterMovies.Include(t => t.Movy).Include(t => t.Showtime).Include(t => t.Theater).Include(t => t.SeatIDs);
    var movies = theaterMovies.Where(x => (x.ReservationLimit - x.SeatIDs.Count) > 0 && x.TheaterID == theaterID && x.MovieID == movieID && x.Date > DateTime.Now).ToList();
    return View(movies);

}
```

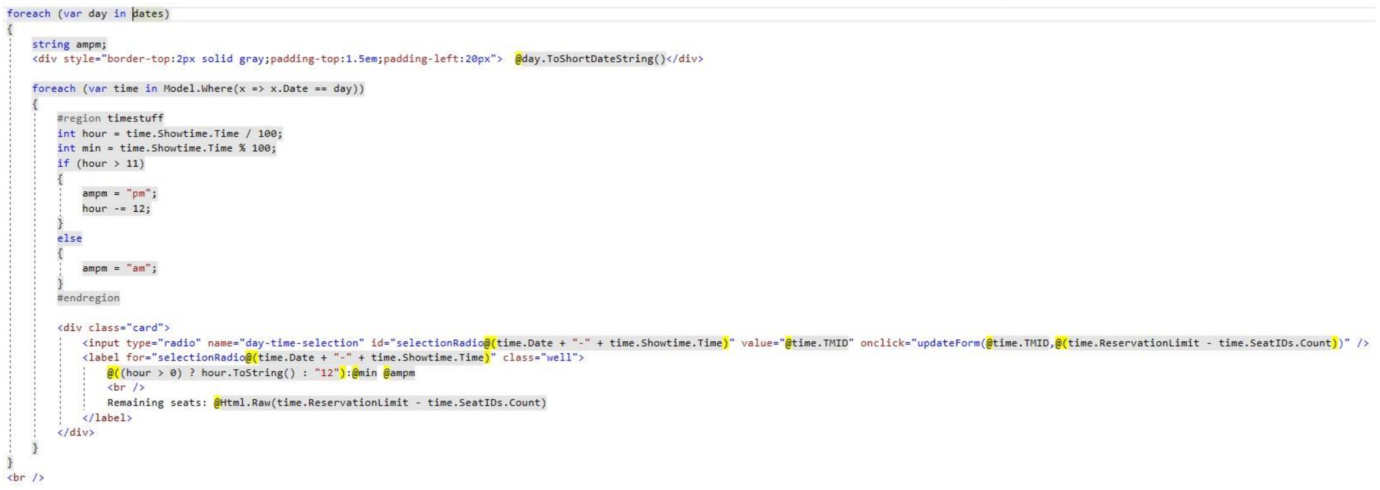
^ The code above remembers what the user was trying to see and where they were trying to see it. It also controls what showtimes can be seen by the user.



```
<script>
    function updateForm(tmid, limit) {
        document.getElementById("tmid").value = tmid;
        document.getElementById("nbrTickets").max = limit;
    }
</script>
```

^ The code above keeps the form updated with the maximum number of tickets the user can get at one time (small detail but the code is easy and is extremely important for form functionality).

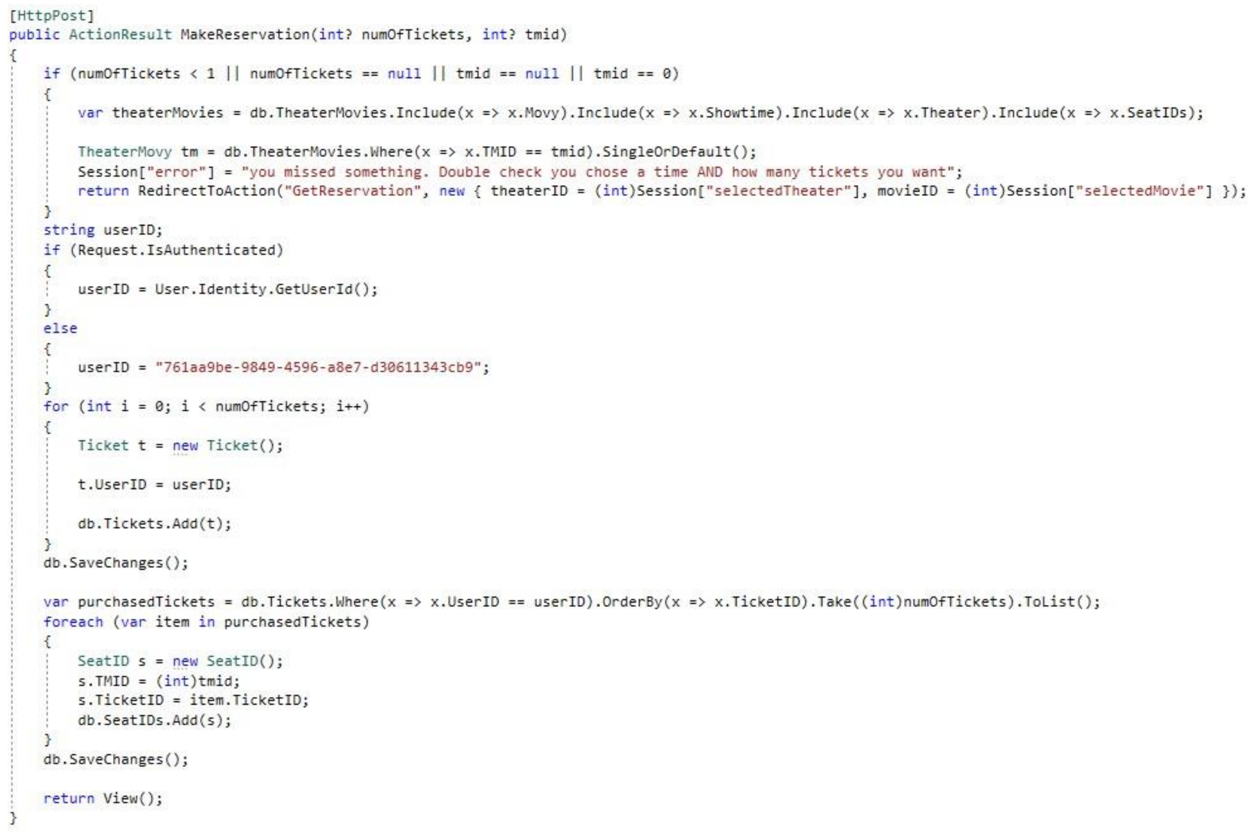
^ The snippet of code below is how the view builds out time selection for the user to choose from. This is fed to the previous script above to keep people from overbooking.



```
foreach (var day in dates)
{
    string ampm;
    <div style="border-top:2px solid gray;padding-top:1.5em;padding-left:20px">  @day.ToShortDateString()</div>

    foreach (var time in Model.Where(x => x.Date == day))
    {
        #region timestuff
        int hour = time.Showtime.Time / 100;
        int min = time.Showtime.Time % 100;
        if (hour > 11)
        {
            ampm = "pm";
            hour -= 12;
        }
        else
        {
            ampm = "am";
        }
        #endregion

        <div class="card">
            <input type="radio" name="day-time-selection" id="selectionRadio@(time.Date + "-" + time.Showtime.Time)" value="@time.TMID" onclick="updateForm(@time.TMID,@(time.ReservationLimit - time.SeatIDs.Count))" />
            <label for="selectionRadio@(time.Date + "-" + time.Showtime.Time)" class="well">
                @((hour > 0) ? hour.ToString() : "12"):@min @ampm
                <br />
                Remaining seats: @Html.Raw(time.ReservationLimit - time.SeatIDs.Count)
            </label>
        </div>
    }
}
<br />
```



```
[HttpPost]
public ActionResult MakeReservation(int? numOfTickets, int? tmid)
{
    if (numOfTickets < 1 || numOfTickets == null || tmid == null || tmid == 0)
    {
        var theaterMovies = db.TheaterMovies.Include(x => x.Movy).Include(x => x.Showtime).Include(x => x.Theater).Include(x => x.SeatIDs);

        TheaterMovy tm = db.TheaterMovies.Where(x => x.TMID == tmid).SingleOrDefault();
        Session["error"] = "you missed something. Double check you chose a time AND how many tickets you want";
        return RedirectToAction("GetReservation", new { theaterID = (int)Session["selectedTheater"], movieID = (int)Session["selectedMovie"] });
    }
    string userID;
    if (Request.IsAuthenticated)
    {
        userID = User.Identity.GetUserId();
    }
    else
    {
        userID = "761aa9be-9849-4596-a8e7-d30611343cb9";
    }
    for (int i = 0; i < numOfTickets; i++)
    {
        Ticket t = new Ticket();

        t.UserID = userID;

        db.Tickets.Add(t);
    }
    db.SaveChanges();

    var purchasedTickets = db.Tickets.Where(x => x.UserID == userID).OrderBy(x => x.TicketID).Take((int)numOfTickets).ToList();
    foreach (var item in purchasedTickets)
    {
        SeatID s = new SeatID();
        s.TMID = (int)tmid;
        s.TicketID = item.TicketID;
        db.SeatIDs.Add(s);
    }
    db.SaveChanges();

    return View();
}
```

^ The above code here is the handler that does all the work for the user after they made their selections. It first checks if everything is valid, then assigns the user the amount of tickets they wanted, and then assigns those tickets to the showtime.

8817 E 81 St
Kansas City, MO, 64138
816-590-0500
Snydercoding.com
Shawnmsnyder717@outlook.com

SHAWN SNYDER

OBJECTIVE	My goal is to get into a position where I can apply my skills to create new projects, and more importantly grow my skills and knowledge of technologies to do bigger and better projects.
TECHNOLGIES & SOFT SKILLS	Technologies: HTML5, CSS3, Java Script, C#, React, npm, jQuery, Json Soft Skills: Problem solver, Reliable, Easy to work with.
EXPERIENCE	**COOK HELPER: LITTLE SISTERS OF THE POOR** March 2018 – Present Help prepare and portion meals, deserts, special requests, and special diets for about 100 people and fill in when needed.
EDUCATION	**CENTRIQ TRAINING – 1740 W 92 ST KANSAS CITY MO** Full Stack Web Developer Track **KANONE INC – NO ADDRESS** Front End Development Course
REFERENCES	**TINA LOWE** Dietitian, Little Sister of the Poor Phone: 913-636-8565 **MARGE CLANAHAN** Kitchen Manager, Little Sister of the Poor Phone: 816-668-2470