***Informe Individual Ingeniería de Software***

***Henri Daniel Peña Dequero C-311 Movie++***

**1-Arquitectura.**

Como se explicó en el informe técnico nuestro proyecto se llevó a cabo con arquitectura de cebolla. Para ello se creó una aplicación web MVC con autenticación puesto que el proyecto requiere de esta, esta va a corresponder a nuestra capa de interfaz visual. Se creó además 3 librerías de clases que corresponden a las otras 3 capas, capa de datos donde está nuestra base de datos, capa de repositorio que une nuestra lógica de negocio con la base de datos y capa de servicio donde se encuentra la lógica de negocio de nuestra aplicación.

**2-Roles y autenticación**

Esta tarea la llevé a cabo junto con David donde diseñamos los cuatro roles que juegan en nuestra aplicación: admin, manager, ticket\_agent y basic\_user. En la base de datos vamos a sembrar un usuario admin con nombre de usuario **admin** y contraseña **Admin123!**, que va a ser el usuario encargado de asignar los roles a los demás, pues tendría todos los roles y así posee el acceso a toda la aplicación, sería como el jefe de todo el cine.

David puso para que cada usuario que se autentique tendrá por defecto el rol basic\_user, ya luego el admin se encargará de asignarle otro rol si lo lleva.

Aquí dejo el código de lo antes dicho para sembrarlo en la base de datos.

public static class ContextSeed

{

public static async Task SeedRolesAsync(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager)

{

//Seed Roles

await roleManager.CreateAsync(new IdentityRole(Roles.Admin.ToString()));

await roleManager.CreateAsync(new IdentityRole(Roles.Manager.ToString()));

await roleManager.CreateAsync(new IdentityRole(Roles.Ticket\_Agent.ToString()));

await roleManager.CreateAsync(new IdentityRole(Roles.Basic\_User.ToString()));

}

public static async Task SeedAdminAsync(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager)

{

//Seed Default User

var defaultUser = new ApplicationUser

{

UserName = "admin",

Email = "admin@gmail.com",

FirstName = "Admin First Name",

LastName = "Admin Last Name",

EmailConfirmed = true,

PhoneNumberConfirmed = true

};

if (userManager.Users.All(u => u.Id != defaultUser.Id))

{

var user = await userManager.FindByEmailAsync(defaultUser.Email);

if (user == null)

{

await userManager.CreateAsync(defaultUser, "Admin123!");

await userManager.AddToRoleAsync(defaultUser, Roles.Basic\_User.ToString());

await userManager.AddToRoleAsync(defaultUser, Roles.Manager.ToString());

await userManager.AddToRoleAsync(defaultUser, Roles.Ticket\_Agent.ToString());

await userManager.AddToRoleAsync(defaultUser, Roles.Admin.ToString());

}

}

}

}

**3-Búsqueda y filtros.**

Realicé también búsquedas y filtros en diferentes páginas de la aplicación para hacerle el trabajo más fácil a los usuarios, pudiéndose hacer búsquedas y filtros por actores, películas, género, ranking, entre otras.

-Aquí está el código de los filtros de Movie implementado en MovieService.

public IEnumerable<Movie> Filters(string \_title, string \_country, string \_kindOfMovie,

int \_duration, string \_actor, int \_ranking, List<Movie> \_movies)

{

if (!string.IsNullOrEmpty(\_title))

{

foreach (var item in \_title.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_movies = \_movies.Where(x => x.Title.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_country))

{

foreach (var item in \_country.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_movies = \_movies.Where(x => x.Country.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_kindOfMovie))

{

foreach (var item in \_kindOfMovie.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_movies = \_movies.Where(x => x.KindOfMovie.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_actor))

{

foreach (var item in \_actor.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_movies = \_movies.Where(x => x.Actors.ToLower().Contains(item.ToLower())).ToList();

}

}

if (\_duration != 0)

{

\_movies = \_movies.Where(x => x.Duration <= \_duration).ToList();

}

if (\_ranking != 0)

{

\_movies = \_movies.Where(x => x.Ranking >= \_ranking).ToList();

}

return \_movies;

}

Como vemos vamos buscando y la vez filtrando haciendo las correspondientes consultas en la base de datos.

-Ahora dejo los filtros de Horaries.

public IEnumerable<Horary> Filters(string \_title, string \_localName, DateTime \_minDate, DateTime \_maxDate,

int \_price, int \_priceInPoints, List<Horary> horaries)

{

if (!string.IsNullOrEmpty(\_title))

{

foreach (var item in \_title.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

horaries = horaries.Where(x => x.Movie.Title.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_localName))

{

foreach (var item in \_localName.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

horaries = horaries.Where(x => x.Movie\_Local.Local\_Name.ToLower().Contains(item.ToLower())).ToList();

}

}

if (\_price != 0)

{

horaries = horaries.Where(x => x.Price <= \_price).ToList();

}

if (\_priceInPoints != 0)

{

horaries = horaries.Where(x => x.PriceInPoints <= \_priceInPoints).ToList();

}

if(\_minDate != default(DateTime))

{

horaries = horaries.Where(x => x.Date >= \_minDate).ToList();

}

if(\_maxDate != default(DateTime))

{

horaries = horaries.Where(x => x.Date <= \_maxDate).ToList();

}

return horaries;

}

Como se pude ver todos siguen la misma metodología de ir filtrando a través de los if y haciendo las consultas en la base de datos.

-Y por último los filtros realizados para las Estadistics.

public IEnumerable<EstadisticViewModel> Filters(string \_title, string \_country, string \_kindOfMovie,int \_duration,

string \_actor, int \_ranking, List<EstadisticViewModel> \_estadistics)

{

if (!string.IsNullOrEmpty(\_title))

{

foreach (var item in \_title.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_estadistics = \_estadistics.Where(x => x.Title.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_country))

{

foreach (var item in \_country.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_estadistics = \_estadistics.Where(x => x.Country.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_kindOfMovie))

{

foreach (var item in \_kindOfMovie.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_estadistics = \_estadistics.Where(x => x.KindOfMovie.ToLower().Contains(item.ToLower())).ToList();

}

}

if (!string.IsNullOrEmpty(\_actor))

{

foreach (var item in \_actor.Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries))

{

\_estadistics = \_estadistics.Where(x => x.Actors.ToLower().Contains(item.ToLower())).ToList();

}

}

if (\_duration != 0)

{

\_estadistics = \_estadistics.Where(x => x.Duration <= \_duration).ToList();

}

if (\_ranking != 0)

{

\_estadistics = \_estadistics.Where(x => x.Ranking >= \_ranking).ToList();

}

return \_estadistics;

}

**4-Socios del cine.**

También fui el encargado de crear los socios del club, los cuales pueden ser creados por el taquillero cuando se les deja un formulario en taquilla o por los propios usuarios de la página y así gozar de los beneficios de estos.

Aquí dejo como quedó el controlador final del mismo.

public class MovieMemberController : Controller

{

private readonly UserManager<ApplicationUser> \_UserManager;

private IUserService \_UserService;

public MovieMemberController(IUserService UserService, UserManager<ApplicationUser> userManager)

{

\_UserManager = userManager;

\_UserService = UserService;

}

public IActionResult Index()

{

if (User.IsInRole(Roles.Roles.Ticket\_Agent.ToString()))

{

ViewData["IsTicketAgent"] = true;

}

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Index(string FirstName, string LastName, long IdentificationNumber, string Email)

{

if (User.IsInRole(Roles.Roles.Ticket\_Agent.ToString()))

{

ViewData["IsTicketAgent"] = true;

}

if (FirstName == null || LastName == null)

{

ViewData["Error"] = "Write a valid FirstName/LastName";

return View();

}

if (IdentificationNumber.ToString().Length != 11)

{

ViewData["Error"] = "Id must have 11 numbers";

return View();

}

if (Email == null && User.IsInRole(Roles.Roles.Ticket\_Agent.ToString()))

{

ViewData["Error"] = "Insert a valid Email";

return View();

}

else if (Email == null)

{

var user = \_UserService.GetUser(\_UserManager.GetUserId(User));

if (user.Code != null)

{

TempData["msg"] = "You already are a member of this movie";

return RedirectToAction("Index", "Home");

}

user.Code = IdentificationNumber;

user.Puntuation = 0;

\_UserService.UpdateUser(user);

TempData["msg"] = "Success: You are a member of MoviePlus";

return RedirectToAction("Index", "Home");

}

var userByEmail = \_UserService.GetUserByEmail(Email);

if (userByEmail != null)

{

if (userByEmail.Code != null)

{

TempData["msg"] = userByEmail.UserName + " is already are a member of this movie";

return RedirectToAction("Index", "Home");

}

userByEmail.Code = IdentificationNumber;

userByEmail.Puntuation = 0;

\_UserService.UpdateUser(userByEmail);

TempData["msg"] = "Success: " + userByEmail.UserName + " are a member of MoviePlus";

return RedirectToAction("Index", "Home");

}

var newUser = new ApplicationUser

{

UserName = Email.Split('@')[0],

Email = Email,

FirstName = "Add First Name",

LastName = "Add Last Name",

EmailConfirmed = true,

PhoneNumberConfirmed = true,

Code = IdentificationNumber,

Puntuation = 0

};

await \_UserService.CreateUser(newUser, \_UserManager);

TempData["msg"] = "New user was added to the system with Email: " + Email +

" and default password: NewUser123!. Password should be change as soon as posible.";

return RedirectToAction("Index", "Home");

}

5-Realicé el paginado de la aplicación haciendo un servicio para este, donde listo 10 objetos por páginas, ya sean películas, horarios o las compras, pues el paginado fue hecho genérico.

public class PagerService<T> : IPagerService<T> where T: class

{

private static int \_RegistersByPage = 10;

private static GenericPager<T> \_pager;

public GenericPager<T> GetPager(List<T> entities, int page)

{

int \_TotalRegisters = entities.Count;

entities = entities.Skip((page - 1) \* \_RegistersByPage)

.Take(\_RegistersByPage)

.ToList();

var \_TotalPages = (int)Math.Ceiling((double)\_TotalRegisters / \_RegistersByPage);

\_pager = new GenericPager<T>()

{

RegistersByPage = \_RegistersByPage,

TotalRegisters = \_TotalRegisters,

TotalPages = \_TotalPages,

ActualPage = page,

Result = entities

};

return \_pager;

}

}

En el informe técnico del proyecto me correspondió desarrollar la arquitectura y enfoque metodológico.