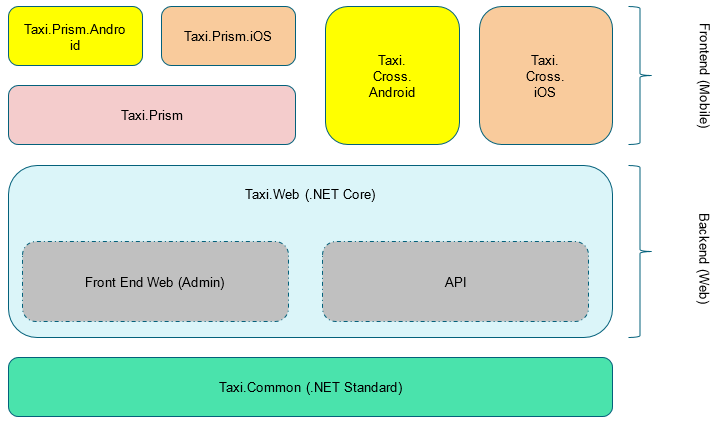
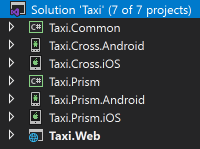
\_La

# Funcionalidad de la Aplicación

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  | |  | |  | |  | |  | |
|  |  |  | |  | |  | |  | |  | |  | |
|  | **Functionality** | | **Web** | | | | | | **App** | | | | **Observaciones** | |
|  | **Admin** | | **Driver** | | **User** | | **Driver** | | **User** | |
|  | Login | | X | | X | | X | | X | | X | | Todos se pueden loguear por Web y por App | |
|  | Register | |  | | X | | X | | X | | X | | Diver y User pueden autoregistrarse en Web y en App | |
|  | Modify profile | | X | | X | | X | | X | | X | | Todos pueden modificar sus datos | |
|  | Recover password | | X | | X | | X | | X | | X | | Todos pueden modificar su Password | |
|  | Admin taxis | | X | | X | | X | | X | | X | | Todos pueden crear Taxis a través de la patente | |
|  | Admin user groups | |  | |  | | X | |  | | X | | Sólo un User puede administrar Grupos | |
|  | Add trips | |  | |  | |  | |  | | X | | Sólo un User puede administrar Viajes desde la App | |
|  | Check history | | X | | X | | X | | X | | X | | Todos pueden ver el Historial de Viajes y las calificaciones | |
|  | View trips for groups | |  | |  | | X | |  | | X | | Sólo un User puede ver los Viajes de un Grupo | |

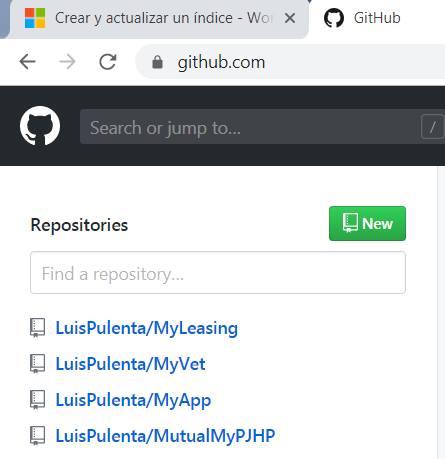
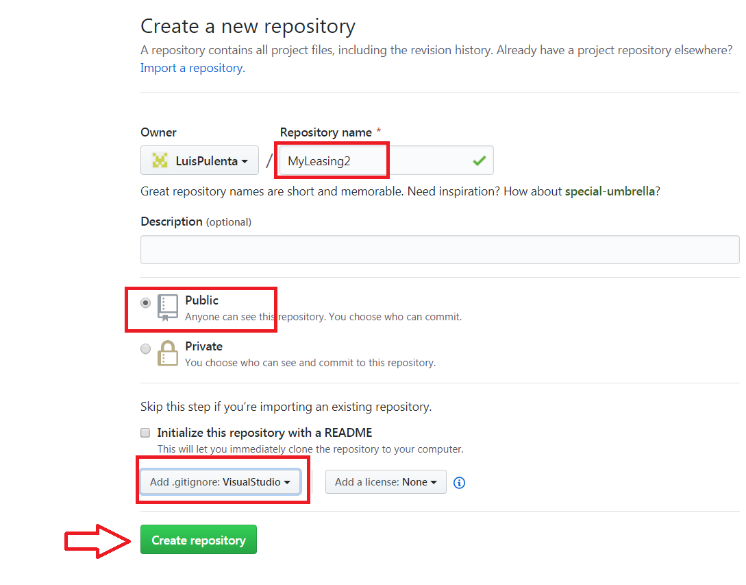
# Arquitectura de la Solución

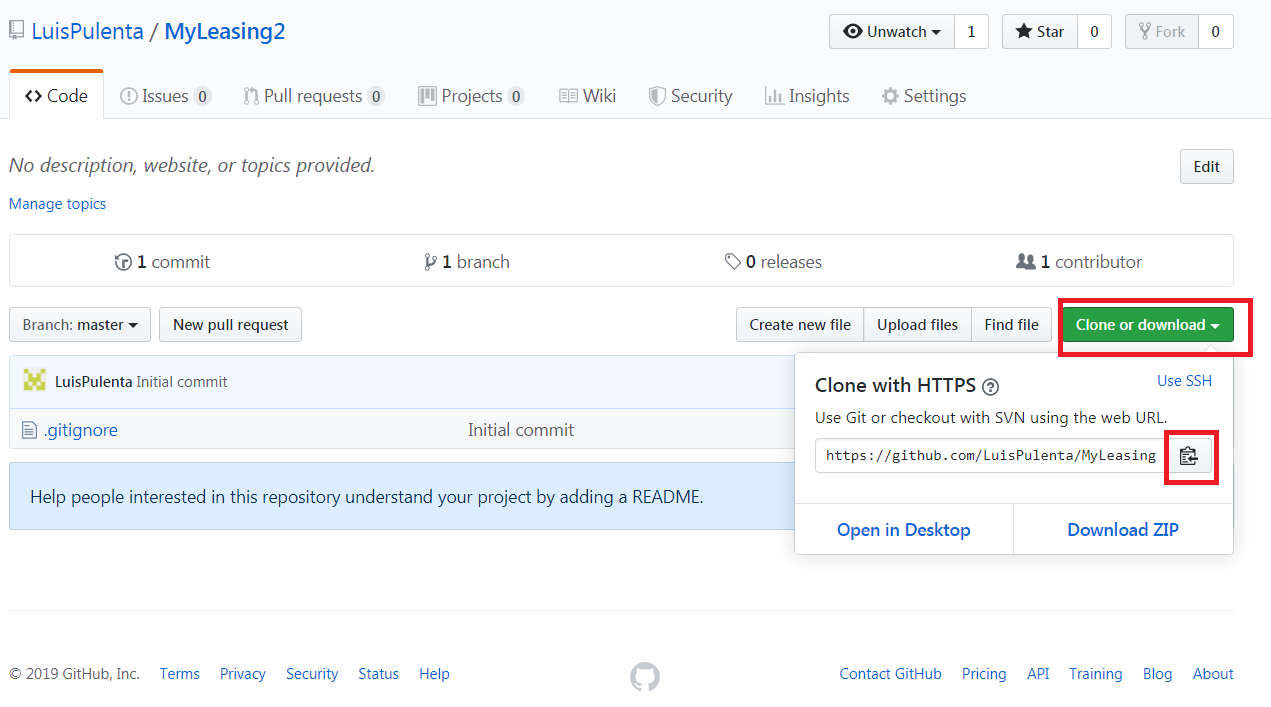
# Repositorio en GitHub

Empezamos haciendo el Repositorio en GitHub

<https://github.com/> **Usuario:** LuisPulenta **Contraseña:** Talleres2306

Taxi



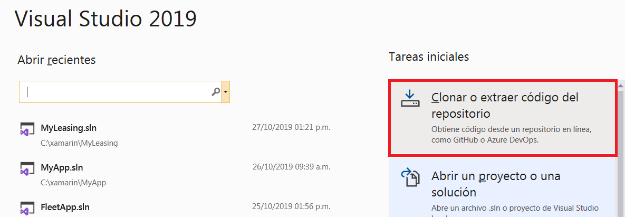
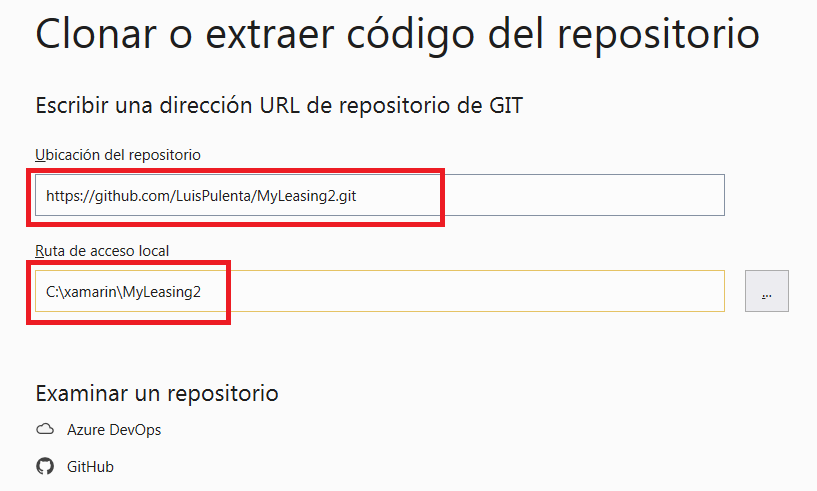
Taxi

Taxi

Copiar

# Crear Solución y Proyectos Common y Web

Abrimos Visual Studio y vamos a la opción Clonar

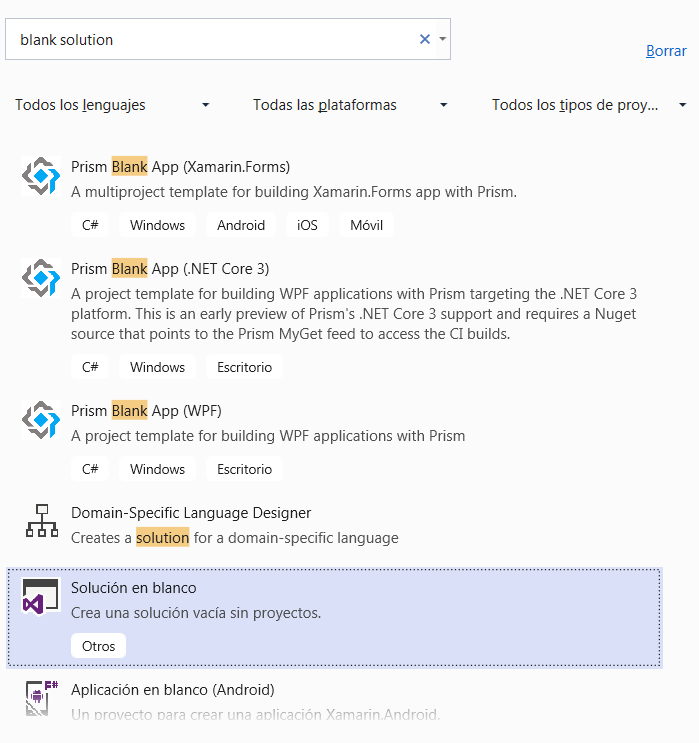
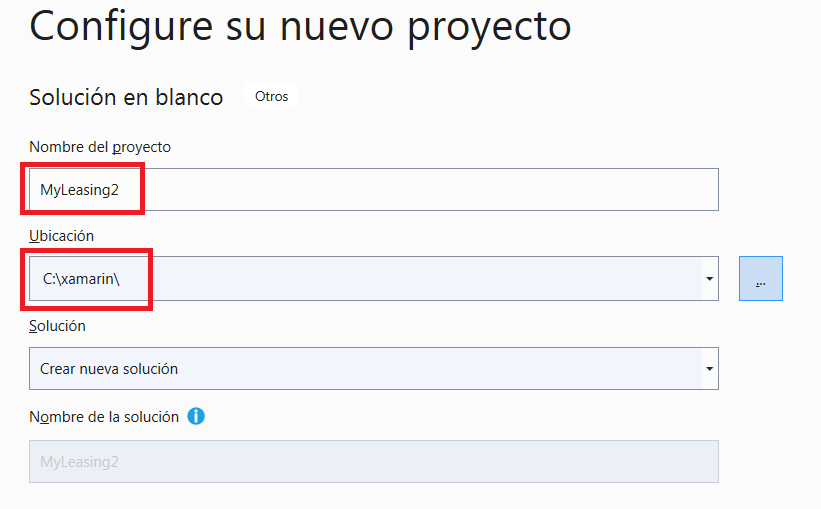
Taxi

Taxi

Pegar

Que sea el mismo nombre del Repositorio

Luego hacemos **Archivo-Nuevo-Proyecto** y buscamos plantillas **blank solution** y elegimos Solución en blanco

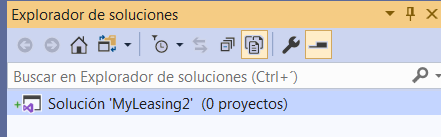
 

Taxi

Que sea el mismo nombre del Repositorio

OJO!!! Debe ser la Carpeta CONTENEDORA

Ahora hacemos clic derecho en la Solución:



Taxi

Y hacemos Agregar Nuevo Proyecto.

Ponemos **.net standard** en la plantilla y elegimos **Biblioteca de Clases c#**



Y la llamamos **Taxi.Common**

(La Clase Class1 que se crea la borramos)

Volvemos a hacer clic derecho en la Solución y ahora ponemos **.Net Core** en la plantilla y elegimos **Aplicación Web.Net Core**



Como nombre le ponemos **Taxi.Web**



# Crear Proyectos Prism y Cross

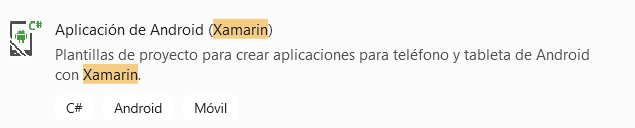
Creamos el Proyecto **Taxi.Prism** tipo **Prism Blank App (Xamarin Forms)**

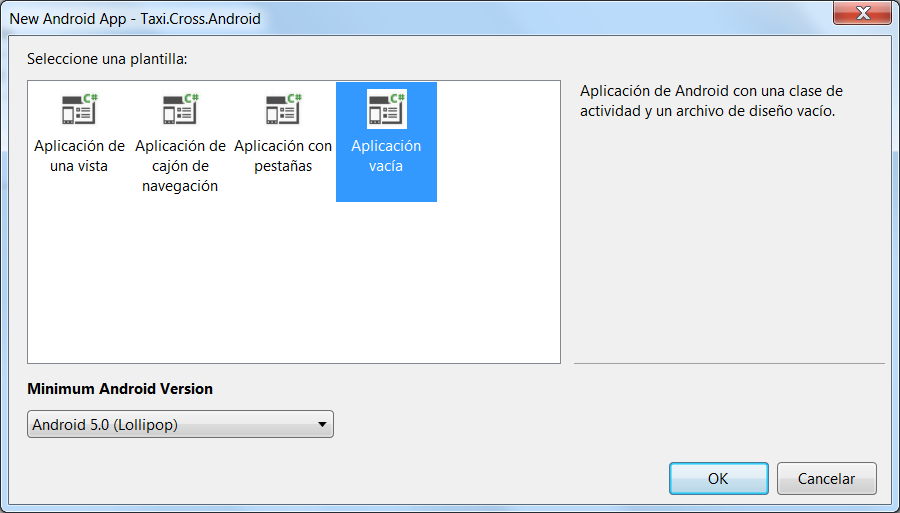
****

Elegimos **Container 🡪 DryIoc**

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Creamos el Proyecto **Taxi.Cross.Android** tipo **Aplicación de Android (Xamarin)**





En el MainActivity que aparece hay un error

Hay que borrar:

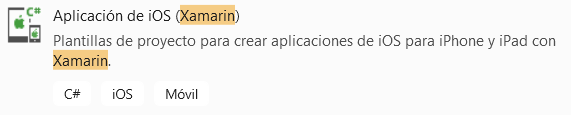
public override void OnRequestPermissionsResult(int requestCode, string[] permissions, [GeneratedEnum] Android.Content.PM.Permission[] grantResults)

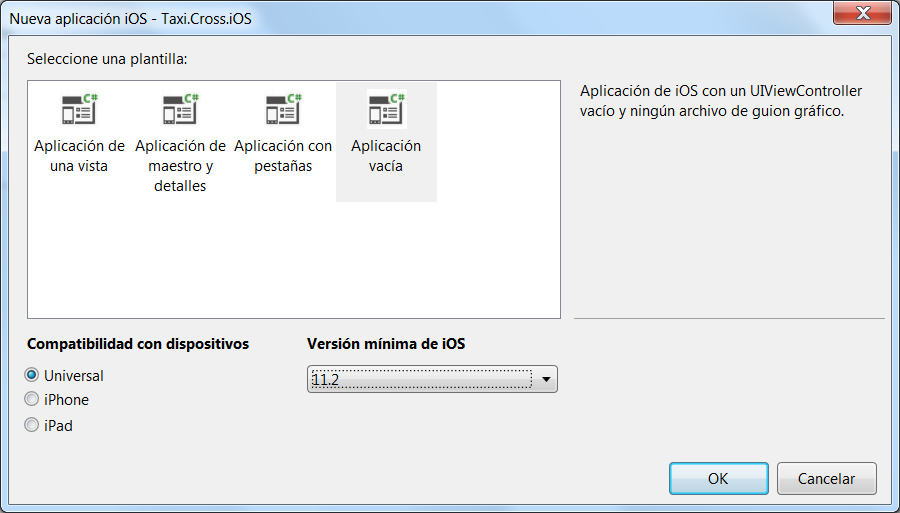
Luego hacer Control Punto y elegir

Using Android.Content.PM

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Creamos el Proyecto **Taxi.Cross.iOS** tipo **Aplicación de iOS (Xamarin)**





# Crear Base de Datos

Las Clases que luego se convertirán en Tablas en una Base de Datos las llamamos Entities. Los Nombres deben ser en inglés en Singular, porque luego al crearse como Tablas se pluralizan.

En el Proyecto Web creamos la carpeta **Data**, y dentro la carpeta **Entities**

## Clase TaxiEntity

Dentro de Entities creamos la Clase **TaxiEntity:**

|  |  |
| --- | --- |
| **Clase TaxiEntity** | **Comentarios** |
| using System.ComponentModel.DataAnnotations;  namespace Taxi.Web.Data.Entities  {  public class TaxiEntity  {  public int Id { get; set; }  [StringLength(6, MinimumLength = 6, ErrorMessage = "The {0} field must have {1} characters.")]  [Required(ErrorMessage = "The field {0} is mandatory.")]  public string Plaque { get; set; }  }  } | Todas las Tablas deben tener un campo Id de tipo Int autoincrementable  **Data Anotations**  **Como queremos que se muestre el campo:**  [Display(Name = "Documento")]  **Tamaño máximo:**  [MaxLength(20, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]  **Para que el campo sea obligatorio:**  [Required(ErrorMessage = "El campo {0} es requerido.")]  **Propiedades de solo lectura (no se mapean en BD)**  Lleva este indicador 🡪 =>  public string FullName => $"{FirstName} {LastName}";  **Concatenar dos campos**  Se hace con el signo $ y cada campo va entre llaves.  También se llama interpolación  $"{FirstName} {LastName}" |

## Clase DataContext

Dentro de **Data** creamos la Clase **DataContext**

Esta Clase se usa para poder mapear en Base de Datos

|  |  |
| --- | --- |
| **Clase DataContext** | **Comentarios** |
| using Microsoft.EntityFrameworkCore;  using Taxi.Web.Data.Entities;  namespace Taxi.Web.Data  {  public class DataContext : DbContext  {  public DataContext(DbContextOptions<DataContext> options) : base(options)  {  }  public DbSet<TaxiEntity> Taxis { get; set; }  }  } | Aquí van declaradas todas las Entities que se convertirán en Tablas en una BD |

## String de conexión

Creamos el string de conexión dentro del archivo **appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": { "DefaultConnection": "Server=keypress.serveftp.net;Database=LuisTaxi;User Id=sa;password=sentey14$;Trusted\_Connection=False;MultipleActiveResultSets=true" }

}

Agregamos estas líneas en el archivo **Startup.cs** (con esto le estamos diciendo que usaremos base de datos)

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

## Crear Base de Datos

Creamos la Base de Datos poniendo en la **Consola del Administrador de Paquete** estos comandos.

Asegurarse de que:

- se está dentro del Proyecto Web

- que el Proyecto Web tenga la referencia del Proyecto Common

update-database

ad d-migration InitialDb

update-database

La otra forma es abrir una ventana de DOS, ubicarse en la carpeta del Proyecto Web y poner estos comandos:

dotnet ef database update

dotnet ef migrations add InitialDb

dotnet ef database update

# Controlador Taxi

Creamos el controlador **TaxisController.**

En la Vista **\_Layout.cshtml** agregamos la fila:

<ul class="nav navbar-nav">

<li><a **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Home</a></li>

<li><a **asp-area**="" **asp-controller**="Home" **asp-action**="About">About</a></li>

<li><a **asp-area**="" **asp-controller**="Home" **asp-action**="Contact">Contact</a></li>

<li><a **asp-area**="" **asp-controller**="Taxis" **asp-action**="Index">Taxis</a></li>

</ul>

Esto es:

asp-controller="Taxis" 🡪 Nombre del Controlador

asp-action="Index"> 🡪 Acción

Taxis 🡪 Como quiero que se vea en el menú

En la vista **Index** de **Taxis** cambiamos el Título

@model IEnumerable<Taxi.Web.Data.Entities.TaxiEntity>

@{

ViewData["Title"] = "Index";

}

<h2>Taxis</h2>

<p>

<a asp-action="Create">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Plaque)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Plaque)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

<a asp-action="Details" asp-route-id="@item.Id">Details</a> |

<a asp-action="Delete" asp-route-id="@item.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

Ya podemos ejecutar el Proyecto y cargar nuevos Taxis, editarlos y borrrarlos.

Agregamos a la Clase **TaxiEntity**

[RegularExpression(@"^([A-Za-z]{3}\d{3})$", ErrorMessage = "El campo {0} debe iniciar con 3 caracteres y terminar con 3 números.")]

Modificamos la vista **Index** del controlador **TaxisController**

@model IEnumerable<Taxi.Web.Data.Entities.TaxiEntity>

@{

ViewData["Title"] = "Index";

}

<h2>Taxis</h2>

<p>

<a **asp-action**="Create" class="btn btn-primary">Crear Nuevo</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Plaque)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Plaque)

</td>

<td>

<a **asp-action**="Edit" **asp-route-id**="@item.Id" class="btn btn-warning">Editar</a>

<a **asp-action**="Details" **asp-route-id**="@item.Id" class="btn btn-info">Detalles</a>

<a **asp-action**="Delete" **asp-route-id**="@item.Id" class="btn btn-danger">Borrar</a>

</td>

</tr>

}

</tbody>

</table>

Modificamos el **método** **Create** (Get y Post) de **TaxisController**

// GET: Taxis/Create

[HttpGet]

public IActionResult Create()

{

return View();

}

// POST: Taxis/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(TaxiEntity taxiEntity)

{

if (ModelState.IsValid)

{

taxiEntity.Plaque = taxiEntity.Plaque.ToUpper();

\_context.Add(taxiEntity);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(taxiEntity);

}

Modificamos la **vista** **Create** de **TaxisController**

@model Taxi.Web.Data.Entities.TaxiEntity

@{

ViewData["Title"] = "Crear";

}

<h2>Crear</h2>

<h4>Taxi</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="Create">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Plaque" class="control-label"></label>

<input **asp-for**="Plaque" class="form-control" />

<span **asp-validation-for**="Plaque" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Crear" class="btn btn-primary" />

<a **asp-action**="Index" class="btn btn-success" >Regresar</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

En **Startup.cs** del **Proyecto Web** tenemos la acción por default del inicio:

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

Modificamos el **método** **Details** de **TaxisController**

// GET: Taxis/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

TaxiEntity taxiEntity = await \_context.Taxis

.FirstOrDefaultAsync(m => m.Id == id);

if (taxiEntity == null)

{

return NotFound();

}

return View(taxiEntity);

}

Modificamos la **vista** **Details** de **TaxisController**

@model Taxi.Web.Data.Entities.TaxiEntity

@{

ViewData["Title"] = "Details";

}

<h2>Detalles</h2>

<div>

<h4>Taxi</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Plaque)

</dt>

<dd>

@Html.DisplayFor(model => model.Plaque)

</dd>

</dl>

</div>

<div>

<a **asp-action**="Edit" **asp-route-id**="@Model.Id" class="btn btn-warning">Editar</a>

<a **asp-action**="Index" class="btn btn-success">Regresar</a>

</div>

Modificamos el **método** **Edit** (Get y Post) de **TaxisController**

// GET: Taxis/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

TaxiEntity taxiEntity = await \_context.Taxis.FindAsync(id);

if (taxiEntity == null)

{

return NotFound();

}

return View(taxiEntity);

}

// POST: Taxis/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id,TaxiEntity taxiEntity)

{

if (id != taxiEntity.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

taxiEntity.Plaque = taxiEntity.Plaque.ToUpper();

\_context.Update(taxiEntity);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(taxiEntity);

}

Modificamos la **vista** **Edit** de **TaxisController**

@model Taxi.Web.Data.Entities.TaxiEntity

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Taxi</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="Edit">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **type**="hidden" **asp-for**="Id" />

<div class="form-group">

<label **asp-for**="Plaque" class="control-label"></label>

<input **asp-for**="Plaque" class="form-control" />

<span **asp-validation-for**="Plaque" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Grabar" class="btn btn-primary" />

<a **asp-action**="Index" class="btn btn-success">Regresar</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Modificamos el **método** **Delete** de **TaxisController**

// GET: Taxis/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

TaxiEntity taxiEntity = await \_context.Taxis

.FirstOrDefaultAsync(m => m.Id == id);

if (taxiEntity == null)

{

return NotFound();

}

\_context.Taxis.Remove(taxiEntity);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

Borramos la **vista** **Delete** de **TaxisController**

# Completar Base de Datos

Completamos la BD con las otras Tablas:

Taxi

Trip

Trip Detail

1

\*

\*

1

## Clase **TripEntity**

En **Entities** creamos la Clase **TripEntity**

|  |  |
| --- | --- |
| **Clase TripEntity** | **Comentarios** |
| using System;  using System.ComponentModel.DataAnnotations;  namespace Taxi.Web.Data.Entities  {  public class TripEntity  {  public int Id { get; set; }  [DataType(DataType.DateTime)]  [Display(Name = "Fecha inicio")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime StartDate { get; set; }  [DataType(DataType.DateTime)]  [Display(Name = "Fecha inicio")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime StartDateLocal => StartDate.ToLocalTime();  [DataType(DataType.DateTime)]  [Display(Name = "Fecha Fin")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime? EndDate { get; set; }  [DataType(DataType.DateTime)]  [Display(Name = "Fecha Fin")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime? EndDateLocal => EndDate?.ToLocalTime();  [Display(Name = "Origen")]  [MaxLength(100, ErrorMessage = "El campo {0} debe tener {1} caracteres.")]  public string Source { get; set; }  [Display(Name = "Destino")]  [MaxLength(100, ErrorMessage = "El campo {0} debe tener {1} caracteres.")]  public string Target { get; set; }  [Display(Name = "Calificación")]  public float Qualification { get; set; }  [Display(Name = "Latitud origen")]  public double SourceLatitude { get; set; }  [Display(Name = "Longitud origen")]  public double SourceLongitude { get; set; }  [Display(Name = "Latitud Destino")]  public double TargetLatitude { get; set; }  [Display(Name = "Longitud Destino")]  public double TargetLongitude { get; set; }  [Display(Name = "Comentarios")]  public string Remarks { get; set; }  public TaxiEntity Taxi { get; set; }  }  } |  |

En **TaxiEntity** agregamos:

public ICollection<TripEntity> Trips { get; set; }

## Clase **TripDetailEntity**

En **Entities** creamos la Clase **TripDetailEntity**

|  |  |
| --- | --- |
| **Clase TripDetailEntity** | **Comentarios** |
| using System;  using System.ComponentModel.DataAnnotations;  namespace Taxi.Web.Data.Entities  {  public class TripDetailEntity  {  public int Id { get; set; }  [DataType(DataType.DateTime)]  [Display(Name = "Fecha Inicio")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime Date { get; set; }  [DataType(DataType.DateTime)]  [Display(Name = "Fecha Inicio")]  [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}", ApplyFormatInEditMode = false)]  public DateTime DateLocal => Date.ToLocalTime();  [Display(Name = "Latitud")]  public double Latitude { get; set; }  [Display(Name = "Longitud")]  public double Longitude { get; set; }  public TripEntity Trip { get; set; }  }  } |  |

En **TripEntity** agregamos:

public ICollection<TripDetailEntity> TripDetails { get; set; }

**Truco del TODO**

Cuando queremos dejar registrado una parte del código que hay que revisar más adelante, le ponemos un comentario que empiece con

//TODO

Y luego vamos al Menú Ver – Lista de Tareas y nos muestra todos los comentarios que empiezan con TODO

## Registrar nuevas Entities en DataContext

En **DataContext** agregamos todas las Entities/Tablas nuevas que se mapearán en la BD:

using Microsoft.EntityFrameworkCore;

using Taxi.Web.Data.Entities;

namespace Taxi.Web.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<TaxiEntity> Taxis { get; set; }

public DbSet<TripEntity> Trips { get; set; }

public DbSet<TripDetailEntity> TripDetails { get; set; }

}

}

Vamos a la Consola del Adminisitrador de Paquetes y ejecutamos

add-migration TripAndTripDetails

update-database

# Seeder

Dentro de Data creamos la Clase **SeedDB**

|  |  |
| --- | --- |
| **Seeder** | **Comentarios** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Taxi.Web.Data.Entities;  namespace Taxi.Web.Data  {  public class SeedDb  {  private readonly DataContext \_dataContext;  public SeedDb(DataContext dataContext)  {  \_dataContext = dataContext;  }  public async Task SeedAsync()  {  await \_dataContext.Database.EnsureCreatedAsync();  await CheckTaxisAsync();  }  private async Task CheckTaxisAsync()  {  if (!\_dataContext.Taxis.Any())  {  \_dataContext.Taxis.Add(new TaxiEntity  {  Plaque = "PJM791",  Trips = new List<TripEntity>  {  new TripEntity  {  StartDate = DateTime.UtcNow,  EndDate = DateTime.UtcNow.AddMinutes(30),  Qualification = 4.5f,  Source = "Barrio Rosedal",  Target = "Centro",  Remarks = "Muy buen servicio"  },  new TripEntity  {  StartDate = DateTime.UtcNow,  EndDate = DateTime.UtcNow.AddMinutes(30),  Qualification = 4.8f,  Source = "Centro",  Target = "Barrio Rosedal",  Remarks = "Conductor muy amable"  }  }  });  \_dataContext.Taxis.Add(new TaxiEntity  {  Plaque = "IJX537",  Trips = new List<TripEntity>  {  new TripEntity  {  StartDate = DateTime.UtcNow,  EndDate = DateTime.UtcNow.AddMinutes(30),  Qualification = 4.5f,  Source = "Barrio Rosedal",  Target = "Barrio General Paz",  Remarks = "Muy buen servicio"  },  new TripEntity  {  StartDate = DateTime.UtcNow,  EndDate = DateTime.UtcNow.AddMinutes(30),  Qualification = 4.8f,  Source = "Barrio General Paz",  Target = "Barrio Rosedal",  Remarks = "Conductor muy amable"  }  }  });  await \_dataContext.SaveChangesAsync();  }  }  }  } |  |

ObraObraModificamos el **program.cs** por estas líneas:

|  |  |
| --- | --- |
| using Microsoft.AspNetCore;  using Microsoft.AspNetCore.Hosting;  using Microsoft.Extensions.DependencyInjection;  using Taxi.Web.Data;  namespace Taxi.Web  {  public class Program  {  public static void Main(string[] args)  {  IWebHost host = CreateWebHostBuilder(args).Build();  RunSeeding(host);  host.Run();  }  private static void RunSeeding(IWebHost host)  {  IServiceScopeFactory scopeFactory = host.Services.GetService<IServiceScopeFactory>();  using (IServiceScope scope = scopeFactory.CreateScope())  {  SeedDb seeder = scope.ServiceProvider.GetService<SeedDb>();  seeder.SeedAsync().Wait();  }  }  public static IWebHostBuilder CreateWebHostBuilder(string[] args)  {  return WebHost.CreateDefaultBuilder(args).UseStartup<Startup>();  }  }  } |  |

En el archivo **Startup.cs** agregamos esta línea

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddTransient<SeedDb>();

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

Borramos la BD con

drop-database

en la **Consola del Administrador de Paquetes.**

Ejecutamos el Proyecto Web y verificamos si se cargaron los datos en las Tablas

# Agregar clave única al campo Patente

Agregamos en la Clase **DataContext**

using Microsoft.EntityFrameworkCore;

using Taxi.Web.Data.Entities;

namespace Taxi.Web.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<TaxiEntity> Taxis { get; set; }

public DbSet<TripEntity> Trips { get; set; }

public DbSet<TripDetailEntity> TripDetails { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<TaxiEntity>()

.HasIndex(t => t.Plaque)

.IsUnique();

}

}

}

Luego vamos a la Consola del Adminisitrador de Paquetes y ejecutamos

add-migration AddIndexInPlaqueOnTaxi

drop-database

update-database

Modificamos los controladores **Create Post** y **Edit** **Post** de **TaxisController**

// POST: Taxis/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(TaxiEntity taxiEntity)

{

if (ModelState.IsValid)

{

taxiEntity.Plaque = taxiEntity.Plaque.ToUpper();

\_context.Add(taxiEntity);

try

{

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

catch (Exception ex)

{

if (ex.InnerException.Message.Contains("duplicate"))

{

ModelState.AddModelError(string.Empty, "Esta Patente ya existe");

}

else

{

ModelState.AddModelError(string.Empty, ex.InnerException.Message);

}

}

}

return View(taxiEntity);

}

// POST: Taxis/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id,TaxiEntity taxiEntity)

{

if (id != taxiEntity.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

taxiEntity.Plaque = taxiEntity.Plaque.ToUpper();

\_context.Update(taxiEntity);

try

{

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

catch (Exception ex)

{

if (ex.InnerException.Message.Contains("duplicate"))

{

ModelState.AddModelError(string.Empty, "Esta Patente ya existe");

}

else

{

ModelState.AddModelError(string.Empty, ex.InnerException.Message);

}

}

}

return View(taxiEntity);

}

# User Identities

En el Proyecto Taxi.Common creamos una carpeta llamada **Enums**

Dentro creamos la enum **UserType**

namespace Taxi.Common.Enums

{

public enum UserType

{

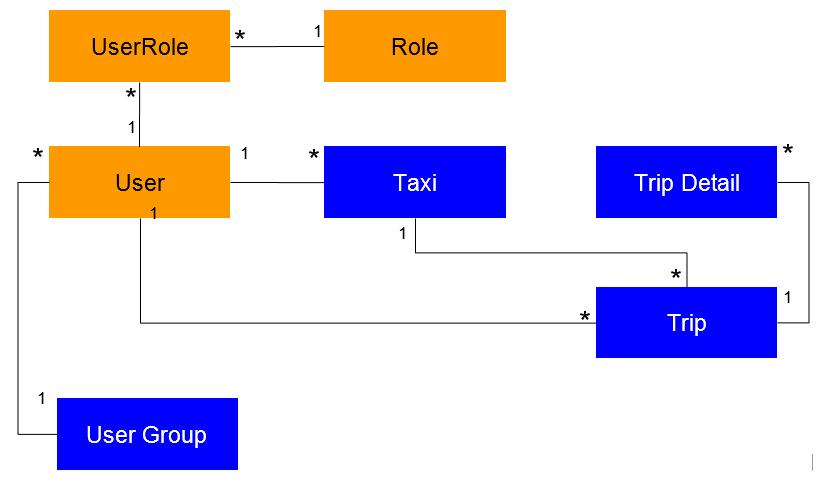
Admin,

Driver,

User

}

}



Agregamos las Tablas

* UserRole
* Role
* User

que son del Entity Framework

## Clase UserEntity

En **Entity** creamos la Clase **UserEntity**

|  |  |
| --- | --- |
| **Clase UserEntity** | **Comentarios** |
| using Microsoft.AspNetCore.Identity;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using Taxi.Common.Enums;  namespace Taxi.Web.Data.Entities  {  public class UserEntity : IdentityUser  {  [Display(Name = "Documento")]  [MaxLength(20, ErrorMessage = "El campo {0} no puede tener mas de {1} caracteres.")]  [Required(ErrorMessage = "El campo {0} es obligatorio.")]  public string Document { get; set; }  [Display(Name = "Nombre")]  [MaxLength(50, ErrorMessage = "El campo {0} no puede tener mas de {1} caracteres.")]  [Required(ErrorMessage = "El campo {0} es obligatorio.")]  public string FirstName { get; set; }  [Display(Name = "Apellido")]  [MaxLength(50, ErrorMessage = "El campo {0} no puede tener mas de {1} caracteres.")]  [Required(ErrorMessage = "El campo {0} es obligatorio.")]  public string LastName { get; set; }  [Display(Name = "Domicilio")]  [MaxLength(100, ErrorMessage = "El campo {0} no puede tener mas de {1} caracteres.")]  public string Address { get; set; }  [Display(Name = "Foto")]  public string PicturePath { get; set; }  [Display(Name = "Tipo Usuario")]  public UserType UserType { get; set; }  [Display(Name = "Nombre Completo")]  public string FullName => $"{FirstName} {LastName}";  [Display(Name = "Nombre Completo y Documento")]  public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";  public ICollection<TaxiEntity> Taxis { get; set; }  public ICollection<TripEntity> Trips { get; set; }  }  } |  |

Lo que hacemos acá es complementar la Tabla User que viene con el Framework agregándole los campos que yo quiero como ser Nombre, Apellido, Dirección, etc.

Agregamos a la Clase **TaxiEntity**

public UserEntity User { get; set; }

Agregamos a la Clase **TripEntity**

public UserEntity User { get; set; }

En **Entity** creamos la Clase **UserGroupEntity**

using System.Collections.Generic;

namespace Taxi.Web.Data.Entities

{

public class UserGroupEntity

{

public int Id { get; set; }

public UserEntity User { get; set; }

public ICollection<UserGroupDetailEntity> Users { get; set; }

}

}

## Actualizar DataContext

Modificamos el **DataContext**

|  |  |
| --- | --- |
| **Clase DataContext** | **Comentarios** |
| using Microsoft.AspNetCore.Identity.EntityFrameworkCore;  using Microsoft.EntityFrameworkCore;  using Taxi.Web.Data.Entities;  namespace Taxi.Web.Data  {  public class DataContext : IdentityDbContext<UserEntity>  {  public DataContext(DbContextOptions<DataContext> options) : base(options)  {  }  public DbSet<TaxiEntity> Taxis { get; set; }  public DbSet<TripEntity> Trips { get; set; }  public DbSet<TripDetailEntity> TripDetails { get; set; }  public DbSet<UserGroupEntity> UserGroups { get; set; }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  base.OnModelCreating(modelBuilder);  modelBuilder.Entity<TaxiEntity>()  .HasIndex(t => t.Plaque)  .IsUnique();  }  }  } |  |

# Interfaz UserHelper

Como las Tablas **User**, **Role** y **UserRole** son tablas prestadas, no podemos acceder a ellas directamente.

Debemos usarlas las Clases **UserManager** y la **RoleManager**

Dentro del Proyecto Taxi.Web creamos la Carpeta **Helpers**

Dentro creamos la Interfaz **IUserHelper**

Aquí tendremos todas las acciones y métodos referidas al manejo de Usuarios.

|  |  |
| --- | --- |
| **Interfaz IUserHelper** | **Comentarios** |
| using Microsoft.AspNetCore.Identity;  using System.Threading.Tasks;  using Taxi.Web.Data.Entities;  namespace Taxi.Web.Helpers  {  public interface IUserHelper  {  Task<UserEntity> GetUserByEmailAsync(string email);  Task<IdentityResult> AddUserAsync(UserEntity user, string password);  Task CheckRoleAsync(string roleName);  Task AddUserToRoleAsync(UserEntity user, string roleName);  Task<bool> IsUserInRoleAsync(UserEntity user, string roleName);  }  } | **GetUserByEmailAsync**  Le pasamos un Email y nos devuelve un Usuario  **AddUserAsync**  Le pasmos un Usuario y un Password y nos crea un Usuario  Nos devuelve un IdentityResult  **CheckRoleAsync**  Chequea si existe un Rol. Si existe… ok, y si no existe lo crea.  **AddUserToRoleAsync**  Le pasamos un Usuario y un Rol y agrega ese Usuario a ese Rol  **IsUserInRoleAsync**  Nos devuelve un booleano, Sirve para preguntar si un Usuario está asignado en un Rol determinado |

Creamos la Clase **UserHelper**

|  |  |
| --- | --- |
| **Clase UserHelper** | **Comentarios** |
| using Microsoft.AspNetCore.Identity;  using System.Threading.Tasks;  using Taxi.Web.Data.Entities;  namespace Taxi.Web.Helpers  {  public class UserHelper : IUserHelper  {  private readonly UserManager<UserEntity> \_userManager;  private readonly RoleManager<IdentityRole> \_roleManager;  public UserHelper(  UserManager<UserEntity> userManager,  RoleManager<IdentityRole> roleManager)  {  \_userManager = userManager;  \_roleManager = roleManager;  }  public async Task<IdentityResult> AddUserAsync(UserEntity user, string password)  {  return await \_userManager.CreateAsync(user, password);  }  public async Task AddUserToRoleAsync(UserEntity user, string roleName)  {  await \_userManager.AddToRoleAsync(user, roleName);  }  public async Task CheckRoleAsync(string roleName)  {  bool roleExists = await \_roleManager.RoleExistsAsync(roleName);  if (!roleExists)  {  await \_roleManager.CreateAsync(new IdentityRole  {  Name = roleName  });  }  }  public async Task<UserEntity> GetUserByEmailAsync(string email)  {  return await \_userManager.FindByEmailAsync(email);  }  public async Task<bool> IsUserInRoleAsync(UserEntity user, string roleName)  {  return await \_userManager.IsInRoleAsync(user, roleName);  }  }  } | Implementa la Interfaz IUserHelper  Inyectamos **UserManager** y **RoleManager**  **AddUserAsync 🡪** CreateAsync del UserManager  **AddUserToRole 🡪** AddToRoleAsync del UserManager  **CheckRoleAsync** 🡪 CreateAsync del RoleManager, usando previamente el RoleExistsAsync del RoleManager.  **GetUserByEmailAsync** 🡪 FindByEmailAsync del UserManager  **IsUserInRoleAsync** 🡪 IsInRoleAsync del UserManager |

## Configuramos la inyección de dependencias del UserHelper

En Startup.cs agregamos esta línea:

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>();

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

Modificamos el **SeedDb**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Common.Enums;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Data

{

public class SeedDb

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

public SeedDb(

DataContext dataContext,

IUserHelper userHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_dataContext.Database.EnsureCreatedAsync();

await CheckRolesAsync();

var admin = await CheckUserAsync("17157729", "Luis", "Nuñez", "luisalbertonu@gmail.com", "156814963", "Espora 2052", UserType.Admin);

var driver = await CheckUserAsync("17000001", "Pablo", "Lacuadri", "lacua@yopmail.com", "350 634 2747", "Villa Santa Ana 123", UserType.Driver);

var user1 = await CheckUserAsync("17000002", "Diego", "Maradona", "maradona@yopmail.com", "350 634 2747", "Villa Fiorito 234", UserType.User);

var user2 = await CheckUserAsync("17000003", "Lionel", "Messi", "messi@yopmail.com", "350 634 2747", "Barcelona 345", UserType.User);

await CheckTaxisAsync(driver, user1, user2);

}

private async Task<UserEntity> CheckUserAsync(

string document,

string firstName,

string lastName,

string email,

string phone,

string address,

UserType userType)

{

var user = await \_userHelper.GetUserByEmailAsync(email);

if (user == null)

{

user = new UserEntity

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

}

return user;

}

private async Task CheckRolesAsync()

{

await \_userHelper.CheckRoleAsync(UserType.Admin.ToString());

await \_userHelper.CheckRoleAsync(UserType.Driver.ToString());

await \_userHelper.CheckRoleAsync(UserType.User.ToString());

}

private async Task CheckTaxisAsync(

UserEntity driver,

UserEntity user1,

UserEntity user2)

{

if (!\_dataContext.Taxis.Any())

{

\_dataContext.Taxis.Add(new TaxiEntity

{

User = driver,

Plaque = "IJX537",

Trips = new List<TripEntity>

{

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.5f,

Source = "Barrio Rosedal",

Target = "Barrio General Paz",

Remarks = "Muy buen servicio",

User = user1

},

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.8f,

Source = "Barrio General Paz",

Target = "Barrio Rosedal",

Remarks = "Conductor muy amable",

User = user1

}

}

});

\_dataContext.Taxis.Add(new TaxiEntity

{

Plaque = "PJM791",

Trips = new List<TripEntity>

{

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.5f,

Source = "Barrio Rosedal",

Target = "Centro",

Remarks = "Muy buen servicio",

User = user2

},

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.8f,

Source = "Centro",

Target = "Barrio Rosedal",

Remarks = "Conductor muy amable",

User = user2

}

}

});

await \_dataContext.SaveChangesAsync();

}

}

}

}

Borramos la BD y agregamos una nueva migración:

drop-database

add-migration AddUserEntitys

update-database

## Configurar como debe ser el Password

En **Startup.cs** agregamos estas líneas que son para definir como se comporta el password.

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddIdentity<UserEntity, IdentityRole>(cfg =>

{

cfg.User.RequireUniqueEmail = true;

cfg.Password.RequireDigit = false;

cfg.Password.RequiredUniqueChars = 0;

cfg.Password.RequireLowercase = false;

cfg.Password.RequireNonAlphanumeric = false;

cfg.Password.RequireUppercase = false;

}).AddEntityFrameworkStores<DataContext>();

services.AddTransient<SeedDb>();

Y más abajo

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseCookiePolicy();

app.UseAuthentication();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

# Mejora del Controlador y las Vistas

Agregamos la Vista Parcial **\_DeleteDialog** en la carpeta **Shared**

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Borrar Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Quiere borrar este registro?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Cerrar</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Borrar</button>

</div>

</div>

</div>

</div>

Modificamos el **Index** de **TaxisController**

@model IEnumerable<Taxi.Web.Data.Entities.TaxiEntity>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a **asp-action**="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Crear Nuevo</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Taxis</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Plaque)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Plaque)

</td>

<td>

<a **asp-action**="Edit" **asp-route-id**="@item.Id" class="btn btn-warning"><i class="glyphicon glyphicon-pencil"></i></a>

<a **asp-action**="Details" **asp-route-id**="@item.Id" class="btn btn-info"><i class="glyphicon glyphicon glyphicon-align-justify"></i></a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<**partial** **name**="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Taxis/Delete/' + item\_to\_delete;

});

});

</script>

}

# Login

## LoginViewModel

Creamos la **LoginViewModel**

|  |  |
| --- | --- |
| **Clase LoginViewModel** | **Comentarios** |
| using System.ComponentModel.DataAnnotations;  namespace Taxi.Web.Models  {  public class LoginViewModel  {  [Required]  [EmailAddress]  public string Username { get; set; }  [Required]  [MinLength(6)]  public string Password { get; set; }  public bool RememberMe { get; set; }  }  } |  |

## Agregamos Login y Logout a IUserHelper

Agregamos a **IUserHelper**

|  |  |
| --- | --- |
| **IUserHelper** | **Comentarios** |
| Task<bool> IsUserInRoleAsync(User user, string roleName);  Task<SignInResult> LoginAsync(LoginViewModel model);  Task LogoutAsync(); | **Login 🡪** Le pasamos el modelo LoginViewModel  **Login 🡪** no se le pasa nada |

Agregamos a **UserHelper**

|  |  |
| --- | --- |
| **UserHelper** | **Comentarios** |
| using Microsoft.AspNetCore.Identity;  using System.Threading.Tasks;  using Taxi.Web.Data.Entities;  using Taxi.Web.Models;  namespace Taxi.Web.Helpers  {  public class UserHelper : IUserHelper  {  private readonly UserManager<UserEntity> \_userManager;  private readonly RoleManager<IdentityRole> \_roleManager;  private readonly SignInManager<UserEntity> \_signInManager;  public UserHelper(  UserManager<UserEntity> userManager,  RoleManager<IdentityRole> roleManager,  SignInManager<UserEntity> signInManager)  {  \_userManager = userManager;  \_roleManager = roleManager;  \_signInManager = signInManager;  }  public async Task<IdentityResult> AddUserAsync(UserEntity user, string password)  {  return await \_userManager.CreateAsync(user, password);  }  public async Task AddUserToRoleAsync(UserEntity user, string roleName)  {  await \_userManager.AddToRoleAsync(user, roleName);  }  public async Task CheckRoleAsync(string roleName)  {  bool roleExists = await \_roleManager.RoleExistsAsync(roleName);  if (!roleExists)  {  await \_roleManager.CreateAsync(new IdentityRole  {  Name = roleName  });  }  }  public async Task<UserEntity> GetUserByEmailAsync(string email)  {  return await \_userManager.FindByEmailAsync(email);  }  public async Task<bool> IsUserInRoleAsync(UserEntity user, string roleName)  {  return await \_userManager.IsInRoleAsync(user, roleName);  }  public async Task<SignInResult> LoginAsync(LoginViewModel model)  {  return await \_signInManager.PasswordSignInAsync(  model.Username,  model.Password,  model.RememberMe,  false);  }  public async Task LogoutAsync()  {  await \_signInManager.SignOutAsync();  }  }  } | **Inyectamos SignInManager**  **Login 🡪** SignInManager.PasswordSignInAsync  El 4to parámetro que está en false es por si quiero que se bloquee el usuario después de un cierto n° de intentos.  **Logout 🡪** SignInManager.SignOutAsync |

## AccountController

Creamos el Controlador en blanco **AccountController** y reemplazamos todo por estas líneas

|  |  |
| --- | --- |
| **Controlador AccountController** | **Comentarios** |
| using Microsoft.AspNetCore.Mvc;  using Taxi.Web.Helpers;  using Taxi.Web.Models;  using System.Linq;  using System.Threading.Tasks;  namespace Taxi.Web.Controllers  {  public class AccountController : Controller  {  private readonly IUserHelper \_userHelper;  public AccountController(IUserHelper userHelper)  {  \_userHelper = userHelper;  }  public IActionResult Login()  {  if (User.Identity.IsAuthenticated)  {  return RedirectToAction("Index", "Home");  }  return View();  }  [HttpPost]  public async Task<IActionResult> Login(LoginViewModel model)  {  if (ModelState.IsValid)  {  var result = await \_userHelper.LoginAsync(model);  if (result.Succeeded)  {  if (Request.Query.Keys.Contains("ReturnUrl"))  {  return Redirect(Request.Query["ReturnUrl"].First());  }  return RedirectToAction("Index", "Home");  }  }  ModelState.AddModelError(string.Empty, "Usuario o contraseña inválidos");  return View(model);  }  public async Task<IActionResult> Logout()  {  await \_userHelper.LogoutAsync();  return RedirectToAction("Index", "Home");  }  }  } | Inyectamos el UserHelper en el Constructor  Acá es donde se hace el Login  si se logueó vuelve a la vista Index del Controlador Home  Acá es donde se hace el Logout |

## Vista Login

Creamos la Vista Login

@model Taxi.Web.Models.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<h2>Login</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div **asp-validation-summary**="ModelOnly"></div>

<div class="form-group">

<label **asp-for**="Username">Username</label>

<input **asp-for**="Username" class="form-control" />

<span **asp-validation-for**="Username" class="text-warning"></span>

</div>

<script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>

<div class="form-group">

<label **asp-for**="Password">Password</label>

<input **asp-for**="Password" **type**="password" class="form-control" />

<span **asp-validation-for**="Password" class="text-warning"></span>

</div>

<div class="form-group">

<div class="form-check">

<input **asp-for**="RememberMe" **type**="checkbox" class="form-check-input" />

<label **asp-for**="RememberMe" class="form-check-label">Remember Me?</label>

</div>

<span **asp-validation-for**="RememberMe" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a **asp-action**="Register" class="btn btn-primary">Registrar Nuevo Usuario</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

## Agregamos Menú Login-Logout

En **\_Layout.cshtml** agregamos:

|  |  |
| --- | --- |
| <div class="navbar-collapse collapse">  <ul class="nav navbar-nav">  <li><a **asp-area**="" **asp-controller**="Home" **asp-action**="Index">Home</a></li>  <li><a **asp-area**="" **asp-controller**="Home" **asp-action**="About">About</a></li>  <li><a **asp-area**="" **asp-controller**="Home" **asp-action**="Contact">Contact</a></li>  @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))  {  <li><a **asp-area**="" **asp-controller**="Taxis" **asp-action**="Index">Taxis</a></li>  }  </ul>  <ul class="nav navbar-nav navbar-right">  @if (User.Identity.IsAuthenticated)  {  <li><a **asp-area**="" **asp-controller**="Account" **asp-action**="ChangeUser">@User.Identity.Name</a></li>  <li><a **asp-area**="" **asp-controller**="Account" **asp-action**="Logout">Logout</a></li>  }  else  {  <li><a **asp-area**="" **asp-controller**="Account" **asp-action**="Login">Login</a></li>  }  </ul>  </ul> | Agregamos un menú a la derecha.  Si está logueado dirá Logout  Si no está logueado dirá Login |

## Colocar Permisos a los Controladores

En **TaxisController** agregamos:

namespace Taxi.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class TaxisController : Controller

{

private readonly DataContext \_context;

# API

Creamos dentro de **Controllers** una Carpeta llamada **API**

Dentro creamos el Controlador API **TaxisController** del tipo “Controlador de API con acciones que usan Entity Framework”

Borramos todos los métodos excepto el Get, y a ese lo modificamos para que nos traiga el Taxi de una Patente:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Threading.Tasks;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

namespace Taxi.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

public class TaxisController : ControllerBase

{

private readonly DataContext \_context;

public TaxisController(DataContext context)

{

\_context = context;

}

// GET: api/Taxis/5

[HttpGet("{plaque}")]

public async Task<IActionResult> GetTaxiEntity([FromRoute] string plaque)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

TaxiEntity taxiEntity = await \_context.Taxis.FirstOrDefaultAsync(t => t.Plaque == plaque);

if (taxiEntity == null)

{

return NotFound();

}

return Ok(taxiEntity);

}

}

}

Modificamos el Método **Get**

// GET: api/Taxis/5

[HttpGet("{plaque}")]

public async Task<IActionResult> GetTaxiEntity([FromRoute] string plaque)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

plaque = plaque.ToUpper();

TaxiEntity taxiEntity = await \_context.Taxis

.Include(t=>t.Trips)

.FirstOrDefaultAsync(t => t.Plaque == plaque);

if (taxiEntity == null)

{

return NotFound();

}

return Ok(taxiEntity);

}

Cuando lo probamos da error, porque se produce una referencia circular (Taxi llama a Trips, y Trips llama a Taxi)

Entonces hay que crear un modelo de respuesta

Agregamos en el Proyecto **Common**, una carpeta llamada **Models**

Dentro creamos la Clase **UserResponse**

using Taxi.Common.Enums;

namespace Taxi.Common.Models

{

public class UserResponse

{

public string Id { get; set; }

public string Email { get; set; }

public string PhoneNumber { get; set; }

public string Document { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

public string PicturePath { get; set; }

public UserType UserType { get; set; }

public string FullName => $"{FirstName} {LastName}";

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

}

También creamos la Clase **TripDetailResponse**

using System;

namespace Taxi.Common.Models

{

public class TripDetailResponse

{

public int Id { get; set; }

public DateTime Date { get; set; }

public DateTime DateLocal => Date.ToLocalTime();

public double Latitude { get; set; }

public double Longitude { get; set; }

}

}

También creamos la Clase **TripResponse**

using System;

using System.Collections.Generic;

namespace Taxi.Common.Models

{

public class TripResponse

{

public int Id { get; set; }

public DateTime StartDate { get; set; }

public DateTime StartDateLocal => StartDate.ToLocalTime();

public DateTime? EndDate { get; set; }

public DateTime? EndDateLocal => EndDate?.ToLocalTime();

public string Source { get; set; }

public string Target { get; set; }

public float Qualification { get; set; }

public double SourceLatitude { get; set; }

public double SourceLongitude { get; set; }

public double TargetLatitude { get; set; }

public double TargetLongitude { get; set; }

public string Remarks { get; set; }

public List<TripDetailResponse> TripDetails { get; set; }

public UserResponse User { get; set; }

}

}

También agregamos la Clase **TaxiResponse**

using System.Collections.Generic;

namespace Taxi.Common.Models

{

public class TaxiResponse

{

public int Id { get; set; }

public string Plaque { get; set; }

public List<TripResponse> Trips { get; set; }

public UserResponse User { get; set; }

}

}

Creamos la Interfaz **IConverterHelper**

using Taxi.Common.Models;

using Taxi.Web.Data.Entities;

namespace Taxi.Web.Helpers

{

public interface IConverterHelper

{

TaxiResponse ToTaxiResponse(TaxiEntity taxiEntity);

}

}

Creamos la implementación de la Interfaz, como la Clase **ConverterHelper**

using System.Linq;

using Taxi.Common.Models;

using Taxi.Web.Data.Entities;

namespace Taxi.Web.Helpers

{

public class ConverterHelper : IConverterHelper

{

public TaxiResponse ToTaxiResponse(TaxiEntity taxiEntity)

{

return new TaxiResponse

{

Id = taxiEntity.Id,

Plaque = taxiEntity.Plaque,

Trips = taxiEntity.Trips?.Select(t => new TripResponse

{

EndDate = t.EndDate,

Id = t.Id,

Qualification = t.Qualification,

Remarks = t.Remarks,

Source = t.Source,

SourceLatitude = t.SourceLatitude,

SourceLongitude = t.SourceLongitude,

StartDate = t.StartDate,

Target = t.Target,

TargetLatitude = t.TargetLatitude,

TargetLongitude = t.TargetLongitude,

TripDetails = t.TripDetails?.Select(td => new TripDetailResponse

{

Date = td.Date,

Id = td.Id,

Latitude = td.Latitude,

Longitude = td.Longitude

}).ToList(),

User = ToUserResponse(t.User)

}).ToList(),

User = ToUserResponse(taxiEntity.User)

};

}

private UserResponse ToUserResponse(UserEntity user)

{

if (user == null)

{

return null;

}

return new UserResponse

{

Address = user.Address,

Document = user.Document,

Email = user.Email,

FirstName = user.FirstName,

Id = user.Id,

LastName = user.LastName,

PhoneNumber = user.PhoneNumber,

PicturePath = user.PicturePath,

UserType = user.UserType

};

}

}

}

Agregamos la inyección en **Startup.cs**

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>();

services.AddScoped<IConverterHelper, ConverterHelper>();

services.AddDbContext<DataContext>(cfg =>

Modificamos el **TaxisController**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Threading.Tasks;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

public class TaxisController : ControllerBase

{

private readonly DataContext \_context;

private readonly IConverterHelper \_converterHelper;

public TaxisController(DataContext context, IConverterHelper converterHelper)

{

\_context = context;

\_converterHelper = converterHelper;

}

// GET: api/Taxis/5

[HttpGet("{plaque}")]

public async Task<IActionResult> GetTaxiEntity([FromRoute] string plaque)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

plaque = plaque.ToUpper();

TaxiEntity taxiEntity = await \_context.Taxis

.Include(t => t.User)

.Include(t => t.Trips)

.ThenInclude(t => t.TripDetails)

.Include(t => t.Trips)

.ThenInclude(t => t.User)

.FirstOrDefaultAsync(t => t.Plaque == plaque);

if (taxiEntity == null)

{

return NotFound();

}

return Ok(\_converterHelper.ToTaxiResponse(taxiEntity));

}

}

}

Publicamos el **API**

**APP MOVIL**

# Mostrar la Home Page

Borramos MainPage.xaml y MainPageViewModel

Actualizamos los Nuggets

Agregamos el Nugget **Xamarin.Forms.Maps** a todos los Proyectos Prism (la versión 4.4.0.991640)

Creamos la **HomePage**

Modificamos la **HomePageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class HomePageViewModel : ViewModelBase

{

public HomePageViewModel(INavigationService navigationService):base(navigationService)

{

Title = "Taxi Qualifier";

}

}

}

Modificamos la Clase **App.xaml.cs**

using Prism;

using Prism.Ioc;

using Taxi.Prism.ViewModels;

using Taxi.Prism.Views;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]

namespace Taxi.Prism

{

public partial class App

{

/\*

\* The Xamarin Forms XAML Previewer in Visual Studio uses System.Activator.CreateInstance.

\* This imposes a limitation in which the App class must have a default constructor.

\* App(IPlatformInitializer initializer = null) cannot be handled by the Activator.

\*/

public App() : this(null) { }

public App(IPlatformInitializer initializer) : base(initializer) { }

protected override async void OnInitialized()

{

InitializeComponent();

await NavigationService.NavigateAsync("NavigationPage/HomePage");

}

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<HomePage, HomePageViewModel>();

}

}

}

Modificamos la **HomePage.xaml**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.HomePage"

Title="{Binding Title}">

<StackLayout>

<maps:Map x:Name="MyMap"

MapType="Street"/>

<StackLayout Padding="5">

<Button BackgroundColor="Green"

CornerRadius="20"

HeightRequest="40"

Margin="10,0"

Text="Iniciar Viaje"

TextColor="White"/>

</StackLayout>

</StackLayout>

</ContentPage>

En **AndroidManifest** ponemos

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.companyname.appname">

<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />

<uses-permission android:name="android.permission.INTERNET" />

<application android:label="Taxi.Prism.Android" android:icon="@mipmap/ic\_launcher">

<meta-data

android:name="com.google.android.maps.v2.API\_KEY"

android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

</application>

</manifest>

# Menu

Agregamos en **Common.Models** la Clase **Menu**

namespace Taxi.Common.Models

{

public class Menu

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

}

}

Agregamos en **Common.Models** la Clase **MenuItemViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class MenuItemViewModel : Menu

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectMenuCommand;

public MenuItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectMenuCommand => \_selectMenuCommand ?? (\_selectMenuCommand = new DelegateCommand(SelectMenuAsync));

private async void SelectMenuAsync()

{

await \_navigationService.NavigateAsync($"/TaxiMasterDetailPage/NavigationPage/{PageName}");

}

}

}

Cartgamos la imagen del logo en la carpeta **Resources/drawable** de proyecto Android

Agregamos la **TaxiMasterDetailPage** (del tipo MasterDetailPage)

<?xml version="1.0" encoding="utf-8" ?>

<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiMasterDetailPage">

<MasterDetailPage.Master>

<ContentPage Title="Menu">

<StackLayout Padding="20">

<Image HeightRequest="150"

Source="taxi"/>

<ListView BackgroundColor="Transparent"

ItemsSource="{Binding Menus}"

HasUnevenRows="True"

SeparatorVisibility="None">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectMenuCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image Grid.Column="0"

HeightRequest="50"

Source="{Binding Icon}"

WidthRequest="50"/>

<Label Grid.Column="1"

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

</MasterDetailPage.Master>

</MasterDetailPage>

Agregamos a la carpeta **Resources/drawable** los íconos del menú

Modificamos la **TaxiMasterDetailPageViewModel**

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class TaxiMasterDetailPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

public TaxiMasterDetailPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

LoadMenus();

}

public ObservableCollection<MenuItemViewModel> Menus { get; set; }

private void LoadMenus()

{

List<Menu> menus = new List<Menu>

{

new Menu

{

Icon = "ic\_airport\_shuttle",

PageName = "HomePage",

Title = "New trip"

},

new Menu

{

Icon = "ic\_local\_taxi",

PageName = "TaxiHistoryPage",

Title = "See taxi history"

},

new Menu

{

Icon = "ic\_people",

PageName = "GroupPage",

Title = "Admin my user group"

},

new Menu

{

Icon = "ic\_account\_circle",

PageName = "ModifyUserPage",

Title = "Modify User"

},

new Menu

{

Icon = "ic\_report",

PageName = "ReportPage",

Title = "Report an incident"

},

new Menu

{

Icon = "ic\_exit\_to\_app",

PageName = "LoginPage",

Title = "Log in"

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title

}).ToList());

}

}

}

Agregamos la **TaxiHistoryPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiHistoryPage"

Title="{Binding Title}">

</ContentPage>

Agregamos la **TaxiHistoryViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class TaxiHistoryPageViewModel : ViewModelBase

{

public TaxiHistoryPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Taxi History";

}

}

}

Agregamos la **GroupPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.GroupPage"

Title="{Binding Title}">

</ContentPage>

Agregamos la **GroupPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class GroupPageViewModel : ViewModelBase

{

public GroupPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Admin my user group";

}

}

}

Agregamos la **ModifyUserPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.ModifyUserPage"

Title="{Binding Title}">

</ContentPage>

Agregamos la **ModifyUserPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class ModifyUserPageViewModel : ViewModelBase

{

public ModifyUserPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Modify User";

}

}

}

Agregamos la **ReportPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.ReportPage"

Title="{Binding Title}">

</ContentPage>

Agregamos la **ReportPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class ReportPageViewModel : ViewModelBase

{

public ReportPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Report an incident";

}

}

}

Agregamos la **LoginPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.LoginPage"

Title="{Binding Title}">

</ContentPage>

Agregamos la **LoginPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class LoginPageViewModel : ViewModelBase

{

public LoginPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Login";

}

}

}

Modificamos en **App.xaml.cs**

await NavigationService.NavigateAsync("/TaxiMasterDetailPage/NavigationPage/HomePage");

# Mover el Mapa a la Posición del Usuario

Agregamos **Xam Plugin Geolocator** a Common y a los Proyectos Prism.

Obtener una KEY para los mapas desde la web: <https://developers.google.com/maps/?hl=es-419>

En Android dentro de Properties hay que ir al archivo **Android Manifest**

|  |  |
| --- | --- |
| **Android Manifest** | **Comentarios** |
| <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.luisnu.MyLeasing" android:installLocation="auto">  <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />  <uses-permission android:name="android.permission.INTERNET" />  <uses-permission android:name="android.permission.ACCESS\_MOCK\_LOCATION" />  <uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />  <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  <application  <meta-data  android:name="com.google.android.maps.v2.API\_KEY"  android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />  android:label="My Leasing" android:icon="@mipmap/ic\_launcher"></application>  </manifest> |  |

En iOS vamos al archivo **Info.plist** y agregamos

|  |  |
| --- | --- |
| **Info.plist** | **Comentarios** |
| <key>NSLocationAlwaysUsageDescription</key>  <string>Can we use your location at all times?</string>  <key>NSLocationWhenInUseUsageDescription</key>  <string>Can we use your location when your app is being used?</string>  <key>NSLocationAlwaysAndWhenInUseUsageDescription</key>  <string>Can we use your location at all times?</string> |  |

En **MainActivity** ponemos esta línea

|  |  |
| --- | --- |
| **MainActivity** | **Comentarios** |
| protected override void OnCreate(Bundle bundle)  {  TabLayoutResource = Resource.Layout.Tabbar;  ToolbarResource = Resource.Layout.Toolbar;  base.OnCreate(bundle);  global::Xamarin.Forms.Forms.Init(this, bundle);  Xamarin.FormsMaps.Init(this, bundle);  FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);  new SfRotatorRenderer();  new SfBusyIndicatorRenderer();  LoadApplication(new App(new AndroidInitializer()));  } |  |

Y mas abajo:

|  |  |
| --- | --- |
| **MainActivity** | **Comentarios** |
| public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity  {  protected override void OnCreate(Bundle bundle)  {  TabLayoutResource = Resource.Layout.Tabbar;  ToolbarResource = Resource.Layout.Toolbar;  base.OnCreate(bundle);  CrossCurrentActivity.Current.Init(this, bundle);  global::Xamarin.Forms.Forms.Init(this, bundle);  Xamarin.FormsMaps.Init(this, bundle);  FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);  new SfRotatorRenderer();  new SfBusyIndicatorRenderer();  LoadApplication(new App(new AndroidInitializer()));  }  public override void OnRequestPermissionsResult(  int requestCode,  string[] permissions,  [GeneratedEnum] Permission[] grantResults)  {  PermissionsImplementation.Current.OnRequestPermissionsResult(  requestCode,  permissions,  grantResults);  }  } |  |

En **AppDelegate** ponemos esta línea

|  |  |
| --- | --- |
| **AppDelegate** | **Comentarios** |
| public override bool FinishedLaunching(UIApplication app, NSDictionary options)  {  global::Xamarin.Forms.Forms.Init();  Xamarin.FormsMaps.Init();  SfRotatorRenderer.Init();  FFImageLoading.Forms.Platform.CachedImageRenderer.Init();  new SfRotatorRenderer();  new SfBusyIndicatorRenderer();  LoadApplication(new App(new iOSInitializer()));  return base.FinishedLaunching(app, options);  } |  |

En **HomePage** agregamos

|  |  |
| --- | --- |
| **HomePage** | **Comentarios** |
| <?xml version="1.0" encoding="utf-8" ?>  <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  xmlns:prism="http://prismlibrary.com"  xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"  prism:ViewModelLocator.AutowireViewModel="True"  x:Class="Taxi.Prism.Views.HomePage"  Title="{Binding Title}">  <StackLayout>  <maps:Map x:Name="MyMap"  MapType="Street"/>  <StackLayout Padding="5">  <Button BackgroundColor="Black"  CornerRadius="20"  HeightRequest="40"  Margin="10,0"  Text="Iniciar Viaje"  TextColor="Yellow"/>  </StackLayout>  </StackLayout>  </ContentPage> |  |

Creamos en **Common** una carpeta **Services**

Agregamos en **Common/Services** la Interfaz **IGeolocatorService**

|  |  |
| --- | --- |
| **IGeolocatorService** | **Comentarios** |
| using System.Threading.Tasks;  namespace Taxi.Common.Services  {  public interface IGeolocatorService  {  double Latitude { get; set; }  double Longitude { get; set; }  Task GetLocationAsync();  }  } |  |

Y la implementación **GeolocatorService**

|  |  |
| --- | --- |
| **GeolocatorService** | **Comentarios** |
| using System;  using System.Threading.Tasks;  using Plugin.Geolocator;  namespace Taxi.Common.Services  {  public class GeolocatorService : IGeolocatorService  {  public double Latitude { get; set; }  public double Longitude { get; set; }  public async Task GetLocationAsync()  {  try  {  var locator = CrossGeolocator.Current;  locator.DesiredAccuracy = 50;  var location = await locator.GetPositionAsync();  Latitude = location.Latitude;  Longitude = location.Longitude;  }  catch (Exception ex)  {  ex.ToString();  }  }  }  } |  |

Agregamos en **App.xaml.cs**

|  |  |
| --- | --- |
| **App.xaml.cs** | **Comentarios** |
| protected override void RegisterTypes(IContainerRegistry containerRegistry)  {  containerRegistry.Register<IApiService, ApiService>();  containerRegistry.Register<IGeolocatorService, GeolocatorService>();  containerRegistry.RegisterForNavigation<NavigationPage>(); } |  |

En **HomePage.xaml.cs**

|  |  |
| --- | --- |
| **HomePage.xaml.cs** | **Comentarios** |
| using Plugin.Permissions;  using Plugin.Permissions.Abstractions;  using System.Threading.Tasks;  using Taxi.Common.Services;  using Xamarin.Forms;  using Xamarin.Forms.Maps;  namespace Taxi.Prism.Views  {  public partial class HomePage : ContentPage  {  private readonly IGeolocatorService \_geolocatorService;  public HomePage(IGeolocatorService geolocatorService)  {  InitializeComponent();  \_geolocatorService = geolocatorService;  }  protected override void OnAppearing()  {  base.OnAppearing();  MoveMapToCurrentPositionAsync();  }  private async void MoveMapToCurrentPositionAsync()  {  bool isLocationPermision = await CheckLocationPermisionsAsync();  if (isLocationPermision)  {  MyMap.IsShowingUser = true;  await \_geolocatorService.GetLocationAsync();  if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)  {  Position position = new Position(  \_geolocatorService.Latitude,  \_geolocatorService.Longitude);  MoveMap(position);  }  }  }  private void MoveMap(Position position)  {  MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(  position,  Distance.FromKilometers(.2)));  }  private async Task<bool> CheckLocationPermisionsAsync()  {  PermissionStatus permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);  PermissionStatus permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);  PermissionStatus permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);  bool isLocationEnabled = permissionLocation == PermissionStatus.Granted ||  permissionLocationAlways == PermissionStatus.Granted ||  permissionLocationWhenInUse == PermissionStatus.Granted;  if (isLocationEnabled)  {  return true;  }  await CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);  permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);  permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);  permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);  return permissionLocation == PermissionStatus.Granted ||  permissionLocationAlways == PermissionStatus.Granted ||  permissionLocationWhenInUse == PermissionStatus.Granted;  }  }  } |  |

# Consumir API

En el método **GetTaxiEntity** de **TaxisControllers** modificamos:

// GET: api/Taxis/5

[HttpGet("{plaque}")]

public async Task<IActionResult> GetTaxiEntity([FromRoute] string plaque)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

plaque = plaque.ToUpper();

TaxiEntity taxiEntity = await \_context.Taxis

.Include(t => t.User) //Conductor

.Include(t => t.Trips)

.ThenInclude(t => t.TripDetails)

.Include(t => t.Trips)

.ThenInclude(t => t.User) //Pasajero

.FirstOrDefaultAsync(t => t.Plaque == plaque);

if (taxiEntity == null)

{

taxiEntity = new TaxiEntity { Plaque = plaque.ToUpper() };

\_context.Taxis.Add(taxiEntity);

await \_context.SaveChangesAsync();

}

return Ok(\_converterHelper.ToTaxiResponse(taxiEntity));

}

En el Proyecto **Common** en la carpeta **Models** creamos la Clase **Response**

namespace Taxi.Common.Models

{

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

}

En el Proyecto **Common** en la carpeta **Services** creamos la Interfaz **IApiService**

using Taxi.Common.Models;

using System.Threading.Tasks;

namespace Taxi.Common.Services

{

public interface IApiService

{

Task<Response> GetTaxiAsync(

string Plaque,

string urlBase,

string servicePrefix,

string controller);

}

}

Creamos la Clase **ApiService**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Threading.Tasks;

using Taxi.Common.Models;

namespace Taxi.Common.Services

{

public class ApiService : IApiService

{

public async Task<Response> GetTaxiAsync(string plaque, string urlBase, string servicePrefix, string controller)

{

try

{

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase),

};

string url = $"{servicePrefix}{controller}/{plaque}";

HttpResponseMessage response = await client.GetAsync(url);

string result = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

TaxiResponse model = JsonConvert.DeserializeObject<TaxiResponse>(result);

return new Response

{

IsSuccess = true,

Result = model

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

}

}

Agregamos la inyección en **App.xaml.cs**

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IGeolocatorService, GeolocatorService>();

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<HomePage, HomePageViewModel>();

containerRegistry.RegisterForNavigation<TaxiMasterDetailPage, TaxiMasterDetailPageViewModel>();

containerRegistry.RegisterForNavigation<TaxiHistoryPage, TaxiHistoryPageViewModel>();

containerRegistry.RegisterForNavigation<GroupPage, GroupPageViewModel>();

containerRegistry.RegisterForNavigation<ModifyUserPage, ModifyUserPageViewModel>();

containerRegistry.RegisterForNavigation<ReportPage, ReportPageViewModel>();

containerRegistry.RegisterForNavigation<LoginPage, LoginPageViewModel>();

}

En **App.xaml** agregamos:

<?xml version="1.0" encoding="utf-8" ?>

<prism:PrismApplication xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="clr-namespace:Prism.DryIoc;assembly=Prism.DryIoc.Forms"

x:Class="Taxi.Prism.App">

<Application.Resources>

<!-- Application resource dictionary -->

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">http://keypress.serveftp.net:88/TaxiApi/</x:String>

</ResourceDictionary>

</Application.Resources>

</prism:PrismApplication>

Agregamos la imagen del ícono **ic\_more\_vert** en la carpeta **drawable**

Modificamos **TaxiResponse**

using System.Collections.Generic;

using System.Linq;

namespace Taxi.Common.Models

{

public class TaxiResponse

{

public int Id { get; set; }

public string Plaque { get; set; }

public List<TripResponse> Trips { get; set; }

public UserResponse User { get; set; }

public float Qualification => Trips == null ? 0 : Trips.Average(t => t.Qualification);

public int NumberOfTrips => Trips == null ? 0 : Trips.Count;

}

}

Modificamos **TaxiHistoryPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiHistoryPage"

Title="{Binding Title}">

<StackLayout Padding="5">

<StackLayout Orientation="Horizontal">

<Label Text="Plaque"

VerticalOptions="Center"/>

<Entry HorizontalOptions="FillAndExpand"

Placeholder="Enter plaque..."

Text="{Binding Plaque}"

VerticalOptions="Center"/>

<Button BackgroundColor="Navy"

Command="{Binding CheckPlaqueCommand}"

CornerRadius="20"

HeightRequest="40"

Margin="10,0"

Text="Check Plaque"

TextColor="White"/>

</StackLayout>

<Label FontAttributes="Bold"

FontSize="48"

HorizontalOptions="Center"

Text="{Binding Taxi.Qualification, StringFormat='{0:N2}'}"/>

<StackLayout Orientation="Horizontal">

<Label Text="Number of trips:"/>

<Label Text="{Binding Taxi.NumberOfTrips, StringFormat='{0:N0}'}"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Driver:"/>

<Label Text="{Binding Taxi.User.FullName, StringFormat='{0:N0}'}"/>

</StackLayout>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width=".5\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="48" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

FontAttributes="Bold"

Text="Start Date"/>

<Label Grid.Column="1"

FontAttributes="Bold"

Text="Score"/>

<Label Grid.Column="2"

FontAttributes="Bold"

Text="Remarks"/>

</Grid>

<CollectionView ItemsSource="{Binding Taxi.Trips}">

<CollectionView.ItemsLayout>

<GridItemsLayout Orientation="Vertical"/>

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width=".5\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Text="{Binding StartDateLocal, StringFormat='{0:yyyy/MM/dd HH:mm}'}"

VerticalOptions="Center"/>

<Label Grid.Column="1"

Text="{Binding Qualification, StringFormat='{0:N2}'}"

VerticalOptions="Center"/>

<Label Grid.Column="2"

Text="{Binding Remarks}"

VerticalOptions="Center"/>

<Image Grid.Column="3"

Source="ic\_more\_vert"/>

</Grid>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</StackLayout>

</ContentPage>

Modificamos la **TaxiHistoryPageViewModel**

# Agregar SfBusyIndicator

Agregamos el Nugget **Syncfusion.Xamarin.SfBusyIndicator (versión 17.3.0.9-beta)** a todos los proyectos Prism

Poner en el **código de SyncFusion**

|  |  |
| --- | --- |
| **App.xaml.cs** | **Comentarios** |
| protected override async void OnInitialized()  {  InitializeComponent();  Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("MTY2MzIyQDMxMzcyZTMzMmUzMFVnNW5KSnM2dTZmRDljWm1RYTduQXFwRmNKSzVPWk1lT1JGSFRySXZCUTA9");  InitializeComponent();  await NavigationService.NavigateAsync("/TaxiMasterDetailPage/NavigationPage/HomePage");  } |  |

En **MainActivity** de Android agregamos:

|  |  |
| --- | --- |
| **MainActivity** | **Comentarios** |
| new SfBusyIndicatorRenderer(); |  |

En **AppDelegate** de iOS agregamos

|  |  |
| --- | --- |
| **AppDelegate** | **Comentarios** |
| new SfBusyIndicatorRenderer(); |  |

En **TaxiHistoryPage** agregamos

|  |  |
| --- | --- |
| **TaxiHistoryPage** | **Comentarios** |
| xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms" |  |

Modificamos la **TaxiHistoryPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiHistoryPage"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<StackLayout Orientation="Horizontal">

<Label Text="Patente:"

VerticalOptions="Center"/>

<Entry HorizontalOptions="FillAndExpand"

Placeholder="Ingrese Patente..."

Text="{Binding Plaque}"

VerticalOptions="Center"/>

<Button BackgroundColor="Navy"

Command="{Binding CheckPlaqueCommand}"

CornerRadius="20"

HeightRequest="40"

Margin="10,0"

Text="Chequear"

TextColor="White"/>

</StackLayout>

<Label FontAttributes="Bold"

FontSize="48"

HorizontalOptions="Center"

Text="{Binding Taxi.Qualification, StringFormat='{0:N2}'}"/>

<StackLayout Orientation="Horizontal">

<Label Text="N° de viajes:"/>

<Label Text="{Binding Taxi.NumberOfTrips, StringFormat='{0:N0}'}"/>

</StackLayout>

<StackLayout Orientation="Horizontal">

<Label Text="Conductor:"/>

<Label Text="{Binding Taxi.User.FullName, StringFormat='{0:N0}'}"/>

</StackLayout>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="48" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

FontAttributes="Bold"

Text="Fecha Inicio"/>

<Label Grid.Column="1"

FontAttributes="Bold"

Text="Calificación"/>

<Label Grid.Column="2"

FontAttributes="Bold"

Text="Comentarios"/>

</Grid>

<CollectionView ItemsSource="{Binding Taxi.Trips}">

<CollectionView.ItemsLayout>

<GridItemsLayout Orientation="Vertical"/>

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width=".5\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Text="{Binding StartDateLocal, StringFormat='{0:dd/MM/yyyy}'}"

VerticalOptions="Center"/>

<Label Grid.Column="1"

Text="{Binding Qualification, StringFormat='{0:N2}'}"

VerticalOptions="Center"/>

<Label Grid.Column="2"

Text="{Binding Remarks}"

VerticalOptions="Center"/>

<Image Grid.Column="3"

Source="ic\_more\_vert"/>

</Grid>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="Silver"

HorizontalOptions="Center"

TextColor="White"

IsVisible="{Binding IsRunning}"

Title="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos la **TaxiHistoryPageViewModel**

private bool \_isRunning;

…………

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

………

IsRunning = true;

string url = App.Current.Resources["UrlAPI"].ToString();

Response response = await \_apiService.GetTaxiAsync(Plaque, url, "api", "/Taxis");

IsRunning = false;

# Agregar SfRating

Agregamos el Nugget **Syncfusion.Xamarin.SfRating (versión 17.3.0.3.0)** a todos los proyectos Prism

En **MainActivity** de Android agregamos:

|  |  |
| --- | --- |
| **MainActivity** | **Comentarios** |
| new SfRatingRenderer(); |  |

En **AppDelegate** de iOS agregamos

|  |  |
| --- | --- |
| **AppDelegate** | **Comentarios** |
| new SfRatingRenderer(); |  |

En **TaxiHistoryPage** agregamos

|  |  |
| --- | --- |
| **TaxiHistoryPage** | **Comentarios** |
| xmlns:rating="clr-namespace:Syncfusion.SfRating.XForms;assembly=Syncfusion.SfRating.XForms" |  |

Y en **TaxiHistoryPage** reemplazamos

<Label FontAttributes="Bold"

FontSize="48"

HorizontalOptions="Center"

Text="{Binding Taxi.Qualification, StringFormat='{0:N2}'}"/>

por:

<rating:SfRating HorizontalOptions="Center"

IsEnabled="false"

Margin="0,5"

Precision="Exact"

Value="{Binding Taxi.Qualification}" />

<Label FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

Text="{Binding Taxi.Qualification, StringFormat='Qualification: {0:N2}'}"/>

# Chequear la Conexión a Internet

Agregamos el Nugget **Xam.Plugin.Connectivity** a todos los Proyectos ***menos Web***

Agregamos a **IApiService**

|  |  |
| --- | --- |
| **IApiService** | **Comentarios** |
| Task<bool> CheckConnection(string url); |  |

Agregamos a **ApiService**

|  |  |
| --- | --- |
| **ApiService** | **Comentarios** |
| public async Task<bool> CheckConnectionAsync(string url)  {  if (!CrossConnectivity.Current.IsConnected)  {  return false;  }  return await CrossConnectivity.Current.IsRemoteReachable(url);  } |  |

Modificamos la **TaxiHistoryPageViewModel**

IsRunning = true;

var url = App.Current.Resources["UrlAPI"].ToString();

var connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

await App.Current.MainPage.DisplayAlert("Error", "Check the internet connection.", "Accept");

return;

}

Response response = await \_apiService.GetTaxiAsync(Plaque, url, "api", "/Taxis");

IsRunning = false;

# Adicionando Estilos

Página para ver combinaciones de colores

<https://color.adobe.com/es/explore>

Usaremos estos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Danger | Primary | Secondary | Accent | Background |



Los definimos en el diccionario de Recursos en **App.xaml**

|  |  |
| --- | --- |
| **App.xaml** | **Comentarios** |
| <!-- Colors -->  <Color x:Key="ColorBackground">#FFFF05</Color>  <Color x:Key="ColorPrimary">#3D3D3D</Color>  <Color x:Key="ColorSecondary">#A6ABAB</Color>  <Color x:Key="ColorDanger">#8D07F6</Color>  <Color x:Key="ColorAccent">#0029FA</Color>  <Color x:Key="ColorFont">#3D3D3D</Color>  <Color x:Key="ColorFontInverse">#FFFF05</Color> |  |

En todas las Vistas menos en **TaxiMasterDetailPage** agregamos:

BackgroundColor="{StaticResource ColorBackground}"

En **TaxiMasterDetailPage** agregamos:

<ContentPage Title="Menu"

BackgroundColor="{StaticResource ColorBackground}">

Modificamos **Android/Resources/values/styles.xml**:

<?xml version="1.0" encoding="utf-8" ?>

<resources>

<style name="MainTheme" parent="MainTheme.Base">

</style>

<!-- Base theme applied no matter what API -->

<style name="MainTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar">

<!--If you are using revision 22.1 please use just windowNoTitle. Without android:-->

<item name="windowNoTitle">true</item>

<!--We will be using the toolbar so no need to show ActionBar-->

<item name="windowActionBar">false</item>

<!-- Set theme colors from http://www.google.com/design/spec/style/color.html#color-color-palette -->

<!-- colorPrimary is used for the default action bar background -->

<item name="colorPrimary">#3D3D3D</item>

<!-- colorPrimaryDark is used for the status bar -->

<item name="colorPrimaryDark">#A6ABAB</item>

<!-- colorAccent is used as the default value for colorControlActivated

which is used to tint widgets -->

<item name="colorAccent">#0029FA</item>

<!-- You can also set colorControlNormal, colorControlActivated

colorControlHighlight and colorSwitchThumbNormal. -->

<item name="windowActionModeOverlay">true</item>

<item name="android:datePickerDialogTheme">@style/AppCompatDialogStyle</item>

</style>

<style name="AppCompatDialogStyle" parent="Theme.AppCompat.Light.Dialog">

<item name="colorAccent">#0029FA</item>

</style>

<style name="Theme.Splash" parent="android:Theme">

<item name="android:windowBackground">@drawable/leasing\_splash</item>

<item name="android:windowNoTitle">true</item>

</style>

</resources>

# Adicionando Estilos a los Botones

Agregamos al diccionario de recursos

|  |  |
| --- | --- |
| **App.xaml** | **Comentarios** |
| <!-- Styles -->  <Style TargetType="Button">  <Setter Property="BackgroundColor" Value="{StaticResource ColorAccent}" />  <Setter Property="HorizontalOptions" Value="FillAndExpand" />  <Setter Property="TextColor" Value="{StaticResource ColorFontInverse}" />  </Style>  <Style TargetType="Label">  <Setter Property="TextColor" Value="{StaticResource ColorFont}" />  </Style>  <Style x:Key="SecondaryButton" TargetType="Button">  <Setter Property="BackgroundColor" Value="{StaticResource ColorSecondary}" />  </Style>  <Style x:Key="DangerButton" TargetType="Button">  <Setter Property="BackgroundColor" Value="{StaticResource ColorDanger}" />  </Style> |  |

Modificamos el Botón **Login** en **HomePage**

<StackLayout Padding="5">

<Button Margin="10,0"

Text="Iniciar Viaje"/>

</StackLayout>

Modificamos **TaxiHistoryPage**

<Button Command="{Binding CheckPlaqueCommand}"

HeightRequest="40"

Margin="10,0"

Text="Chequear"/>

………………………….

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorDanger}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

# Splash e ícono en Android

Debemos tener una imagen splash.png de 480 x 800 pixeles y la agregamos a la carpeta drawable.

Dentro de **Android/Resources/values** hay un archivo llamado **styles.xml**

Allí agregamos

|  |  |
| --- | --- |
| **styles.xml** | **Comentarios** |
| <style name="Theme.Splash" parent="android:Theme">  <item name="android:windowBackground">@drawable/splash</item>  <item name="android:windowNoTitle">true</item>  </style> | drawable/splash 🡨 el archivo que agregamos en el punto anterior |

En Android hacemos clic derecho – Agregar – Nuevo Elemento – Actividad 🡪 **SplashActivity**

Lo modificamos por

|  |  |
| --- | --- |
| **SplashActivity** | **Comentarios** |
| using Android.App;  using Android.OS;  namespace Taxi.Prism.Droid  {  [Activity(  Theme = "@style/Theme.Splash",  MainLauncher = true,  NoHistory = true)]  public class SplashActivity : Activity  {  protected override void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  System.Threading.Thread.Sleep(1);  StartActivity(typeof(MainActivity));  }  }  } | Acá ponemos true porque ponemos false en el MainActivity  Tiempo que lo ponemos a “dormir”  Con StartActivity se le da el control al MainActivity |

En **MainActivity** cambiamos

|  |  |
| --- | --- |
| **MainActivity** | **Comentarios** |
| namespace MyLeasing.Prism.Droid  {  [Activity(Label = " namespace MyLeasing.Prism.Droid  {  [Activity(Label = "Taxi",  Icon = "@mipmap/ic\_launcher",  Theme = "@style/MainTheme",  MainLauncher = false,  ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)] ",  Icon = "@mipmap/ic\_launcher",  Theme = "@style/MainTheme",  MainLauncher = false,  ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)] | Ponemos false porque ponemos true en el SplashActivity |

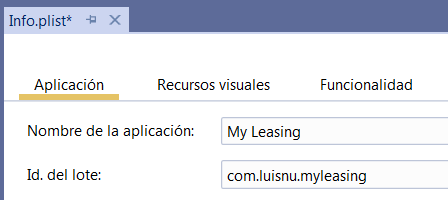
Para el ícono vamos al **Android** **Asset** **Studio** a la opción **Launcher** **icon** **Generator**

Bajamos el ZIP, lo descomprimimos e incorporamos los archivos a las carpetas mimaps

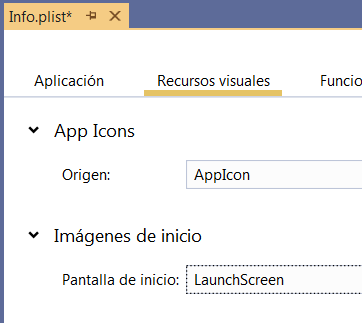
Por otro lado en el archivo Properties de Android, en la opción Manifiesto cambiamos el nombre de la App

# Splash e ícono en iOS

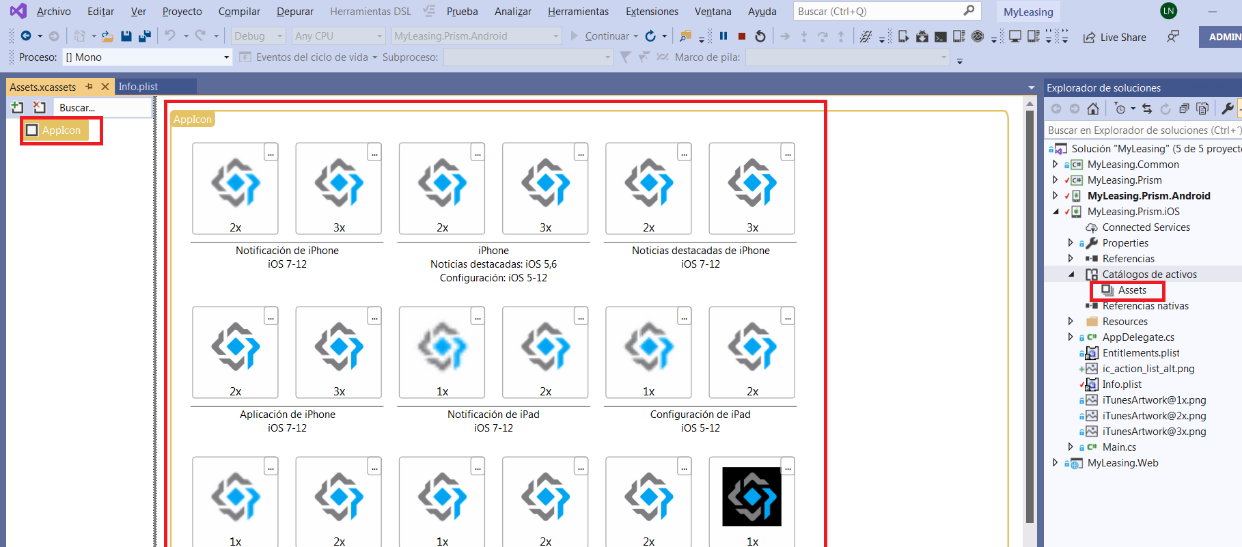
En Info.plist cambiamos el nombre de la Aplicación



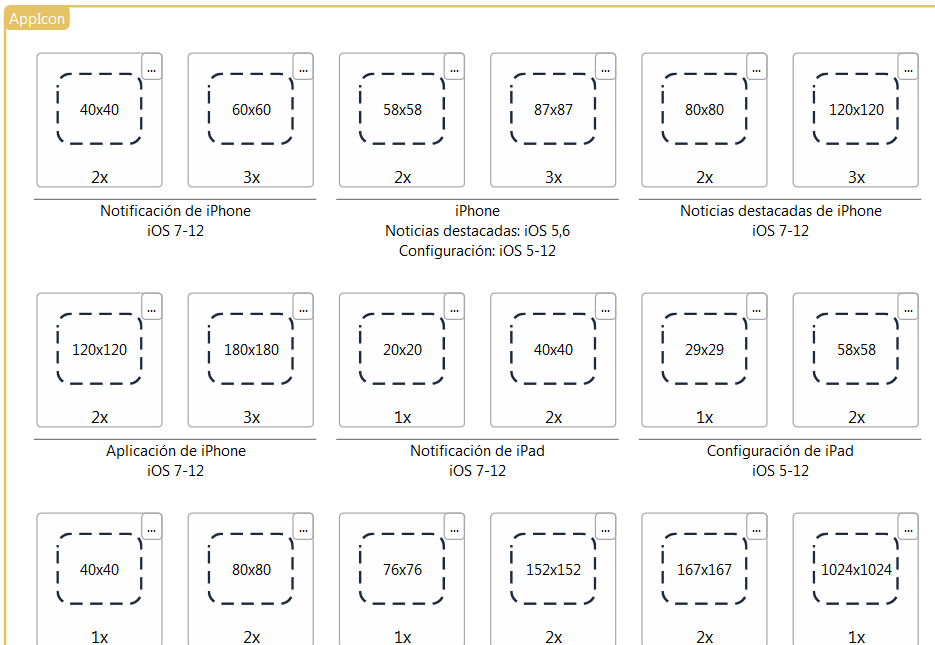
En Recursos visuales ponemos AppIcon y LaunchScreen



En iOS/Catálogos de activos/Assets tenemos:



Haciendo clic derecho Anular en cada uno, aparecen los tamaños exactos que deben tener:



Los íconos en iOS no pueden ser transparentes. Partamos de tener una imagen en JPG NO TRANSPARENTE.

Vamos a la página web <https://makeappicon.com/> . Cargo el archivo JPG, la pagina me genera los íconos y me los envía por mail.

Cargo cada jpg. Para el splash hace falta tener una Mac.

# Navegar a otra página y pasar parámetros

En la Tabla Trips en el Campo Remarks ponemos unos Lorem Ipsum

Luego en **TaxiHistoryPage** agregamos al campo Remarks:

<Label Grid.Column="2"

Text="{Binding Remarks}"

MaxLines="2"

LineBreakMode="TailTruncation"

VerticalOptions="Center"/>

Creamos en P**rism/ViewModels** la Clase **TripItemViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class TripItemViewModel : TripResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectTripCommand;

public TripItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectTripCommand => \_selectTripCommand ?? (\_selectTripCommand = new DelegateCommand(SelectTripAsync));

private async void SelectTripAsync()

{

await \_navigationService.NavigateAsync("TripDetailPage");

}

}

}

Creamos la **TripDetailPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TripDetailPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos la **TripDetailPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class TripDetailPageViewModel : ViewModelBase

{

public TripDetailPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Detalle de un viaje";

}

}

}

Modificamos la **TaxiHistoryPage**

<CollectionView ItemsSource="{Binding Trips}">

<CollectionView.ItemsLayout>

<GridItemsLayout Orientation="Vertical"/>

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectTripCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width=".8\*" />

<ColumnDefinition Width=".5\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

Modificamos la **TaxiHistoryPageViewModel**

using Prism.Commands;

using Prism.Navigation;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using Taxi.Common.Models;

using Taxi.Common.Services;

namespace Taxi.Prism.ViewModels

{

public class TaxiHistoryPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private TaxiResponse \_taxi;

private DelegateCommand \_checkPlaqueCommand;

private bool \_isRunning;

private List<TripItemViewModel> \_trips;

public List<TripItemViewModel> Trips

{

get => \_trips;

set => SetProperty(ref \_trips, value);

}

public TaxiHistoryPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Historia del Taxi";

}

public TaxiResponse Taxi

{

get => \_taxi;

set => SetProperty(ref \_taxi, value);

}

public string Plaque { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public DelegateCommand CheckPlaqueCommand => \_checkPlaqueCommand ?? (\_checkPlaqueCommand = new DelegateCommand(CheckPlaqueAsync));

private async void CheckPlaqueAsync()

{

if (string.IsNullOrEmpty(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Patente.",

"Aceptar");

return;

}

Regex regex = new Regex(@"^([A-Za-z]{3}\d{3})$");

if (!regex.IsMatch(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"La Patente debe empezar con 3 letras y finalizar con 3 números.",

"Accept");

return;

}

IsRunning = true;

var url = App.Current.Resources["UrlAPI"].ToString();

var connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

await App.Current.MainPage.DisplayAlert("Error", "Chequee su conexión a Internet.", "Accept");

return;

}

Response response = await \_apiService.GetTaxiAsync(Plaque, url, "api", "/Taxis");

IsRunning = false;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

return;

}

Taxi = (TaxiResponse)response.Result;

Trips = Taxi.Trips.Select(t => new TripItemViewModel(\_navigationService)

{

EndDate = t.EndDate,

Id = t.Id,

Qualification = t.Qualification,

Remarks = t.Remarks,

Source = t.Source,

SourceLatitude = t.SourceLatitude,

SourceLongitude = t.SourceLongitude,

StartDate = t.StartDate,

Target = t.Target,

TargetLatitude = t.TargetLatitude,

TargetLongitude = t.TargetLongitude,

TripDetails = t.TripDetails,

User = t.User

}).ToList();

}

}

}

Modificamos **TripDetailPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:rating="clr-namespace:Syncfusion.SfRating.XForms;assembly=Syncfusion.SfRating.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TripDetailPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<StackLayout Padding="5">

<rating:SfRating HorizontalOptions="Center"

IsEnabled="False"

Margin="0,5"

Precision="Exact"

Value="{Binding Trip.Qualification}">

<rating:SfRating.RatingSettings>

<rating:SfRatingSettings RatedFill="{StaticResource ColorDanger}"/>

</rating:SfRating.RatingSettings>

</rating:SfRating>

<StackLayout HorizontalOptions="Center"

Orientation="Horizontal">

<Label FontAttributes="Bold"

FontSize="Large"

Text="Calificación"/>

<Label FontAttributes="Bold"

FontSize="Large"

Text="{Binding Trip.Qualification, StringFormat='{0:N2}'}"/>

</StackLayout>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

FontAttributes="Bold"

Text="Fecha Inicio:"/>

<Label Grid.Column="1"

Grid.Row="0"

Text="{Binding Trip.StartDateLocal, StringFormat='{0:yyyy/MM/dd}'}"/>

<Label Grid.Column="0"

Grid.Row="1"

LineBreakMode="CharacterWrap"

FontAttributes="Bold"

Text="Comentarios:"/>

<Label Grid.Column="1"

Grid.Row="1"

Text="{Binding Trip.Remarks}"/>

</Grid>

</StackLayout>

</ContentPage>

Modificamos la **TripItemViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Models;

using Taxi.Prism.Views;

namespace Taxi.Prism.ViewModels

{

public class TripItemViewModel : TripResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectTripCommand;

public TripItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectTripCommand => \_selectTripCommand ?? (\_selectTripCommand = new DelegateCommand(SelectTripAsync));

private async void SelectTripAsync()

{

NavigationParameters parameters = new NavigationParameters

{

{ "trip", this }

};

await \_navigationService.NavigateAsync(nameof(TripDetailPage), parameters);

}

}

}

Modificamos la **TripDetailPageViewModel**

using Prism.Navigation;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class TripDetailPageViewModel : ViewModelBase

{

private TripResponse \_trip;

public TripDetailPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Detalle de un viaje";

}

public TripResponse Trip

{

get => \_trip;

set => SetProperty(ref \_trip, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("trip"))

{

Trip = parameters.GetValue<TripResponse>("trip");

}

}

}

}

# Area Segura en iOS

Agregar en todas las Vistas

ios:Page.UseSafeArea=”true”

# Redireccionar Paginas

En **AccountController** creamos el método **NotAuthorized**

public IActionResult NotAuthorized()

{

return View();

}

Creamos la Vista **NotAuthorized**

@{

ViewData["Title"] = "NotAuthorized";

}

<br />

<br />

<img src="~/images/error404.png" />

<h2>No está autorizado para esta opción!!</h2>

Modificamos el **StartUp.cs**

services.ConfigureApplicationCookie(options =>

{

options.LoginPath = "/Account/NotAuthorized";

options.AccessDeniedPath = "/Account/NotAuthorized";

});

Más abajo tambièn agregamos:

app.UseStatusCodePagesWithReExecute("/error/{0}");

Dentro de **HomeController** agregamos

[Route("error/404")]

public IActionResult Error404()

{

return View();

}

Cremos la Vista **Error404**

@{

ViewData["Title"] = "Error404";

}

<br />

<br />

<img src="~/images/error404.png" />

<h2>Página no encontrada</h2>

Importamos la imagen **error404.png**

# Autoregistro de Usuarios

Creamos en **Web/Models** la **EditUserViewModel**

using Microsoft.AspNetCore.Http;

using System.ComponentModel.DataAnnotations;

namespace Taxi.Web.Models

{

public class EditUserViewModel

{

public int Id { get; set; }

[Display(Name = "Documento")]

[MaxLength(20, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Document { get; set; }

[Display(Name = "Nombre")]

[MaxLength(50, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string FirstName { get; set; }

[Display(Name = "Apellido")]

[MaxLength(50, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string LastName { get; set; }

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

public string Address { get; set; }

[Display(Name = "Teléfono")]

[MaxLength(50, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

public string PhoneNumber { get; set; }

[Display(Name = "Foto")]

public IFormFile PictureFile { get; set; }

[Display(Name = "Ruta Foto")]

public string PicturePath { get; set; }

}

}

Creamos en **Web/Models** la **AddUserViewModel**

using Microsoft.AspNetCore.Mvc.Rendering;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Taxi.Common.Enums;

namespace Taxi.Web.Models

{

public class AddUserViewModel : EditUserViewModel

{

[Display(Name = "Email")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[MaxLength(100, ErrorMessage = "El campo {0} no puede tener más de {1} caracteres.")]

[EmailAddress]

public string Username { get; set; }

[Display(Name = "Password")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

public string Password { get; set; }

[Display(Name = "Confirm. Password")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

[Compare("Password")]

public string PasswordConfirm { get; set; }

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[Display(Name = "Registrar como")]

[Range(1, int.MaxValue, ErrorMessage = "Debe seleecionar un Rol")]

public int UserTypeId { get; set; }

public IEnumerable<SelectListItem> UserTypes { get; set; }

}

}

Agregamos una carpeta **Users** en **wwwroot/images**

Agregamos en **IUserHelper**

Task<UserEntity> AddUserAsync(AddUserViewModel model, string path);

Agregamos en **UserHelper**

public async Task<UserEntity> AddUserAsync(AddUserViewModel model, string path)

{

UserEntity userEntity = new UserEntity

{

Address = model.Address,

Document = model.Document,

Email = model.Username,

FirstName = model.FirstName,

LastName = model.LastName,

PicturePath = path,

PhoneNumber = model.PhoneNumber,

UserName = model.Username,

UserType = model.UserTypeId == 1 ? UserType.Driver : UserType.User

};

IdentityResult result = await \_userManager.CreateAsync(userEntity, model.Password);

if (result != IdentityResult.Success)

{

return null;

}

UserEntity newUser = await GetUserByEmailAsync(model.Username);

await AddUserToRoleAsync(newUser, userEntity.UserType.ToString());

return newUser;

}

Agregamos la Interfaz **ICombosHelper**

using Microsoft.AspNetCore.Mvc.Rendering;

using System.Collections.Generic;

namespace Taxi.Web.Helpers

{

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboRoles();

}

}

Agregamos la Clase **CombosHelper**

using Microsoft.AspNetCore.Mvc.Rendering;

using System.Collections.Generic;

namespace Taxi.Web.Helpers

{

public class CombosHelper : ICombosHelper

{

public IEnumerable<SelectListItem> GetComboRoles()

{

List<SelectListItem> list = new List<SelectListItem>

{

new SelectListItem { Value = "0", Text = "[Seleccione un Rol...]" },

new SelectListItem { Value = "1", Text = "Conductor" },

new SelectListItem { Value = "2", Text = "Usuario" }

};

return list;

}

}

}

Agregamos la Interfaz **IImageHelper**

using Microsoft.AspNetCore.Http;

using System.Threading.Tasks;

namespace Taxi.Web.Helpers

{

public interface IImageHelper

{

Task<string> UploadImageAsync(IFormFile imageFile, string folder);

}

}

Agregamos la Clase **ImageHelper**

using Microsoft.AspNetCore.Http;

using System;

using System.IO;

using System.Threading.Tasks;

namespace Taxi.Web.Helpers

{

public class ImageHelper : IImageHelper

{

public async Task<string> UploadImageAsync(IFormFile imageFile, string folder)

{

string guid = Guid.NewGuid().ToString();

string file = $"{guid}.jpg";

string path = Path.Combine(

Directory.GetCurrentDirectory(),

$"wwwroot\\images\\{folder}",

file);

using (FileStream stream = new FileStream(path, FileMode.Create))

{

await imageFile.CopyToAsync(stream);

}

return $"~/images/{folder}/{file}";

}

}

}

En **StartUp** inyectamos las nuevas interfaces

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>();

services.AddScoped<IConverterHelper, ConverterHelper>();

services.AddScoped<ICombosHelper, CombosHelper>();

services.AddScoped<IImageHelper, ImageHelper>();

En **AccountController** inyectamos **IcombosHelper** y **IImageHelper**

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

private readonly IImageHelper \_imageHelper;

public AccountController(IUserHelper userHelper, ICombosHelper combosHelper,IImageHelper imageHelper)

{

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_imageHelper = imageHelper;

}

Agregamos en **AccountController**

public IActionResult Register()

{

var model = new AddUserViewModel

{

UserTypes = \_combosHelper.GetComboRoles()

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(AddUserViewModel model)

{

if (ModelState.IsValid)

{

string path = string.Empty;

if (model.PictureFile != null)

{

path = await \_imageHelper.UploadImageAsync(model.PictureFile, "Users");

}

UserEntity user = await \_userHelper.AddUserAsync(model, path);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

model.UserTypes = \_combosHelper.GetComboRoles();

return View(model);

}

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

var response = \_mailHelper.SendMail(model.Username, "Email confirmation",

$"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

if (response.IsSuccess)

{

ViewBag.Message = "The instructions to allow your user has been sent to email.";

return View(model);

}

ModelState.AddModelError(string.Empty, response.Message);

}

model.UserTypes = \_combosHelper.GetComboRoles();

return View(model);

}

En **Account** agregamos la Vista Parcial **\_User**

@model Taxi.Web.Models.EditUserViewModel

<div class="form-group">

<label **asp-for**="Document" class="control-label"></label>

<input **asp-for**="Document" class="form-control" />

<span **asp-validation-for**="Document" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="FirstName" class="control-label"></label>

<input **asp-for**="FirstName" class="form-control" />

<span **asp-validation-for**="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="LastName" class="control-label"></label>

<input **asp-for**="LastName" class="form-control" />

<span **asp-validation-for**="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Address" class="control-label"></label>

<input **asp-for**="Address" class="form-control" />

<span **asp-validation-for**="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="PhoneNumber" class="control-label"></label>

<input **asp-for**="PhoneNumber" class="form-control" />

<span **asp-validation-for**="PhoneNumber" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="PictureFile" class="control-label"></label>

<input **asp-for**="PictureFile" class="form-control" **type**="file" />

<span **asp-validation-for**="PictureFile" class="text-danger"></span>

</div>

Agregamos la vista **Register**

@model Taxi.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Registrarse</h2>

<h4>User</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="Register" enctype="multipart/form-data">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Username" class="control-label"></label>

<input **asp-for**="Username" class="form-control" />

<span **asp-validation-for**="Username" class="text-danger"></span>

</div>

<**partial** **name**="\_User" />

<div class="form-group">

<label **asp-for**="UserTypeId" class="control-label"></label>

<select **asp-for**="UserTypeId" **asp-items**="Model.UserTypes" class="form-control"></select>

<span **asp-validation-for**="UserTypeId" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Password" class="control-label"></label>

<input **asp-for**="Password" class="form-control" />

<span **asp-validation-for**="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="PasswordConfirm" class="control-label"></label>

<input **asp-for**="PasswordConfirm" class="form-control" />

<span **asp-validation-for**="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Register" class="btn btn-primary" />

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

# Modificar Usuarios

Creamos la **ChangePasswordViewModel**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Web.Models

{

public class ChangePasswordViewModel

{

[Display(Name = "Password actual")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

public string OldPassword { get; set; }

[Display(Name = "Nuevo password")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

public string NewPassword { get; set; }

[Display(Name = "Confirm. Password")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

[Compare("NewPassword")]

public string Confirm { get; set; }

}

}

Agregamos al **IUserHelper**

Task<IdentityResult> ChangePasswordAsync(UserEntity user, string oldPassword, string newPassword);

Task<IdentityResult> UpdateUserAsync(UserEntity user);

Agregamos al **UserHelper**

public async Task<IdentityResult> ChangePasswordAsync(UserEntity user, string oldPassword, string newPassword)

{

return await \_userManager.ChangePasswordAsync(user, oldPassword, newPassword);

}

public async Task<IdentityResult> UpdateUserAsync(UserEntity user)

{

return await \_userManager.UpdateAsync(user);

}

Agregamos al **AccountController**

public async Task<IActionResult> ChangeUser()

{

UserEntity user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

if (user == null)

{

return NotFound();

}

EditUserViewModel model = new EditUserViewModel

{

Address = user.Address,

Document = user.Document,

FirstName = user.FirstName,

LastName = user.LastName,

PhoneNumber = user.PhoneNumber,

PicturePath = user.PicturePath

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangeUser(EditUserViewModel model)

{

if (ModelState.IsValid)

{

string path = model.PicturePath;

if (model.PictureFile != null)

{

path = await \_imageHelper.UploadImageAsync(model.PictureFile, "Users");

}

UserEntity user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

user.Document = model.Document;

user.FirstName = model.FirstName;

user.LastName = model.LastName;

user.Address = model.Address;

user.PhoneNumber = model.PhoneNumber;

user.PicturePath = path;

await \_userHelper.UpdateUserAsync(user);

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

}

return View(model);

}

Agregamos la vista **ChangeUser**

@model Taxi.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar Usuario</h2>

<h4></h4>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="ChangeUser" enctype="multipart/form-data">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **type**="hidden" **asp-for**="Id" />

<input **type**="hidden" **asp-for**="PicturePath" />

<**partial** **name**="\_User" />

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a **asp-action**="ChangePassword" class="btn btn-warning">Change Password</a>

</div>

</form>

</div>

<div class="col-md-4">

@if (!string.IsNullOrEmpty(Model.PicturePath))

{

<img src="@Url.Content(Model.PicturePath)" alt="Image" style="width:300px;height:300px;max-width: 100%; height: auto;" />

}

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Agregar al **AccountController**

public IActionResult ChangePassword()

{

return View();

}

[HttpPost]

public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

if (user != null)

{

var result = await \_userHelper.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

return RedirectToAction("ChangeUser");

}

else

{

ModelState.AddModelError(string.Empty, result.Errors.FirstOrDefault().Description);

}

}

else

{

ModelState.AddModelError(string.Empty, "User no found.");

}

}

return View(model);

}

Agregamos la vista **ChangePassword**

@model Taxi.Web.Models.ChangePasswordViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Cambiar Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div **asp-validation-summary**="ModelOnly"></div>

<div class="form-group">

<label **asp-for**="OldPassword">Current password</label>

<input **asp-for**="OldPassword" **type**="password" class="form-control" />

<span **asp-validation-for**="OldPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label **asp-for**="NewPassword">New password</label>

<input **asp-for**="NewPassword" **type**="password" class="form-control" />

<span **asp-validation-for**="NewPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label **asp-for**="Confirm">Confirm</label>

<input **asp-for**="Confirm" **type**="password" class="form-control" />

<span **asp-validation-for**="Confirm" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Change password" class="btn btn-primary" />

<a **asp-action**="ChangeUser" class="btn btn-success">Back to user</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

# Agregar API para Trips (Viajes)

Agregamos esta propiedad a **TripDetailEntity**

[MaxLength(500, ErrorMessage = "El campo {0} debe tener {1} caracteres.")]

public string Address { get; set; }

Agregamos a **TripDetailResponse**

public string Address { get; set; }

Modificamos en **TripEntity**

[Display(Name = "Origen")]

[MaxLength(500, ErrorMessage = "El campo {0} debe tener {1} caracteres.")]

public string Source { get; set; }

[Display(Name = "Destino")]

[MaxLength(500, ErrorMessage = "El campo {0} debe tener {1} caracteres.")]

public string Target { get; set; }

Ejecutamos en la Consola del Administrador de Paquetes:

Drop-database

add-migration TripEntityModified

update-database

En **IUserHelper** renombramos **GetUserByEmailAsync** por **GetUserAsync** (hacemos Control punto para renombrar en todo el Proyecto

Además le hacemos una sobrecarga

Task<UserEntity> GetUserAsync(Guid userId);

Y en **UserHelper** hacemos la implementacón

public async Task<UserEntity> GetUserAsync(Guid userId)

{

return await \_userManager.FindByIdAsync(userId.ToString());

}

Agregamos al **IConverterHelper**

TripResponse ToTripResponse(TripEntity tripEntity);

Agregamos al **ConverterHelper**

public TripResponse ToTripResponse(TripEntity tripEntity)

{

return new TripResponse

{

EndDate = tripEntity.EndDate,

Id = tripEntity.Id,

Qualification = tripEntity.Qualification,

Remarks = tripEntity.Remarks,

Source = tripEntity.Source,

SourceLatitude = tripEntity.SourceLatitude,

SourceLongitude = tripEntity.SourceLongitude,

StartDate = tripEntity.StartDate,

Target = tripEntity.Target,

TargetLatitude = tripEntity.TargetLatitude,

TargetLongitude = tripEntity.TargetLongitude,

TripDetails = tripEntity.TripDetails?.Select(td => new TripDetailResponse

{

Address = td.Address,

Date = td.Date,

Id = td.Id,

Latitude = td.Latitude,

Longitude = td.Longitude

}).ToList(),

User = ToUserResponse(tripEntity.User)

};

}

En **Common/Models** agregamos la clase **TripRequest**

using System;

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class TripRequest

{

[RegularExpression(@"^([A-Za-z]{3}\d{3})$", ErrorMessage = "El campo {0} debe tener 3 caracteres y 3 números.")]

[Required(ErrorMessage = "El campo{0} es obligatorio.")]

[StringLength(6, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener al menos {1} caracteres.")]

public string Plaque { get; set; }

[Required(ErrorMessage = "El campo{0} es obligatorio.")]

public Guid UserId { get; set; }

public string Address { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

}

}

Agregamos el Controlador API **TripsController**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Taxi.Common.Models;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

public class TripsController : ControllerBase

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly IConverterHelper \_converterHelper;

public TripsController(DataContext context, IUserHelper userHelper, IConverterHelper converterHelper)

{

\_context = context;

\_userHelper = userHelper;

\_converterHelper = converterHelper;

}

[HttpPost]

public async Task<IActionResult> PostTripEntity([FromBody] TripRequest tripRequest)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserEntity userEntity = await \_userHelper.GetUserAsync(tripRequest.UserId);

if (userEntity == null)

{

return BadRequest("User doesn't exists.");

}

TaxiEntity taxiEntity = await \_context.Taxis.FirstOrDefaultAsync(t => t.Plaque == tripRequest.Plaque);

if (taxiEntity == null)

{

\_context.Taxis.Add(new TaxiEntity { Plaque = tripRequest.Plaque.ToUpper() });

await \_context.SaveChangesAsync();

taxiEntity = await \_context.Taxis.FirstOrDefaultAsync(t => t.Plaque == tripRequest.Plaque);

}

TripEntity tripEntity = new TripEntity

{

Source = tripRequest.Address,

SourceLatitude = tripRequest.Latitude,

SourceLongitude = tripRequest.Longitude,

StartDate = DateTime.UtcNow,

Taxi = taxiEntity,

TripDetails = new List<TripDetailEntity>

{

new TripDetailEntity

{

Date = DateTime.UtcNow,

Latitude = tripRequest.Latitude,

Longitude = tripRequest.Longitude

}

},

User = userEntity,

};

\_context.Trips.Add(tripEntity);

await \_context.SaveChangesAsync();

return Ok(\_converterHelper.ToTripResponse(tripEntity));

}

}

}

# Agregar Generación de Tokens

Agregamos en **appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": { "DefaultConnection": "Server=keypress.serveftp.net;Database=LuisTaxi;User Id=sa;password=sentey14$;Trusted\_Connection=False;MultipleActiveResultSets=true" },

"Tokens": {

"Key": "asdfghjikbnvcgfdsrtfyhgcvgfxdgc",

"Issuer": "localhost",

"Audience": "users"

}

}

Agregamos al **IUserHelper**

Task<SignInResult> ValidatePasswordAsync(UserEntity user, string password);

Agregamos al **UserHelper**

public async Task<SignInResult> ValidatePasswordAsync(UserEntity user, string password)

{

return await \_signInManager.CheckPasswordSignInAsync(user, password, false);

}

Modificamos en **AccountController**

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

private readonly IImageHelper \_imageHelper;

private readonly IConfiguration \_configuration;

public AccountController(IUserHelper userHelper, ICombosHelper combosHelper,IImageHelper imageHelper,

IConfiguration configuration)

{

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_imageHelper = imageHelper;

\_configuration = configuration;

}

Agregamos en **AccountController**

[HttpPost]

public async Task<IActionResult> CreateToken([FromBody] LoginViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(model.Username);

if (user != null)

{

var result = await \_userHelper.ValidatePasswordAsync(user, model.Password);

if (result.Succeeded)

{

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.Email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Tokens:Key"]));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

\_configuration["Tokens:Issuer"],

\_configuration["Tokens:Audience"],

claims,

expires: DateTime.UtcNow.AddDays(15),

signingCredentials: credentials);

var results = new

{

token = new JwtSecurityTokenHandler().WriteToken(token),

expiration = token.ValidTo

};

return Created(string.Empty, results);

}

}

}

return BadRequest();

}

Agregamos en el API **TripsController**

[Route("api/[controller]")]

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

Agregamos en **StartUp**

}).AddEntityFrameworkStores<DataContext>();

services.AddAuthentication()

.AddCookie()

.AddJwtBearer(cfg =>

{

cfg.TokenValidationParameters = new TokenValidationParameters

{

ValidIssuer = Configuration["Tokens:Issuer"],

ValidAudience = Configuration["Tokens:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Tokens:Key"]))

};

});

# Confirmar Registro por Email

Agregamos en **Startup.cs** estas líneas

|  |  |
| --- | --- |
| **Startup.cs** | **Comentarios** |
| services.AddIdentity<User, IdentityRole>(cfg =>  {  cfg.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;  cfg.SignIn.RequireConfirmedEmail = true;  cfg.User.RequireUniqueEmail = true;  cfg.Password.RequireDigit = false;  cfg.Password.RequiredUniqueChars = 0;  cfg.Password.RequireLowercase = false;  cfg.Password.RequireNonAlphanumeric = false;  cfg.Password.RequireUppercase = false;  }).AddDefaultTokenProviders()  .AddEntityFrameworkStores<DataContext>(); |  |

Comprobar si la cuenta de correo electrónico está habilitada para enviar correos electrónicos en: <https://myaccount.google.com/lesssecureapps>

En **appsettings.json** agregar

|  |  |
| --- | --- |
| **appsettings.json** | **Comentarios** |
| {  "Logging": {  "LogLevel": {  "Default": "Warning"  }  },  "AllowedHosts": "\*",  "ConnectionStrings": {  "DefaultConnection": "Server=keypress.serveftp.net;Database=LuisSoccer;User Id=sa;password=sentey14$;Trusted\_Connection=False;MultipleActiveResultSets=true",  "Tokens": {  "Key": "asdfghjikbnvcgfdsrtfyhgcvgfxdgc",  "Issuer": "localhost",  "Audience": "users"  },  "Mail": {  "From": "luisalbertonu@gmail.com",  "Smtp": "smtp.gmail.com",  "Port": 587,  "Password": "Cuestablanca1972\*"  }  } |  |

Agregamos el Nugget **Mailkit** version 2.4.1 al Proyecto Web

Agregamos en **Helpers** la Interfaz **IMailHelper**

|  |  |
| --- | --- |
| **IMailHelper** | **Comentarios** |
| using Taxi.Common.Models;  namespace Taxi.Web.Helpers  {  public interface IMailHelper  {  Response SendMail(string to, string subject, string body);  }  } |  |

Agregamos en **Helpers** la Clase **MailHelper**

|  |  |
| --- | --- |
| **MailHelper** | **Comentarios** |
| using MailKit.Net.Smtp;  using Microsoft.Extensions.Configuration;  using MimeKit;  using Taxi.Common.Models;  using System;  namespace Taxi.Web.Helpers  {  public class MailHelper : IMailHelper  {  private readonly IConfiguration \_configuration;  public MailHelper(IConfiguration configuration)  {  \_configuration = configuration;  }  public Response SendMail(string to, string subject, string body)  {  try  {  string from = \_configuration["Mail:From"];  string smtp = \_configuration["Mail:Smtp"];  string port = \_configuration["Mail:Port"];  string password = \_configuration["Mail:Password"];  MimeMessage message = new MimeMessage();  message.From.Add(new MailboxAddress(from));  message.To.Add(new MailboxAddress(to));  message.Subject = subject;  BodyBuilder bodyBuilder = new BodyBuilder  {  HtmlBody = body  };  message.Body = bodyBuilder.ToMessageBody();  using (SmtpClient client = new SmtpClient())  {  client.Connect(smtp, int.Parse(port), false);  client.Authenticate(from, password);  client.Send(message);  client.Disconnect(true);  }  return new Response { IsSuccess = true };  }  catch (Exception ex)  {  return new Response  {  IsSuccess = false,  Message = ex.Message,  Result = ex  };  }  }  }  } |  |

En **Startup.cs** agregamos la línea:

|  |  |
| --- | --- |
| **Startup.cs** | **Comentarios** |
| services.AddTransient<SeedDb>();  services.AddScoped<IMailHelper, MailHelper>();  services.AddScoped<IUserHelper, UserHelper>();  services.AddScoped<ICombosHelper, CombosHelper>();  services.AddScoped<IConverterHelper, ConverterHelper>();  services.AddScoped<IImageHelper, ImageHelper>();  services.AddDbContext<DataContext>(cfg => |  |

Agregamos al **IUserHelper**

|  |  |
| --- | --- |
| **IUserHelper** | **Comentarios** |
| Task<string> GenerateEmailConfirmationTokenAsync(UserEntity user);  Task<IdentityResult> ConfirmEmailAsync(UserEntity user, string token); |  |

Agregamos al **UserHelper**

|  |  |
| --- | --- |
| **UserHelper** | **Comentarios** |
| public async Task<IdentityResult> ConfirmEmailAsync(UserEntity user, string token)  {  return await \_userManager.ConfirmEmailAsync(user, token);  }  public async Task<string> GenerateEmailConfirmationTokenAsync(UserEntity user)  {  return await \_userManager.GenerateEmailConfirmationTokenAsync(user);  } |  |

En **AccountController** agregamos:

|  |  |
| --- | --- |
| **AccountController** | **Comentarios** |
| public class AccountController : Controller  {  private readonly DataContext \_dataContext;  private readonly IUserHelper \_userHelper;  private readonly IConfiguration \_configuration;  private readonly ICombosHelper \_combosHelper;  private readonly IMailHelper \_mailHelper;  public AccountController(DataContext dataContext, IUserHelper userHelper,  IConfiguration configuration, ICombosHelper combosHelper,  IMailHelper mailHelper)  {  \_dataContext = dataContext;  \_userHelper = userHelper;  \_configuration = configuration;  \_combosHelper = combosHelper;  \_mailHelper = mailHelper;  } |  |

Modificamos el método **Register**

|  |  |
| --- | --- |
| **Register** | **Comentarios** |
| [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Register(AddUserViewModel model)  {  if (ModelState.IsValid)  {  string path = string.Empty;  if (model.PictureFile != null)  {  path = await \_imageHelper.UploadImageAsync(model.PictureFile, "Users");  }  UserEntity user = await \_userHelper.AddUserAsync(model, path, UserType.User);  if (user == null)  {  ModelState.AddModelError(string.Empty, "This email is already used.");  model.UserTypes = \_combosHelper.GetComboRoles();  return View(model);  }  var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);  var tokenLink = Url.Action("ConfirmEmail", "Account", new  {  userid = user.Id,  token = myToken  }, protocol: HttpContext.Request.Scheme);  var response = \_mailHelper.SendMail(model.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +  $"To allow the user, " +  $"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");  if (response.IsSuccess)  {  ViewBag.Message = "The instructions to allow your user has been sent to email.";  return View(model);  }  ModelState.AddModelError(string.Empty, response.Message);  }  model.UserTypes = \_combosHelper.GetComboRoles();  return View(model);  } |  |

En la Vista **Register** agregamos:

|  |  |
| --- | --- |
| **Register** | **Comentarios** |
| <div class="form-group">  <input type="submit" value="Register" class="btn btn-primary" />  </div>  </form>  </div>  </div>  <div class="text-success">  <p>  @ViewBag.Message  </p>  </div>  @section Scripts {  @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}  } |  |

En **AccountController** agregamos:

|  |  |
| --- | --- |
| **AccountController** | **Comentarios** |
| public async Task<IActionResult> ConfirmEmail(string userId, string token)  {  if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(token))  {  return NotFound();  }  UserEntity user = await \_userHelper.GetUserAsync(new Guid(userId));  if (user == null)  {  return NotFound();  }  Microsoft.AspNetCore.Identity.IdentityResult result = await \_userHelper.ConfirmEmailAsync(user, token);  if (!result.Succeeded)  {  return NotFound();  }  return View();  } |  |

Creamos la Vista **ConfirmEmail**

|  |  |
| --- | --- |
| **ConfirmEmail** | **Comentarios** |
| @{  ViewData["Title"] = "Confirm email";  }  <h2>@ViewData["Title"]</h2>  <div>  <p>  Gracias por confirmar su email. Ahora puede loguearse al Sistema.  </p>  </div> |  |

Ejecutamos en la Consola del Administrador de Paquetes:

Drop-database

Modificamos el método **CheckUserAsync** en **SeedDb**

private async Task<UserEntity> CheckUserAsync(

string document,

string firstName,

string lastName,

string email,

string phone,

string address,

UserType userType)

{

var user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

user = new UserEntity

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

UserType = userType

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

var token = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

# Password Recovery

En la vista de Login agregamos

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a **asp-action**="Register" class="btn btn-primary">Registrar Nuevo Usuario</a>

<a **asp-action**="RecoverPassword" class="btn btn-link">Olvidó su Contraseña?</a>

</div>

Agregamos en **Models** la Clase **RecoverPasswordViewModel**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Web.Models

{

public class RecoverPasswordViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

Agregamos en **Models** la Clase **ResetPasswordViewModel**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Web.Models

{

public class ResetPasswordViewModel

{

[Required]

public string UserName { get; set; }

[Required]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe tener entre {2} y {1} caracteres.")]

[DataType(DataType.Password)]

[Compare("Password")]

public string ConfirmPassword { get; set; }

[Required]

public string Token { get; set; }

}

}

Agregamos al **IUserHelper**

Task<string> GeneratePasswordResetTokenAsync(UserEntity user);

Task<IdentityResult> ResetPasswordAsync(UserEntity user, string token, string password);

Agregamos al **UserHelper**

public async Task<string> GeneratePasswordResetTokenAsync(UserEntity user)

{

return await \_userManager.GeneratePasswordResetTokenAsync(user);

}

public async Task<IdentityResult> ResetPasswordAsync(UserEntity user, string token, string password)

{

return await \_userManager.ResetPasswordAsync(user, token, password);

}

Agregamos al **AccountController**

public IActionResult RecoverPassword()

{

return View();

}

[HttpPost]

public async Task<IActionResult> RecoverPassword(RecoverPasswordViewModel model)

{

if (ModelState.IsValid)

{

UserEntity user = await \_userHelper.GetUserAsync(model.Email);

if (user == null)

{

ModelState.AddModelError(string.Empty, "El email no corresponde a un usuario registrado.");

return View(model);

}

string myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

string link = Url.Action(

"ResetPassword",

"Account",

new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(model.Email, "Taxi Reseteo de Password", $"<h1>Taxi Reseteo de Password</h1>" +

$"Para resetear el password haga clic en este link: </br></br>" +

$"<a href = \"{link}\">Resetear Password</a>");

ViewBag.Message = "Las instrucciones para recuperar su password han sido enviadas por mail.";

return View();

}

return View(model);

}

public IActionResult ResetPassword(string token)

{

return View();

}

[HttpPost]

public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)

{

UserEntity user = await \_userHelper.GetUserAsync(model.UserName);

if (user != null)

{

Microsoft.AspNetCore.Identity.IdentityResult result = await \_userHelper.ResetPasswordAsync(user, model.Token, model.Password);

if (result.Succeeded)

{

ViewBag.Message = "Password reseteado correctamente.";

return View();

}

ViewBag.Message = "Error mientras se reseteaba el password.";

return View(model);

}

ViewBag.Message = "Usuario no encontrado.";

return View(model);

}

Agregamos la vista **RecoverPassword**

@model Taxi.Web.Models.RecoverPasswordViewModel

@{

ViewData["Title"] = "Recuperar Password";

}

<h2>Recuperar Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div **asp-validation-summary**="ModelOnly"></div>

<div class="form-group">

<label **asp-for**="Email">Email</label>

<input **asp-for**="Email" class="form-control" />

<span **asp-validation-for**="Email" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Recuperar Password" class="btn btn-primary" />

<a **asp-action**="Login" class="btn btn-success">Regresar</a>

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Agregamos la vista **ResetPassword**

@model Taxi.Web.Models.ResetPasswordViewModel

@{

ViewData["Title"] = "Reset Password";

}

<h1>Resetear Password</h1>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div **asp-validation-summary**="All"></div>

<input **type**="hidden" **asp-for**="Token" />

<div class="form-group">

<label **asp-for**="UserName">Email</label>

<input **asp-for**="UserName" class="form-control" />

<span **asp-validation-for**="UserName" class="text-warning"></span>

</div>

<div class="form-group">

<label **asp-for**="Password">New password</label>

<input **asp-for**="Password" **type**="password" class="form-control" />

<span **asp-validation-for**="Password" class="text-warning"></span>

</div>

<div class="form-group">

<label **asp-for**="ConfirmPassword">Confirm</label>

<input **asp-for**="ConfirmPassword" **type**="password" class="form-control" />

<span **asp-validation-for**="ConfirmPassword" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Resetear password" class="btn btn-primary" />

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

# API para registrar nuevos usuarios

Agregamos en **Common/Models** el modelo **UserRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class UserRequest

{

[Required]

public string Document { get; set; }

[Required]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

[Required]

public string Address { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Phone { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string Password { get; set; }

[Required]

public int UserTypeId { get; set; } // 1: User, 2: Driver

public byte[] PictureArray { get; set; }

public string PasswordConfirm { get; set; }

}

}

Agregamos a **IImageHelper**

string UploadImage(byte[] pictureArray, string folder);

Agregamos a **ImageHelper**

public string UploadImage(byte[] pictureArray, string folder)

{

MemoryStream stream = new MemoryStream(pictureArray);

string guid = Guid.NewGuid().ToString();

string file = $"{guid}.jpg";

try

{

stream.Position = 0;

string path = Path.Combine(Directory.GetCurrentDirectory(), $"wwwroot\\images\\{folder}", file);

File.WriteAllBytes(path, stream.ToArray());

}

catch

{

return string.Empty;

}

return $"~/images/{folder}/{file}";

}

Agregamos el Controlador API **AccountController**

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Common.Enums;

using Taxi.Common.Models;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Controllers.API

{

[Route("api/[Controller]")]

public class AccountController : ControllerBase

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

private readonly IImageHelper \_imageHelper;

public AccountController(

DataContext dataContext,

IUserHelper userHelper,

IMailHelper mailHelper,

IImageHelper imageHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

\_imageHelper = imageHelper;

}

[HttpPost]

public async Task<IActionResult> PostUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request"

});

}

UserEntity user = await \_userHelper.GetUserAsync(request.Email);

if (user != null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Este mail ya está registrado."

});

}

string picturePath = string.Empty;

if (request.PictureArray != null && request.PictureArray.Length > 0)

{

picturePath = \_imageHelper.UploadImage(request.PictureArray, "Users");

}

user = new UserEntity

{

Address = request.Address,

Document = request.Document,

Email = request.Email,

FirstName = request.FirstName,

LastName = request.LastName,

PhoneNumber = request.Phone,

UserName = request.Email,

PicturePath = picturePath,

UserType = request.UserTypeId == 1 ? UserType.User : UserType.Driver

};

IdentityResult result = await \_userHelper.AddUserAsync(user, request.Password);

if (result != IdentityResult.Success)

{

return BadRequest(result.Errors.FirstOrDefault().Description);

}

UserEntity userNew = await \_userHelper.GetUserAsync(request.Email);

await \_userHelper.AddUserToRoleAsync(userNew, user.UserType.ToString());

string myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

string tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Confirmación de Email", $"<h1>Confirmación de Email</h1>" +

$"Para habilitar el usuario, " +

$"haga clic en este link: </br></br><a href = \"{tokenLink}\">Confirmar Email</a>");

return Ok(new Response

{

IsSuccess = true,

Message = "Un mail de confirmación fue enviado. Ingrese a su cuenta para confirmar su mail e ingrese a la App."

});

}

}

}

# API para recuperar Password

Creamos el modelo **EmailRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class EmailRequest

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

Agregamos el método **RecoverPassword** a **AccountController API**

[HttpPost]

[Route("RecoverPassword")]

public async Task<IActionResult> RecoverPassword([FromBody] EmailRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserAsync(request.Email);

if (user == null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Este mail no está asignado a ningún usuario."

});

}

var myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

var link = Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Resetear Password", $"<h1>Resetear Password</h1>" +

$"Para resetear el password haga clic en este link: </br></br>" +

$"<a href = \"{link}\">Resetear Password</a>");

return Ok(new Response

{

IsSuccess = true,

Message = "Se ha enviado un mail con instrucciones para cambiar el password."

});

}

# API para modificar usuario

En **AccountController** **API** agregamos el método **PutUser**

[HttpPut]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> PutUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserEntity userEntity = await \_userHelper.GetUserAsync(request.Email);

if (userEntity == null)

{

return BadRequest("Usuario no encontrado");

}

string picturePath = userEntity.PicturePath;

if (request.PictureArray != null && request.PictureArray.Length > 0)

{

picturePath = \_imageHelper.UploadImage(request.PictureArray, "Users");

}

userEntity.FirstName = request.FirstName;

userEntity.LastName = request.LastName;

userEntity.Address = request.Address;

userEntity.PhoneNumber = request.Phone;

userEntity.Document = request.Document;

userEntity.PicturePath = picturePath;

IdentityResult respose = await \_userHelper.UpdateUserAsync(userEntity);

if (!respose.Succeeded)

{

return BadRequest(respose.Errors.FirstOrDefault().Description);

}

UserEntity updatedUser = await \_userHelper.GetUserAsync(request.Email);

return Ok(updatedUser);

}

# API para modificar password

Agregamos el modelo **ChangePasswordRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class ChangePasswordRequest

{

[Required]

[StringLength(20, MinimumLength = 6)]

public string OldPassword { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string NewPassword { get; set; }

[Required]

public string Email { get; set; }

}

}

Agregamos al **AccountController** **API** el método **ChangePassword**

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPost]

[Route("ChangePassword")]

public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request",

Result = ModelState

});

}

UserEntity user = await \_userHelper.GetUserAsync(request.Email);

if (user == null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Usuario no encontrado"

});

}

IdentityResult result = await \_userHelper.ChangePasswordAsync(user, request.OldPassword, request.NewPassword);

if (!result.Succeeded)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = result.Errors.FirstOrDefault().Description

});

}

return Ok(new Response

{

IsSuccess = true,

Message = "El passwrod fue cambiado con éxito."

});

}

# API para GetuserByEmail

Agregamos en **IConverterHelper**

UserResponse ToUserResponse(UserEntity user);

Modificamos en **ConverterHelper**

public UserResponse ToUserResponse(UserEntity user)

En el **AccountController** **API** agregamos el método **GetUserByEmail**

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPost]

[Route("GetUserByEmail")]

public async Task<IActionResult> GetUserByEmail([FromBody] EmailRequest emailRequest)

{

if (!ModelState.IsValid)

{

return BadRequest();

}

UserEntity userEntity = await \_userHelper.GetUserAsync(emailRequest.Email);

if (userEntity == null)

{

return NotFound("Error002");

}

return Ok(\_converterHelper.ToUserResponse(userEntity));

}

# API para completar viajes

En **Common**/**Models** agregamos **TripDetailRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class TripDetailRequest

{

[Required]

public int TripId { get; set; }

[MaxLength(500, ErrorMessage = "El campo {0} debe tener al menos {1} caracteres.")]

public string Address { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

}

}

Agregamos la Clase **TripDetailsRequest**

using System.Collections.Generic;

namespace Taxi.Common.Models

{

public class TripDetailsRequest

{

public List<TripDetailRequest> TripDetails { get; set; }

}

}

Agregamos la Clase **CompleteTripRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class CompleteTripRequest

{

[Required]

public int TripId { get; set; }

[MaxLength(500, ErrorMessage = "El campo {0} debe tener al menos {1} caracteres.")]

public string Target { get; set; }

public double TargetLatitude { get; set; }

public double TargetLongitude { get; set; }

public float Qualification { get; set; }

public string Remarks { get; set; }

}

}

Agregamos la Clase **MyTripsRequest**

using System;

namespace Taxi.Common.Models

{

public class MyTripsRequest

{

public string UserId { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

}

}

Agregamos la Clase **IncidentRequest**

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class IncidentRequest : TripRequest

{

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Remarks { get; set; }

}

}

En **TripsController** **API** agregamos estos métodos

[HttpPost]

[Route("CompleteTrip")]

public async Task<IActionResult> CompleteTrip([FromBody] CompleteTripRequest completeTripRequest)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

TripEntity trip = await \_context.Trips

.Include(t => t.TripDetails)

.FirstOrDefaultAsync(t => t.Id == completeTripRequest.TripId);

if (trip == null)

{

return BadRequest("Viaje no encontrado");

}

trip.EndDate = DateTime.UtcNow;

trip.Qualification = completeTripRequest.Qualification;

trip.Remarks = completeTripRequest.Remarks;

trip.Target = completeTripRequest.Target;

trip.TargetLatitude = completeTripRequest.TargetLatitude;

trip.TargetLongitude = completeTripRequest.TargetLongitude;

trip.TripDetails.Add(new TripDetailEntity

{

Date = DateTime.UtcNow,

Latitude = completeTripRequest.TargetLatitude,

Longitude = completeTripRequest.TargetLongitude

});

\_context.Trips.Update(trip);

await \_context.SaveChangesAsync();

return NoContent();

}

# Setup multiples enviroments en Postman

# GetTripEntity

[HttpGet("{id}")]

public async Task<IActionResult> GetTripEntity([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

TripEntity tripEntity = await \_context.Trips

.Include(t => t.TripDetails)

.FirstOrDefaultAsync(t => t.Id == id);

if (tripEntity == null)

{

return BadRequest("Viaje no encontrado");

}

return Ok(\_converterHelper.ToTripResponse(tripEntity));

}

# DeleteTrip

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteTripEntity([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var tripEntity = await \_context.Trips

.Include(t => t.TripDetails)

.FirstOrDefaultAsync(t => t.Id == id);

if (tripEntity == null)

{

return NotFound();

}

\_context.Trips.Remove(tripEntity);

await \_context.SaveChangesAsync();

return NoContent();

}

# AddTripDetails

[HttpPost]

[Route("AddTripDetails")]

public async Task<IActionResult> AddTripDetails([FromBody] TripDetailsRequest tripDetailsRequest)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (tripDetailsRequest.TripDetails == null || tripDetailsRequest.TripDetails.Count == 0)

{

return NoContent();

}

TripEntity trip = await \_context.Trips

.Include(t => t.TripDetails)

.FirstOrDefaultAsync(t => t.Id == tripDetailsRequest.TripDetails.FirstOrDefault().TripId);

if (trip == null)

{

return BadRequest("Viaje no encontrado");

}

if (trip.TripDetails == null)

{

trip.TripDetails = new List<TripDetailEntity>();

}

foreach (TripDetailRequest tripDetailRequest in tripDetailsRequest.TripDetails)

{

trip.TripDetails.Add(new TripDetailEntity

{

Date = DateTime.UtcNow,

Latitude = tripDetailRequest.Latitude,

Longitude = tripDetailRequest.Longitude

});

}

\_context.Trips.Update(trip);

await \_context.SaveChangesAsync();

return NoContent();

}

# Métodos para Trip

Agregamos en **Common/Models** la Clase **TripResponseWithTaxi**

namespace Taxi.Common.Models

{

public class TripResponseWithTaxi : TripResponse

{

public TaxiResponse Taxi { get; set; }

}

}

Agregamos al **IConverterHelper**

List<TripResponseWithTaxi> ToTripResponse(List<TripEntity> tripEntities);

Agregamos al **ConverterHelper**

public List<TripResponseWithTaxi> ToTripResponse(List<TripEntity> tripEntities)

{

return tripEntities.Select(t => new TripResponseWithTaxi

{

EndDate = t.EndDate,

Id = t.Id,

Qualification = t.Qualification,

Remarks = t.Remarks,

Source = t.Source,

SourceLatitude = t.SourceLatitude,

SourceLongitude = t.SourceLongitude,

StartDate = t.StartDate,

Target = t.Target,

Taxi = ToTaxiResponse2(t.Taxi),

TargetLatitude = t.TargetLatitude,

TargetLongitude = t.TargetLongitude,

TripDetails = t.TripDetails.Select(td => new TripDetailResponse

{

Address = td.Address,

Date = td.Date,

Id = td.Id,

Latitude = td.Latitude,

Longitude = td.Longitude

}).ToList()

}).ToList();

}

private TaxiResponse ToTaxiResponse2(TaxiEntity taxi)

{

return new TaxiResponse

{

Id = taxi.Id,

Plaque = taxi.Plaque,

User = ToUserResponse(taxi.User)

};

}

Agregamos al Controlador **TripsController API**

[HttpPost]

[Route("GetMyTrips")]

public async Task<IActionResult> GetMyTrips([FromBody] MyTripsRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var tripEntities = await \_context.Trips

.Include(t => t.User)

.Include(t => t.TripDetails)

.Include(t => t.Taxi)

.Where(t => t.User.Id == request.UserId &&

t.StartDate >= request.StartDate &&

t.StartDate <= request.EndDate)

.OrderByDescending(t => t.StartDate)

.ToListAsync();

return Ok(\_converterHelper.ToTripResponse(tripEntities));

}

Agregamos al Controlador **TripsController API**

[HttpPost]

[Route("AddIncident")]

public async Task<IActionResult> AddIncident([FromBody] IncidentRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserEntity userEntity = await \_userHelper.GetUserAsync(request.UserId);

if (userEntity == null)

{

return BadRequest("Este Usuario no existe.");

}

TaxiEntity taxiEntity = await \_context.Taxis.FirstOrDefaultAsync(t => t.Plaque == request.Plaque);

if (taxiEntity == null)

{

\_context.Taxis.Add(new TaxiEntity { Plaque = request.Plaque.ToUpper() });

await \_context.SaveChangesAsync();

taxiEntity = await \_context.Taxis.FirstOrDefaultAsync(t => t.Plaque == request.Plaque);

}

TripEntity tripEntity = new TripEntity

{

Source = request.Address,

SourceLatitude = request.Latitude,

SourceLongitude = request.Longitude,

StartDate = DateTime.UtcNow,

Taxi = taxiEntity,

EndDate = DateTime.UtcNow,

Qualification = 1,

Remarks = request.Remarks,

Target = request.Address,

TargetLatitude = request.Latitude,

TargetLongitude = request.Longitude,

TripDetails = new List<TripDetailEntity>

{

new TripDetailEntity

{

Date = DateTime.UtcNow,

Latitude = request.Latitude,

Longitude = request.Longitude

},

new TripDetailEntity

{

Date = DateTime.UtcNow,

Latitude = request.Latitude,

Longitude = request.Longitude

}

},

User = userEntity,

};

\_context.Trips.Add(tripEntity);

await \_context.SaveChangesAsync();

return NoContent();

}

# Fix User Groups Tables

Agregamos la Entity **UserGroupDetailEntity**

namespace Taxi.Web.Data.Entities

{

public class UserGroupDetailEntity

{

public int Id { get; set; }

public UserEntity User { get; set; }

public UserGroupEntity UserGroup { get; set; }

}

}

Agregamos la Entity **UserGroupRequestEntity**

using System.Collections.Generic;

namespace Taxi.Web.Data.Entities

{

public class UserGroupEntity

{

public int Id { get; set; }

public UserEntity User { get; set; }

public ICollection<UserGroupDetailEntity> Users { get; set; }

}

}

En **Common/Enums** agregamos **UserGroupStatus**

namespace Taxi.Common.Enums

{

public enum UserGroupStatus

{

Pending,

Accepted,

Rejected

}

}

Agregamos la Entity **UserGroupRequestEntity**

using System;

using Taxi.Common.Enums;

namespace Taxi.Web.Data.Entities

{

public class UserGroupRequestEntity

{

public int Id { get; set; }

public UserEntity ProposalUser { get; set; }

public UserEntity RequiredUser { get; set; }

public UserGroupStatus Status { get; set; }

public Guid Token { get; set; }

}

}

Modificamos **DataContext**

public DbSet<UserGroupDetailEntity> UserGroupDetails { get; set; }

public DbSet<UserGroupRequestEntity> UserGroupRequests { get; set; }

Modificamos el **SeedDb**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Common.Enums;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Data

{

public class SeedDb

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

public SeedDb(

DataContext dataContext,

IUserHelper userHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_dataContext.Database.EnsureCreatedAsync();

await CheckRolesAsync();

var admin = await CheckUserAsync("17157729", "Luis", "Nuñez", "luisalbertonu@gmail.com", "156814963", "Espora 2052", UserType.Admin);

var driver = await CheckUserAsync("17000001", "Pablo", "Lacuadri", "lacua@yopmail.com", "350 634 2747", "Villa Santa Ana 123", UserType.Driver);

var user1 = await CheckUserAsync("17000002", "Diego", "Maradona", "maradona@yopmail.com", "350 634 2747", "Villa Fiorito 234", UserType.User);

var user2 = await CheckUserAsync("17000003", "Lionel", "Messi", "messi@yopmail.com", "350 634 2747", "Barcelona 345", UserType.User);

var user3 = await CheckUserAsync("17000004", "Mario", "Kempes", "kempes@yopmail.com", "350 634 2747", "Alta Cordoba", UserType.User);

var user4 = await CheckUserAsync("17000005", "Gabriel", "Batistuta", "batistuta@yopmail.com", "350 634 2747", "Florencia 123", UserType.User);

await CheckTaxisAsync(driver, user1, user2);

await CheckUserGroups(user1, user2,user3,user4);

}

private async Task CheckUserGroups(UserEntity user1, UserEntity user2, UserEntity user3, UserEntity user4)

{

if (!\_dataContext.UserGroups.Any())

{

\_dataContext.UserGroups.Add(new UserGroupEntity

{

User = user1,

Users = new List<UserGroupDetailEntity>

{

new UserGroupDetailEntity { User = user2 },

new UserGroupDetailEntity { User = user3 },

new UserGroupDetailEntity { User = user4 }

}

});

\_dataContext.UserGroups.Add(new UserGroupEntity

{

User = user2,

Users = new List<UserGroupDetailEntity>

{

new UserGroupDetailEntity { User = user1 },

new UserGroupDetailEntity { User = user3 },

new UserGroupDetailEntity { User = user4 }

}

});

await \_dataContext.SaveChangesAsync();

}

}

private async Task<UserEntity> CheckUserAsync(

string document,

string firstName,

string lastName,

string email,

string phone,

string address,

UserType userType)

{

var user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

user = new UserEntity

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

UserType = userType

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

var token = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

private async Task CheckRolesAsync()

{

await \_userHelper.CheckRoleAsync(UserType.Admin.ToString());

await \_userHelper.CheckRoleAsync(UserType.Driver.ToString());

await \_userHelper.CheckRoleAsync(UserType.User.ToString());

}

private async Task CheckTaxisAsync(

UserEntity driver,

UserEntity user1,

UserEntity user2)

{

if (!\_dataContext.Taxis.Any())

{

\_dataContext.Taxis.Add(new TaxiEntity

{

User = driver,

Plaque = "IJX537",

Trips = new List<TripEntity>

{

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.5f,

Source = "Barrio Rosedal",

Target = "Barrio General Paz",

Remarks = "Es un hecho establecido hace demasiado tiempo que un lector se distraerá con el contenido del texto de un sitio mientras que mira su diseño. El punto de usar Lorem Ipsum es que tiene una distribución más o menos normal de las letras, al contrario de usar textos como por ejemplo. Estos textos hacen parecerlo un español que se puede leer. Muchos paquetes de autoedición y editores de páginas web usan el Lorem Ipsum como su texto por defecto, y al hacer una búsqueda de Lorem Ipsum va a dar por resultado muchos sitios web que usan este texto si se encuentran en estado de desarrollo. Muchas versiones han evolucionado a través de los años, algunas veces por accidente, otras veces a propósito (por ejemplo insertándole humor y cosas por el estilo).",

User = user1

},

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.8f,

Source = "Barrio General Paz",

Target = "Barrio Rosedal",

Remarks = "Conductor muy amable",

User = user1

}

}

});

\_dataContext.Taxis.Add(new TaxiEntity

{

Plaque = "PJM791",

Trips = new List<TripEntity>

{

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.5f,

Source = "Barrio Rosedal",

Target = "Centro",

Remarks = "Hay muchas variaciones de los pasajes de Lorem Ipsum disponibles, pero la mayoría sufrió alteraciones en alguna manera, ya sea porque se le agregó humor, o palabras aleatorias que no parecen ni un poco creíbles. Si vas a utilizar un pasaje de Lorem Ipsum, necesitás estar seguro de que no hay nada avergonzante escondido en el medio del texto. Todos los generadores de Lorem Ipsum que se encuentran en Internet tienden a repetir trozos predefinidos cuando sea necesario, haciendo a este el único generador verdadero (válido) en la Internet. Usa un diccionario de mas de 200 palabras provenientes del latín, combinadas con estructuras muy útiles de sentencias, para generar texto de Lorem Ipsum que parezca razonable. Este Lorem Ipsum generado siempre estará libre de repeticiones, humor agregado o palabras no características del lenguaje, etc.",

User = user2

},

new TripEntity

{

StartDate = DateTime.UtcNow,

EndDate = DateTime.UtcNow.AddMinutes(30),

Qualification = 4.8f,

Source = "Centro",

Target = "Barrio Rosedal",

Remarks = "Conductor muy amable",

User = user2

}

}

});

await \_dataContext.SaveChangesAsync();

}

}

}

}

En la **Consola del Administrador de Paquetes** hacemos:

add-migration AddUserGroups

drop-database

update-database

# API to get the user group from a user

Creamos en **Common/Models** la clase **UserGroupDetailResponse**

namespace Taxi.Common.Models

{

public class UserGroupDetailResponse

{

public int Id { get; set; }

public UserResponse User { get; set; }

}

}

Creamos en **Common/Models** la clase **UserGroupResponse**

using System.Collections.Generic;

namespace Taxi.Common.Models

{

public class UserGroupResponse

{

public int Id { get; set; }

public UserResponse User { get; set; }

public List<UserGroupDetailResponse> Users { get; set; }

}

}

Agregamos en **IconverterHelper**

List<UserGroupDetailResponse> ToUserGroupResponse(List<UserGroupDetailEntity> users);

Agregamos en **ConverterHelper**

public List<UserGroupDetailResponse> ToUserGroupResponse(List<UserGroupDetailEntity> users)

{

return users.Select(u => new UserGroupDetailResponse

{

Id = u.Id,

User = ToUserResponse(u.User)

}).ToList();

}

Creamos el Controlador **API** **UserGroupsController**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Controllers.API

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class UserGroupsController : ControllerBase

{

private readonly DataContext \_context;

private readonly IConverterHelper \_converterHelper;

public UserGroupsController(

DataContext context,

IConverterHelper converterHelper)

{

\_context = context;

\_converterHelper = converterHelper;

}

[HttpGet("{id}")]

public async Task<IActionResult> GetUserGroup([FromRoute] string id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserGroupEntity userGroup = await \_context.UserGroups

.Include(ug => ug.Users)

.ThenInclude(u => u.User)

.FirstOrDefaultAsync(u => u.User.Id == id);

if (userGroup == null || userGroup?.Users == null)

{

return Ok();

}

return Ok(\_converterHelper.ToUserGroupResponse(userGroup.Users.ToList()));

}

}

}

Creamos en **Common/Models** la clase **AddUserGroupRequest**

using System;

using System.ComponentModel.DataAnnotations;

namespace Taxi.Common.Models

{

public class AddUserGroupRequest

{

[Required]

public Guid UserId { get; set; }

[EmailAddress]

[Required]

public string Email { get; set; }

}

}

Modificamos **UserGroupsController**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Common.Enums;

using Taxi.Common.Models;

using Taxi.Web.Data;

using Taxi.Web.Data.Entities;

using Taxi.Web.Helpers;

namespace Taxi.Web.Controllers.API

{

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class UserGroupsController : ControllerBase

{

private readonly DataContext \_context;

private readonly IConverterHelper \_converterHelper;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

public UserGroupsController(

DataContext context,

IConverterHelper converterHelper,

IUserHelper userHelper,

IMailHelper mailHelper)

{

\_context = context;

\_converterHelper = converterHelper;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

}

[HttpGet("{id}")]

public async Task<IActionResult> GetUserGroup([FromRoute] string id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserGroupEntity userGroup = await \_context.UserGroups

.Include(ug => ug.Users)

.ThenInclude(u => u.User)

.FirstOrDefaultAsync(u => u.User.Id == id);

if (userGroup == null || userGroup?.Users == null)

{

return Ok();

}

return Ok(\_converterHelper.ToUserGroupResponse(userGroup.Users.ToList()));

}

[HttpPost]

public async Task<IActionResult> PostUserGroup([FromBody] AddUserGroupRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

UserEntity proposalUser = await \_userHelper.GetUserAsync(request.UserId);

if (proposalUser == null)

{

return BadRequest("Este Usuario no existe.");

}

UserEntity requiredUser = await \_userHelper.GetUserAsync(request.Email);

if (requiredUser == null)

{

return BadRequest("Este Usuario no existe.");

}

UserGroupEntity userGroup = await \_context.UserGroups

.Include(ug => ug.Users)

.ThenInclude(u => u.User)

.FirstOrDefaultAsync(ug => ug.User.Id == request.UserId.ToString());

if (userGroup != null)

{

UserGroupDetailEntity user = userGroup.Users.FirstOrDefault(u => u.User.Email == request.Email);

if (user != null)

{

return BadRequest("Este Usuario ya pertenece al Grupo.");

}

}

UserGroupRequestEntity userGroupRequest = new UserGroupRequestEntity

{

ProposalUser = proposalUser,

RequiredUser = requiredUser,

Status = UserGroupStatus.Pending,

Token = Guid.NewGuid()

};

try

{

\_context.UserGroupRequests.Add(userGroupRequest);

await \_context.SaveChangesAsync();

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

string linkConfirm = Url.Action("ConfirmUserGroup", "Account", new

{

requestId = userGroupRequest.Id,

token = userGroupRequest.Token

}, protocol: HttpContext.Request.Scheme);

string linkReject = Url.Action("RejectUserGroup", "Account", new

{

requestId = userGroupRequest.Id,

token = userGroupRequest.Token

}, protocol: HttpContext.Request.Scheme);

Response response = \_mailHelper.SendMail(request.Email, "Solicitud de unirse a un Grupo", $"<h1>Solicitud de unirse a un Grupo</h1>" +

$"El Usuario: {proposalUser.FullName} ({proposalUser.Email}), ha solicitado que sea miembro de su grupo de usuarios en la aplicación TAXI." +

$"</hr></br></br>Si desea aceptar, haga clic aquí: <a href = \"{linkConfirm}\">Confirmar</a>" +

$"</hr></br></br>Si desea rechazar, haga clic aquí: <a href = \"{linkReject}\">Rechazar</a>");

if (!response.IsSuccess)

{

return BadRequest(response.Message);

}

return Ok("Se ha enviado un correo electrónico al usuario con su solicitud, esperamos a que responda pronto!");

}

}

}

Modificamos el **AccountController**

using Microsoft.AspNetCore.Mvc;

using Taxi.Web.Helpers;

using Taxi.Web.Models;

using System.Linq;

using System.Threading.Tasks;

using Taxi.Web.Data.Entities;

using Microsoft.Extensions.Configuration;

using System.Security.Claims;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.IdentityModel.Tokens;

using System;

using System.Text;

using Taxi.Common.Enums;

using Taxi.Web.Data;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

namespace Taxi.Web.Controllers

{

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

private readonly IImageHelper \_imageHelper;

private readonly IConfiguration \_configuration;

private readonly IMailHelper \_mailHelper;

private readonly DataContext \_context;

public AccountController(IUserHelper userHelper, ICombosHelper combosHelper,IImageHelper imageHelper,

IConfiguration configuration,IMailHelper mailHelper,DataContext context)

{

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_imageHelper = imageHelper;

\_configuration = configuration;

\_mailHelper = mailHelper;

\_context = context;

}

……………

public async Task<IActionResult> ConfirmUserGroup(int requestId, string token)

{

if (requestId == 0 || string.IsNullOrEmpty(token))

{

return NotFound();

}

UserGroupRequestEntity userGroupRequest = await \_context.UserGroupRequests

.Include(ugr => ugr.ProposalUser)

.Include(ugr => ugr.RequiredUser)

.FirstOrDefaultAsync(ugr => ugr.Id == requestId &&

ugr.Token == new Guid(token));

if (userGroupRequest == null)

{

return NotFound();

}

await AddGroupAsync(userGroupRequest.ProposalUser, userGroupRequest.RequiredUser);

await AddGroupAsync(userGroupRequest.RequiredUser, userGroupRequest.ProposalUser);

userGroupRequest.Status = UserGroupStatus.Accepted;

\_context.UserGroupRequests.Update(userGroupRequest);

await \_context.SaveChangesAsync();

return View();

}

private async Task AddGroupAsync(UserEntity proposalUser, UserEntity requiredUser)

{

UserGroupEntity userGroup = await \_context.UserGroups

.Include(ug => ug.Users)

.ThenInclude(u => u.User)

.FirstOrDefaultAsync(ug => ug.User.Id == proposalUser.Id);

if (userGroup != null)

{

UserGroupDetailEntity user = userGroup.Users.FirstOrDefault(u => u.User.Id == requiredUser.Id);

if (user == null)

{

userGroup.Users.Add(new UserGroupDetailEntity { User = requiredUser });

}

\_context.UserGroups.Update(userGroup);

}

else

{

\_context.UserGroups.Add(new UserGroupEntity

{

User = proposalUser,

Users = new List<UserGroupDetailEntity>

{

new UserGroupDetailEntity { User = requiredUser }

}

});

}

}

public async Task<IActionResult> RejectUserGroup(int requestId, string token)

{

if (requestId == 0 || string.IsNullOrEmpty(token))

{

return NotFound();

}

UserGroupRequestEntity userGroupRequest = await \_context

.UserGroupRequests.FirstOrDefaultAsync(ugr => ugr.Id == requestId &&

ugr.Token == new Guid(token));

if (userGroupRequest == null)

{

return NotFound();

}

userGroupRequest.Status = UserGroupStatus.Rejected;

\_context.UserGroupRequests.Update(userGroupRequest);

await \_context.SaveChangesAsync();

return View();

}

Agregamos la vista **ConfirmUserGroup**

@{

ViewData["Title"] = "Confirmar Usuario de un Grupo";

}

<h2>@ViewData["Title"]</h2>

<div>

<p>

Gracias por confirmar el Usuario. Ahora es parte de un Grupo y puede ver los viajes de los otros integrantes en tiempo real.

</p>

</div>

Agregamos la vista **RejectUserGroup**

@{

ViewData["Title"] = "Rechazo Usuario de Grupo";

}

<h2>@ViewData["Title"]</h2>

<div>

<p>

Gracias por contestar. Tal vez en otra ocasión puedas integrar el Grupo.

</p>

</div>

**Publicamos el API**

# Arreglos en iOS

En AppDelegate ponemos:

global::Xamarin.Forms.Forms.Init();

Xamarin.FormsMaps.Init();

Para que no se solape el Logo con la muesca que tienen los IPhone, hacemos en **TaxiMasterDetailPage**

(es agregar un Padding acorde a la plataforma)

<ContentPage.Padding>

<OnPlatform x:TypeArguments="Thickness">

<On Platform="Android, UWP">0</On>

<On Platform="iOS">0,20,0,0</On>

</OnPlatform>

</ContentPage.Padding>

# Login

Agregamos en **App.xaml**

<Style TargetType="Entry">

<Setter Property="BackgroundColor" Value="{StaticResource ColorFontInverse}" />

</Style>

<Style TargetType="Picker">

<Setter Property="BackgroundColor" Value="{StaticResource ColorFontInverse}" />

</Style>

Modificamos la **LoginPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:i18n="clr-namespace:Taxi.Prism.Helpers"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.LoginPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Image HeightRequest="150"

Margin="0,20,0,0"

Source="taxi"/>

<Grid Margin="20"

VerticalOptions="CenterAndExpand">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

HorizontalTextAlignment="End"

Text="Email:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="0"

BackgroundColor="{StaticResource ColorFontInverse}"

Keyboard="Email"

Placeholder="Ingrese Enail..."

Text="{Binding Email}"/>

<Label Grid.Column="0"

Grid.Row="1"

HorizontalTextAlignment="End"

Text="Password:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="1"

BackgroundColor="{StaticResource ColorFontInverse}"

IsPassword="True"

Placeholder="Ingrese Password..."

Text="{Binding Password}"/>

</Grid>

<StackLayout>

<Button Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="LogIn"/>

<Button Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Registrarse"

Style="{StaticResource DangerButton}"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos la **LoginPageViewModel**

using Prism.Commands;

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class LoginPageViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private string \_password;

private DelegateCommand \_loginCommand;

private DelegateCommand \_registerCommand;

public LoginPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Login";

IsEnabled = true;

}

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(LoginAsync));

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(RegisterAsync));

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public string Email { get; set; }

public string Password

{

get => \_password;

set => SetProperty(ref \_password, value);

}

private async void LoginAsync()

{

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Email",

"Aceptar");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Password",

"Aceptar");

return;

}

}

private void RegisterAsync()

{

}

}

}

En **Common/Models** agregamos la Clase **TokenRequest**

namespace Taxi.Common.Models

{

public class TokenRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

}

En **Common/Models** agregamos la Clase **TokenResponse**

using System;

namespace Taxi.Common.Models

{

public class TokenResponse

{

public string Token { get; set; }

public DateTime Expiration { get; set; }

public DateTime ExpirationLocal => Expiration.ToLocalTime();

}

}

Agregamos a **IapiService**

Task<Response> GetTokenAsync(string urlBase, string servicePrefix, string controller, TokenRequest request);

Task<Response> GetUserByEmail(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, EmailRequest request);

Agregamos a **ApiService**

public async Task<Response> GetTokenAsync(string urlBase, string servicePrefix, string controller, TokenRequest request)

{

try

{

string requestString = JsonConvert.SerializeObject(request);

StringContent content = new StringContent(requestString, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string result = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(result);

return new Response

{

IsSuccess = true,

Result = token

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

public async Task<Response> GetUserByEmail(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

EmailRequest request)

{

try

{

string requestString = JsonConvert.SerializeObject(request);

StringContent content = new StringContent(requestString, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string result = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

UserResponse userResponse = JsonConvert.DeserializeObject<UserResponse>(result);

return new Response

{

IsSuccess = true,

Result = userResponse

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

Agregamos el Nugget **Xam.Plugins.Settings** en el Proyecto **Common**

En **Common** creamos una carpeta llamada **Helpers**

En **Common/Helpers** creamos la Clase **Settings**

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace Taxi.Common.Helpers

{

public static class Settings

{

private const string \_user = "user";

private const string \_token = "token";

private const string \_isLogin = "isLogin";

private static readonly string \_stringDefault = string.Empty;

private static readonly bool \_boolDefault = false;

private static ISettings AppSettings => CrossSettings.Current;

public static string User

{

get => AppSettings.GetValueOrDefault(\_user, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_user, value);

}

public static string Token

{

get => AppSettings.GetValueOrDefault(\_token, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_token, value);

}

public static bool IsLogin

{

get => AppSettings.GetValueOrDefault(\_isLogin, \_boolDefault);

set => AppSettings.AddOrUpdateValue(\_isLogin, value);

}

}

}

Modificamos la **LoginPageViewModel**

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

namespace Taxi.Prism.ViewModels

{

public class LoginPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private string \_password;

private DelegateCommand \_loginCommand;

private DelegateCommand \_registerCommand;

public LoginPageViewModel(INavigationService navigationService, IApiService apiServiceService) : base(navigationService)

{

Title = "Login";

IsEnabled = true;

\_navigationService = navigationService;

\_apiService = apiServiceService;

}

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(LoginAsync));

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(RegisterAsync));

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public string Email { get; set; }

public string Password

{

get => \_password;

set => SetProperty(ref \_password, value);

}

private async void LoginAsync()

{

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Email",

"Aceptar");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Password",

"Aceptar");

return;

}

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Chequee su conexión a Internet",

"Aceptar");

return;

}

TokenRequest request = new TokenRequest

{

Password = Password,

Username = Email

};

Response response = await \_apiService.GetTokenAsync(url, "Account", "/CreateToken", request);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Usuario o Password incorrrectos",

"Aceptar");

Password = string.Empty;

return;

}

TokenResponse token = (TokenResponse)response.Result;

EmailRequest emailRequest = new EmailRequest

{

Email = Email

};

Response response2 = await \_apiService.GetUserByEmail(url, "api", "/Account/GetUserByEmail", "bearer", token.Token, emailRequest);

UserResponse userResponse = (UserResponse)response2.Result;

Settings.User = JsonConvert.SerializeObject(userResponse);

Settings.Token = JsonConvert.SerializeObject(token);

Settings.IsLogin = true;

IsRunning = false;

IsEnabled = true;

await \_navigationService.NavigateAsync("/TaxiMasterDetailPage/NavigationPage/HomePage");

Password = string.Empty;

}

private void RegisterAsync()

{

}

}

}

Modificamos la **TaxiMasterDetailPage**

<?xml version="1.0" encoding="utf-8" ?>

<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiMasterDetailPage">

<MasterDetailPage.Master>

<ContentPage Title="Menu"

BackgroundColor="{StaticResource ColorBackground}">

<StackLayout Padding="20">

<Image HeightRequest="150"

Source="taxi"/>

<Label FontAttributes="Bold"

FontSize="Large"

Text="{Binding User.FullName}"

TextColor="Black"/>

Modificamos la **TaxiMasterDetailPageViewModel**

using Newtonsoft.Json;

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class TaxiMasterDetailPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private UserResponse \_user;

public TaxiMasterDetailPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

LoadUser();

LoadMenus();

}

public ObservableCollection<MenuItemViewModel> Menus { get; set; }

public UserResponse User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

private void LoadUser()

{

if (Settings.IsLogin)

{

User = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

}

}

private void LoadMenus()

{

List<Menu> menus = new List<Menu>

{

new Menu

{

Icon = "ic\_action\_airport\_shuttle",

PageName = "HomePage",

Title = "New trip"

},

new Menu

{

Icon = "ic\_action\_local\_taxi",

PageName = "TaxiHistoryPage",

Title = "See taxi history"

},

new Menu

{

Icon = "ic\_action\_people",

PageName = "GroupPage",

Title = "Admin my user group"

},

new Menu

{

Icon = "ic\_action\_account\_circle",

PageName = "ModifyUserPage",

Title = "Modify User"

},

new Menu

{

Icon = "ic\_action\_report",

PageName = "ReportPage",

Title = "Report an incident"

},

new Menu

{

Icon = "ic\_action\_exit\_to\_app",

PageName = "LoginPage",

Title = Settings.IsLogin ? "Logout" : "Login"

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title

}).ToList());

}

}

}

Modificamos la **MenuItemViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

namespace Taxi.Prism.ViewModels

{

public class MenuItemViewModel : Menu

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectMenuCommand;

public MenuItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectMenuCommand => \_selectMenuCommand ?? (\_selectMenuCommand = new DelegateCommand(SelectMenuAsync));

private async void SelectMenuAsync()

{

if (PageName == "LoginPage" && Settings.IsLogin)

{

Settings.IsLogin = false;

Settings.User = null;

Settings.Token = null;

}

await \_navigationService.NavigateAsync($"/TaxiMasterDetailPage/NavigationPage/{PageName}");

}

}

}

Agregamos los Nugget **Xamarin.FFImageLoading.Forms** y **Xamarin.FFImageLoading.Transformations** a todos los proyectos Prism

En el **MainActivity** de Andorid ponemos:

base.OnCreate(bundle);

CrossCurrentActivity.Current.Init(this, bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfBusyIndicatorRenderer();

En el **AppDelegate** de iOS ponemos:

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

new SfBusyIndicatorRenderer();

new SfRatingRenderer();

LoadApplication(new App(new iOSInitializer()));

En **UserResponse** agregamos

public string PictureFullPath => string.IsNullOrEmpty(PicturePath)

? "noimage"//null

: $"http://keypress.serveftp.net:88/TaxiApi{PicturePath.Substring(1)}";

Modificamos la **TaxiMasterDetailPage**

<?xml version="1.0" encoding="utf-8" ?>

<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

xmlns:fftransformations="clr-namespace:FFImageLoading.Transformations;assembly=FFImageLoading.Transformations"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.TaxiMasterDetailPage">

<MasterDetailPage.Master>

<ContentPage Title="Menu"

BackgroundColor="{StaticResource ColorBackground}">

<StackLayout Padding="20">

<RelativeLayout>

<Image x:Name="Logo" HorizontalOptions="Center"

RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToParent,Property=X,Factor=1,Constant=60}"

HeightRequest="150"

Source="taxi"/>

<ffimageloading:CachedImage x:Name="Picture"

RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=Logo,Property=Y,Factor=1,Constant=30}"

Aspect="AspectFit"

Source="{Binding User.PictureFullPath}"

CacheDuration= "50"

HeightRequest="100"

Margin="5"

RetryCount= "3"

RetryDelay= "600">

<ffimageloading:CachedImage.Transformations>

<fftransformations:CircleTransformation />

</ffimageloading:CachedImage.Transformations>

</ffimageloading:CachedImage>

<Label RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=Picture,Property=Y,Factor=1,Constant=110}"

FontAttributes="Bold"

FontSize="Large"

Text="{Binding User.FullName}"

TextColor="Black"/>

</RelativeLayout>

<ListView BackgroundColor="Transparent"

ItemsSource="{Binding Menus}"

HasUnevenRows="True"

SeparatorVisibility="None">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectMenuCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image Grid.Column="0"

HeightRequest="40"

Source="{Binding Icon}"

WidthRequest="40"/>

<Label Grid.Column="1"

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

</MasterDetailPage.Master>

</MasterDetailPage>

# Register New User from App

Agregamos la **RegisterPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.RegisterPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos la **RegisterPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

public RegisterPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Registro de Nuevo Usuario";

}

}

}

Modificamos la **LoginPageViewModel**

private async void RegisterAsync()

{

await \_navigationService.NavigateAsync(nameof(RegisterPage));

}

En **Common/Helpers** agregamos la Clase **IRegexHelper**

namespace Taxi.Common.Helpers

{

public interface IRegexHelper

{

bool IsValidEmail(string emailaddress);

}

}

En **Common/Helpers** agregamos la Clase **RegexHelper**

using System;

using System.Net.Mail;

namespace Taxi.Common.Helpers

{

public class RegexHelper : IRegexHelper

{

public bool IsValidEmail(string emailaddress)

{

try

{

new MailAddress(emailaddress);

return true;

}

catch (FormatException)

{

return false;

}

}

}

}

Agregamos la inyección en **App.xmls.cs**

containerRegistry.Register<IRegexHelper, RegexHelper>();

Agregamos en **Common/Models** la Clase **Role**

namespace Taxi.Common.Models

{

public class Role

{

public int Id { get; set; }

public string Name { get; set; }

}

}

En **Taxi.Prism** creamos una carpeta **Helpers**

Dentro creamos la Clase **CombosHelper**

using System.Collections.Generic;

using Taxi.Common.Models;

namespace Taxi.Prism.Helpers

{

public static class CombosHelper

{

public static List<Role> GetRoles()

{

return new List<Role>

{

new Role { Id = 1, Name = "Usuario" },

new Role { Id = 2, Name = "Conductor" }

};

}

}

}

<x:String x:Key="UrlNoImage">noimage.png</x:String>

<Style TargetType="Entry">

<Setter Property="BackgroundColor" Value="{StaticResource ColorFontInverse}" />

</Style>

<Style TargetType="Picker">

<Setter Property="BackgroundColor" Value="{StaticResource ColorFontInverse}" />

</Style>

Modificamos la **RegisterPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:i18n="clr-namespace:Taxi.Prism.Helpers"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

xmlns:fftransformations="clr-namespace:FFImageLoading.Transformations;assembly=FFImageLoading.Transformations"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.RegisterPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<ScrollView>

<StackLayout>

<ffimageloading:CachedImage Aspect="AspectFit"

Source="{Binding Image}"

CacheDuration= "50"

HeightRequest="150"

Margin="5"

RetryCount= "3"

RetryDelay= "600">

<ffimageloading:CachedImage.Transformations>

<fftransformations:CircleTransformation />

</ffimageloading:CachedImage.Transformations>

</ffimageloading:CachedImage>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Row="0"

Grid.Column="0"

Text="Documento:"

VerticalOptions="Center"/>

<Entry Grid.Row="0"

Grid.Column="1"

Keyboard="Numeric"

Placeholder="Escriba Documento..."

Text="{Binding User.Document}"/>

<Label Grid.Row="1"

Grid.Column="0"

Text="Nombre:"

VerticalOptions="Center"/>

<Entry Grid.Row="1"

Grid.Column="1"

Placeholder="Escriba Nombre..."

Text="{Binding User.FirstName}"/>

<Label Grid.Row="2"

Grid.Column="0"

Text="Apellido:"

VerticalOptions="Center"/>

<Entry Grid.Row="2"

Grid.Column="1"

Placeholder="Escriba Apellido..."

Text="{Binding User.LastName}"/>

<Label Grid.Row="3"

Grid.Column="0"

Text="Domicilio:"

VerticalOptions="Center"/>

<Entry Grid.Row="3"

Grid.Column="1"

Placeholder="Escriba Domicilio..."

Text="{Binding User.Address}"/>

<Label Grid.Row="4"

Grid.Column="0"

Text="Email:"

VerticalOptions="Center"/>

<Entry Grid.Row="4"

Grid.Column="1"

Keyboard="Email"

Placeholder="Escriba Email..."

Text="{Binding User.Email}"/>

<Label Grid.Row="5"

Grid.Column="0"

Text="Teléfono:"

VerticalOptions="Center"/>

<Entry Grid.Row="5"

Grid.Column="1"

Keyboard="Telephone"

Placeholder="Escriba Teléfono..."

Text="{Binding User.Phone}"/>

<Label Grid.Row="6"

Grid.Column="0"

Text="Password:"

VerticalOptions="Center"/>

<Entry Grid.Row="6"

Grid.Column="1"

IsPassword="True"

Placeholder="Escriba Password..."

Text="{Binding User.Password}"/>

<Label Grid.Row="7"

Grid.Column="0"

Text="Confirm. Password:"

VerticalOptions="Center"/>

<Entry Grid.Row="7"

Grid.Column="1"

IsPassword="True"

Placeholder="Escriba Confirm. Password..."

Text="{Binding User.PasswordConfirm}"/>

<Label Grid.Row="8"

Grid.Column="0"

Text="Registrar como:"

VerticalOptions="Center"/>

<Picker Grid.Row="8"

Grid.Column="1"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding Roles}"

SelectedItem="{Binding Role}"

Title="Seleccione una opción...">

</Picker>

</Grid>

</StackLayout>

</ScrollView>

<Button Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Registrar"/>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos la **RegisterPageViewModel**

using Prism.Commands;

using Prism.Navigation;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Helpers;

using Xamarin.Forms;

namespace Taxi.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IRegexHelper \_regexHelper;

private readonly IApiService \_apiService;

private ImageSource \_image;

private UserRequest \_user;

private Role \_role;

private ObservableCollection<Role> \_roles;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

public RegisterPageViewModel(

INavigationService navigationService,

IRegexHelper regexHelper,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_regexHelper = regexHelper;

\_apiService = apiService;

Title = "Registro de Nuevo Usuario";

Image = App.Current.Resources["UrlNoImage"].ToString();

IsEnabled = true;

User = new UserRequest();

Roles = new ObservableCollection<Role>(CombosHelper.GetRoles());

}

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(RegisterAsync));

public ImageSource Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public UserRequest User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

public Role Role

{

get => \_role;

set => SetProperty(ref \_role, value);

}

public ObservableCollection<Role> Roles

{

get => \_roles;

set => SetProperty(ref \_roles, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void RegisterAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(User.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Documento",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Nombre",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Apellido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Domicilio",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Email) || !\_regexHelper.IsValidEmail(User.Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Email válido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Phone))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Teléfono",

"Aceptar");

return false;

}

if (Role == null)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe seleccionar un Rol",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Password) || User.Password?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Password",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Confirmación de Password",

"Aceptar");

return false;

}

if (User.Password != User.PasswordConfirm)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"El Password y su confirmación no coinciden",

"Aceptar");

return false;

}

return true;

}

}

}

Agregamos al **IApiService**

Task<Response> RegisterUserAsync(string urlBase, string servicePrefix, string controller, UserRequest userRequest);

Agregamos al **ApiService**

public async Task<Response> RegisterUserAsync(string urlBase, string servicePrefix, string controller, UserRequest userRequest)

{

try

{

string request = JsonConvert.SerializeObject(userRequest);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer

};

}

Response obj = JsonConvert.DeserializeObject<Response>(answer);

return obj;

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

Modificamos la **RegisterPageViewModel**

using Prism.Commands;

using Prism.Navigation;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Helpers;

using Xamarin.Forms;

namespace Taxi.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IRegexHelper \_regexHelper;

private readonly IApiService \_apiService;

private ImageSource \_image;

private UserRequest \_user;

private Role \_role;

private ObservableCollection<Role> \_roles;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

public RegisterPageViewModel(

INavigationService navigationService,

IRegexHelper regexHelper,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_regexHelper = regexHelper;

\_apiService = apiService;

Title = "Registro de Nuevo Usuario";

Image = App.Current.Resources["UrlNoImage"].ToString();

IsEnabled = true;

User = new UserRequest();

Roles = new ObservableCollection<Role>(CombosHelper.GetRoles());

}

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(RegisterAsync));

public ImageSource Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public UserRequest User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

public Role Role

{

get => \_role;

set => SetProperty(ref \_role, value);

}

public ObservableCollection<Role> Roles

{

get => \_roles;

set => SetProperty(ref \_roles, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void RegisterAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

User.UserTypeId = Role.Id;

Response response = await \_apiService.RegisterUserAsync(url, "api", "/Account", User);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(User.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Documento",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Nombre",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Apellido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Domicilio",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Email) || !\_regexHelper.IsValidEmail(User.Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Email válido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Phone))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Teléfono",

"Aceptar");

return false;

}

if (Role == null)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe seleccionar un Rol",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Password) || User.Password?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Password",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Confirmación de Password",

"Aceptar");

return false;

}

if (User.Password != User.PasswordConfirm)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"El Password y su confirmación no coinciden",

"Aceptar");

return false;

}

return true;

}

}

}

Agregamos el Nugget **Xam.Plugin.Media** a los ¨Proyectos Prism

Modificamos el **AndroidManifest**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.luisnu.taxi" android:installLocation="auto">

<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_MOCK\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application android:label="Taxi" android:icon="@mipmap/ic\_launcher">

<meta-data android:name="com.google.android.maps.v2.API\_KEY" android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

<provider android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.fileprovider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths"></meta-data>

</provider>

<uses-library android:name="org.apache.http.legacy" android:required="false" />

</application>

</manifest>

Agregamos la carpeta **xml** dentro de **Resources** y dentro creamos el archivo **file\_paths.xml**:

<?xml version="1.0" encoding="utf-8" ?>

<paths xmlns:android="http://schemas.android.com/apk/res/android">

<external-files-path name="my\_images" path="Pictures" />

<external-files-path name="my\_movies" path="Movies" />

</paths>

Modificamos el **MainActivity**

using Android.App;

using Android.Content.PM;

using Android.OS;

using Android.Runtime;

using Plugin.CurrentActivity;

using Plugin.Permissions;

using Prism;

using Prism.Ioc;

using Syncfusion.SfBusyIndicator.XForms.Droid;

namespace Soccer2020.Prism.Droid

{

[Activity(Label = "Soccer 2020", Icon = "@mipmap/ic\_launcher", Theme = "@style/MainTheme", MainLauncher = false, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity

{

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

CrossCurrentActivity.Current.Init(this, bundle);

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfBusyIndicatorRenderer();

LoadApplication(new App(new AndroidInitializer()));

}

public override void OnRequestPermissionsResult(int requestCode, string[] permissions, [GeneratedEnum] Permission[] grantResults)

{

PermissionsImplementation.Current.OnRequestPermissionsResult(requestCode, permissions, grantResults);

}

}

public class AndroidInitializer : IPlatformInitializer

{

public void RegisterTypes(IContainerRegistry containerRegistry)

{

// Register any platform specific implementations

}

}

}

Modificamos el **info.plist**:

<key>NSCameraUsageDescription</key>

<string>This app needs access to the camera to take photos.</string>

<key>NSPhotoLibraryUsageDescription</key>

<string>This app needs access to photos.</string>

<key>NSMicrophoneUsageDescription</key>

<string>This app needs access to microphone.</string>

<key>NSPhotoLibraryAddUsageDescription</key>

<string>This app needs access to the photo gallery.</string>

Modificamos la **RegisterPage**

<ffimageloading:CachedImage.Transformations>

<fftransformations:CircleTransformation />

</ffimageloading:CachedImage.Transformations>

<ffimageloading:CachedImage.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ChangeImageCommand}"/>

</ffimageloading:CachedImage.GestureRecognizers>

</ffimageloading:CachedImage>

Agregamos el **IFilesHelper**

using System.IO;

namespace Taxi.Common.Helpers

{

public interface IFilesHelper

{

byte[] ReadFully(Stream input);

}

}

Agregamos el **FilesHelper**

namespace Taxi.Common.Helpers

{

public class FilesHelper

{

public static byte[] ReadFully(Stream input)

{

using (MemoryStream ms = new MemoryStream())

{

input.CopyTo(ms);

return ms.ToArray();

}

}

}

}

Agregamos la inyección en **App.xaml.cs**

containerRegistry.Register<IFilesHelper, FilesHelper>();

Modificamos la **RegisterPageViewModel**

using Plugin.Media;

using Plugin.Media.Abstractions;

using Prism.Commands;

using Prism.Navigation;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Helpers;

using Xamarin.Forms;

namespace Taxi.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IRegexHelper \_regexHelper;

private readonly IApiService \_apiService;

private readonly IFilesHelper \_filesHelper;

private ImageSource \_image;

private UserRequest \_user;

private Role \_role;

private ObservableCollection<Role> \_roles;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

private MediaFile \_file;

private DelegateCommand \_changeImageCommand;

public RegisterPageViewModel(

INavigationService navigationService,

IRegexHelper regexHelper,

IApiService apiService, IFilesHelper filesHelper) : base(navigationService)

{

\_navigationService = navigationService;

\_regexHelper = regexHelper;

\_apiService = apiService;

\_filesHelper = filesHelper;

Title = "Registro de Nuevo Usuario";

Image = App.Current.Resources["UrlNoImage"].ToString();

IsEnabled = true;

User = new UserRequest();

Roles = new ObservableCollection<Role>(CombosHelper.GetRoles());

}

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(RegisterAsync));

public DelegateCommand ChangeImageCommand => \_changeImageCommand ?? (\_changeImageCommand = new DelegateCommand(ChangeImageAsync));

public ImageSource Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public UserRequest User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

public Role Role

{

get => \_role;

set => SetProperty(ref \_role, value);

}

public ObservableCollection<Role> Roles

{

get => \_roles;

set => SetProperty(ref \_roles, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void RegisterAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

byte[] imageArray = null;

if (\_file != null)

{

imageArray = \_filesHelper.ReadFully(\_file.GetStream());

}

User.PictureArray = imageArray;

User.UserTypeId = Role.Id;

Response response = await \_apiService.RegisterUserAsync(url, "api", "/Account", User);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(User.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Documento",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Nombre",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Apellido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Domicilio",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Email) || !\_regexHelper.IsValidEmail(User.Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Email válido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Phone))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Teléfono",

"Aceptar");

return false;

}

if (Role == null)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe seleccionar un Rol",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Password) || User.Password?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar un Password",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Confirmación de Password",

"Aceptar");

return false;

}

if (User.Password != User.PasswordConfirm)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"El Password y su confirmación no coinciden",

"Aceptar");

return false;

}

return true;

}

private async void ChangeImageAsync()

{

await CrossMedia.Current.Initialize();

string source = await Application.Current.MainPage.DisplayActionSheet(

"¿De dónde quiere tomar la foto?",

"Cancelar",

null,

"Galería",

"Cámara");

if (source == "Cancelar")

{

\_file = null;

return;

}

if (source == "Cámara")

{

\_file = await CrossMedia.Current.TakePhotoAsync(

new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

}

);

}

else

{

\_file = await CrossMedia.Current.PickPhotoAsync();

}

if (\_file != null)

{

Image = ImageSource.FromStream(() =>

{

System.IO.Stream stream = \_file.GetStream();

return stream;

});

}

}

}

}

# Improve Delete Dialogs

Agregamos un archivo Java Script en **wwwroot/js** llamado **deleteDialog.js**

(function (taxiDeleteDialog) {

var methods = {

"openModal": openModal,

"deleteItem": deleteItem

};

var item\_to\_delete;

/\*\*

\* Open a modal by class name or Id.

\*

\* @return string id item.

\*/

function openModal(modalName, classOrId, sourceEvent, deletePath, eventClassOrId) {

var textEvent;

if (classOrId) {

textEvent = "." + modalName;

} else {

textEvent = "#" + modalName;

}

$(textEvent).click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

deleteItem(sourceEvent, deletePath, eventClassOrId);

});

}

/\*\*

\* Path to delete an item.

\*

\* @return void.

\*/

function deleteItem(sourceEvent, deletePath, eventClassOrId) {

var textEvent;

if (eventClassOrId) {

textEvent = "." + sourceEvent;

} else {

textEvent = "#" + sourceEvent;

}

$(textEvent).click(function () {

window.location.href = deletePath + item\_to\_delete;

});

}

taxiDeleteDialog.sc\_deleteDialog = methods;

})(window);

Modificamos en los script de Delete dentro de las Vistas del Proyecto Web:

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script src="/js/deleteDialog.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

sc\_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Taxis/Delete/', false);

});

</script>

}

# Recuperar Password desde la App

Agregamos la **RememberPasswordPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:i18n="clr-namespace:Soccer.Prism.Helpers"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.RememberPasswordPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Agregamos la **RememberPasswordPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class RememberPasswordPageViewModel : ViewModelBase

{

public RememberPasswordPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Recupero de Password";

}

}

}

Modificamos la **LoginPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.LoginPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Image HeightRequest="150"

Margin="0,20,0,0"

Source="Logo"/>

<StackLayout VerticalOptions="CenterAndExpand">

<Grid Margin="20"

VerticalOptions="CenterAndExpand">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

HorizontalTextAlignment="End"

Text="Email:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="0"

BackgroundColor="{StaticResource ColorFontInverse}"

Keyboard="Email"

Placeholder="Ingrese Email..."

Text="{Binding Email}"/>

<Label Grid.Column="0"

Grid.Row="1"

HorizontalTextAlignment="End"

Text="Password:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="1"

BackgroundColor="{StaticResource ColorFontInverse}"

IsPassword="True"

Placeholder="Ingrese Password..."

Text="{Binding Password}"/>

</Grid>

<Label FontAttributes="Bold"

HorizontalOptions="Center"

Text="¿Olvidó su Password?"

TextColor="{StaticResource ColorAccent}">

<Label.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ForgotPasswordCommand}"/>

</Label.GestureRecognizers>

</Label>

</StackLayout>

<StackLayout>

<Button Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Iniciar Sesion"/>

<Button Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Registrarse"

Style="{StaticResource DangerButton}"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos la **LoginViewModel**

private DelegateCommand \_forgotPasswordCommand;

…

public DelegateCommand ForgotPasswordCommand => \_forgotPasswordCommand ?? (\_forgotPasswordCommand = new DelegateCommand(ForgotPasswordAsync));

…

private async void ForgotPasswordAsync()

{

await \_navigationService.NavigateAsync(nameof(RememberPasswordPage));

}

Modificamos la **RememberPasswordPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Soccer2020.Prism.Views.RememberPasswordPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Label Text="Email:"/>

<Entry Keyboard="Email"

HorizontalOptions="FillAndExpand"

Placeholder="Ingrese Mail..."

Text="{Binding Email}"/>

<Button

Command="{Binding RecoverCommand}"

IsEnabled="{Binding IsEnabled}"

HorizontalOptions="FillAndExpand"

Text="Recuperar Password"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Agregamos al **IApiService**

Task<Response> RecoverPasswordAsync(string urlBase, string servicePrefix, string controller, EmailRequest emailRequest);

Agregamos al **ApiService**

public async Task<Response> RecoverPasswordAsync(string urlBase, string servicePrefix, string controller, EmailRequest emailRequest)

{

try

{

string request = JsonConvert.SerializeObject(emailRequest);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

Response obj = JsonConvert.DeserializeObject<Response>(answer);

return obj;

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos la **RememberPasswordPageViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using System.Threading.Tasks;

namespace Taxi.Prism.ViewModels

{

public class RememberPasswordPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private readonly IRegexHelper \_regexHelper;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_recoverCommand;

public RememberPasswordPageViewModel(INavigationService navigationService, IApiService apiService, IRegexHelper regexHelper)

: base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

\_regexHelper = regexHelper;

Title = "Recupero de Password";

IsEnabled = true;

}

public DelegateCommand RecoverCommand => \_recoverCommand ?? (\_recoverCommand = new DelegateCommand(RecoverAsync));

public string Email { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void RecoverAsync()

{

bool isValid = await ValidateData();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

EmailRequest request = new EmailRequest

{

Email = Email,

};

string url = App.Current.Resources["UrlAPI"].ToString();

Response response = await \_apiService.RecoverPasswordAsync(url, "api", "/Account/RecoverPassword", request);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Email) || !\_regexHelper.IsValidEmail(Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un mail válido.",

"Aceptar");

return false;

}

return true;

}

}

}

# Modificar Usuario desde la App

Modificamos la **ModifyUserPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:i18n="clr-namespace:Taxi.Prism.Helpers"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

xmlns:fftransformations="clr-namespace:FFImageLoading.Transformations;assembly=FFImageLoading.Transformations"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.ModifyUserPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<ScrollView>

<StackLayout>

<ffimageloading:CachedImage Aspect="AspectFit"

Source="{Binding Image}"

CacheDuration= "50"

HeightRequest="150"

Margin="5"

RetryCount= "3"

RetryDelay= "600">

<ffimageloading:CachedImage.Transformations>

<fftransformations:CircleTransformation />

</ffimageloading:CachedImage.Transformations>

<ffimageloading:CachedImage.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ChangeImageCommand}"/>

</ffimageloading:CachedImage.GestureRecognizers>

</ffimageloading:CachedImage>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Row="0"

Grid.Column="0"

Text="Documento:"

VerticalOptions="Center"/>

<Entry Grid.Row="0"

Grid.Column="1"

Keyboard="Numeric"

Placeholder="Ingrese Documento..."

Text="{Binding User.Document}"/>

<Label Grid.Row="1"

Grid.Column="0"

Text="Nombre:"

VerticalOptions="Center"/>

<Entry Grid.Row="1"

Grid.Column="1"

Placeholder="Ingrese Nombre..."

Text="{Binding User.FirstName}"/>

<Label Grid.Row="2"

Grid.Column="0"

Text="Apellido:"

VerticalOptions="Center"/>

<Entry Grid.Row="2"

Grid.Column="1"

Placeholder="Ingrese Apellido..."

Text="{Binding User.LastName}"/>

<Label Grid.Row="3"

Grid.Column="0"

Text="Domicilio:"

VerticalOptions="Center"/>

<Entry Grid.Row="3"

Grid.Column="1"

Placeholder="Ingrese Domicilio..."

Text="{Binding User.Address}"/>

<Label Grid.Row="4"

Grid.Column="0"

Text="Teléfono:"

VerticalOptions="Center"/>

<Entry Grid.Row="4"

Grid.Column="1"

Keyboard="Telephone"

Placeholder="Ingrese Teléfono..."

Text="{Binding User.PhoneNumber}"/>

</Grid>

</StackLayout>

</ScrollView>

<StackLayout Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button Command="{Binding SaveCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Guardar"/>

<Button Command="{Binding ChangePasswordCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Cambiar Password"

Style="{StaticResource DangerButton}"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Guardando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos la **ModifyUserPageViewModel**

using Newtonsoft.Json;

using Plugin.Media;

using Plugin.Media.Abstractions;

using Prism.Commands;

using Prism.Navigation;

using System.Threading.Tasks;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Xamarin.Forms;

namespace Taxi.Prism.ViewModels

{

public class ModifyUserPageViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private ImageSource \_image;

private UserResponse \_user;

private MediaFile \_file;

private DelegateCommand \_changeImageCommand;

private DelegateCommand \_saveCommand;

public ModifyUserPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Modificar Usuario";

IsEnabled = true;

User = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

Image = User.PictureFullPath;

}

public DelegateCommand ChangeImageCommand => \_changeImageCommand ?? (\_changeImageCommand = new DelegateCommand(ChangeImageAsync));

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(SaveAsync));

public ImageSource Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public UserResponse User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void SaveAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(User.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Documento",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Nombre",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Apellido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Docmicilio",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.PhoneNumber))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Teléfono",

"Aceptar");

return false;

}

return true;

}

private async void ChangeImageAsync()

{

await CrossMedia.Current.Initialize();

string source = await Application.Current.MainPage.DisplayActionSheet(

"¿De dónde quiere tomar la foto?",

"Cancelar",

null,

"Galería",

"Cámara");

if (source == "Cancelar")

{

\_file = null;

return;

}

if (source == "Cámara")

{

\_file = await CrossMedia.Current.TakePhotoAsync(

new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

}

);

}

else

{

\_file = await CrossMedia.Current.PickPhotoAsync();

}

if (\_file != null)

{

Image = ImageSource.FromStream(() =>

{

System.IO.Stream stream = \_file.GetStream();

return stream;

});

}

}

}

}

Agregamos al **IApiService**

Task<Response> PutAsync<T>(string urlBase, string servicePrefix, string controller, T model, string tokenType, string accessToken);

Agregamos al **IApiService**

public async Task<Response> PutAsync<T>(string urlBase, string servicePrefix, string controller, T model, string tokenType, string accessToken)

{

try

{

string request = JsonConvert.SerializeObject(model);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PutAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos **ModifyUserPageViewModel**

using Newtonsoft.Json;

using Plugin.Media;

using Plugin.Media.Abstractions;

using Prism.Commands;

using Prism.Navigation;

using System.Threading.Tasks;

using Taxi.Common.Enums;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Xamarin.Forms;

namespace Taxi.Prism.ViewModels

{

public class ModifyUserPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private readonly IFilesHelper \_filesHelper;

private bool \_isRunning;

private bool \_isEnabled;

private ImageSource \_image;

private UserResponse \_user;

private MediaFile \_file;

private DelegateCommand \_changeImageCommand;

private DelegateCommand \_saveCommand;

public ModifyUserPageViewModel(INavigationService navigationService, IApiService apiService, IFilesHelper filesHelper)

: base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

\_filesHelper = filesHelper;

Title = "Modificar Usuario";

IsEnabled = true;

User = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

Image = User.PictureFullPath;

}

public DelegateCommand ChangeImageCommand => \_changeImageCommand ?? (\_changeImageCommand = new DelegateCommand(ChangeImageAsync));

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(SaveAsync));

public ImageSource Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public UserResponse User

{

get => \_user;

set => SetProperty(ref \_user, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void SaveAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

byte[] imageArray = null;

if (\_file != null)

{

imageArray = \_filesHelper.ReadFully(\_file.GetStream());

}

var userRequest = new UserRequest

{

Address = User.Address,

Document = User.Document,

Email = User.Email,

FirstName = User.FirstName,

LastName = User.LastName,

Password = "123456", // It doesn't matter what is sent here. It is only for the model to be valid

Phone = User.PhoneNumber,

PictureArray = imageArray,

UserTypeId = User.UserType == UserType.User ? 1 : 2,

};

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.PutAsync(url, "api", "/Account", userRequest, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

return;

}

Settings.User = JsonConvert.SerializeObject(User);

await App.Current.MainPage.DisplayAlert(

"Ok",

"Usuario modificado con éxito!",

"Aceptar");

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(User.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Documento",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Nombre",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Apellido",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Docmicilio",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(User.PhoneNumber))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un Teléfono",

"Aceptar");

return false;

}

return true;

}

private async void ChangeImageAsync()

{

await CrossMedia.Current.Initialize();

string source = await Application.Current.MainPage.DisplayActionSheet(

"¿De dónde quiere tomar la foto?",

"Cancelar",

null,

"Galería",

"Cámara");

if (source == "Cancelar")

{

\_file = null;

return;

}

if (source == "Cámara")

{

\_file = await CrossMedia.Current.TakePhotoAsync(

new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

}

);

}

else

{

\_file = await CrossMedia.Current.PickPhotoAsync();

}

if (\_file != null)

{

Image = ImageSource.FromStream(() =>

{

System.IO.Stream stream = \_file.GetStream();

return stream;

});

}

}

}

}

Modificamos la **TaxiMasterDetailPage**

<ffimageloading:CachedImage x:Name="Picture"

RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView, ElementName=Logo,Property=Y,Factor=1,Constant=30}"

Aspect="AspectFill"

Source="{Binding User.PictureFullPath}"

CacheDuration= "50"

HeightRequest="110"

WidthRequest="110"

Margin="5"

RetryCount= "3"

RetryDelay= "600">

<ffimageloading:CachedImage.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ModifyUserCommand}"/>

</ffimageloading:CachedImage.GestureRecognizers>

<ffimageloading:CachedImage.Transformations>

<fftransformations:CircleTransformation />

</ffimageloading:CachedImage.Transformations>

</ffimageloading:CachedImage>

Modificamos la **TaxiMasterDetailPageViewModel**

private DelegateCommand \_modifyUserCommand;

…

public DelegateCommand ModifyUserCommand => \_modifyUserCommand ?? (\_modifyUserCommand = new DelegateCommand(ModifyUserAsync));

…

private async void ModifyUserAsync()

{

await \_navigationService.NavigateAsync($"/TaxiMasterDetailPage/NavigationPage/{nameof(ModifyUserPage)}");

}

private readonly IApiService \_apiService;

…

private static TaxiMasterDetailPageViewModel \_instance;

…

public SoccerMasterDetailPageViewModel(INavigationService navigationService, IApiService apiService ) : base(navigationService)

…

public SoccerMasterDetailPageViewModel(INavigationService navigationService, IApiService apiService ) : base(navigationService)

{

\_instance = this;

\_navigationService = navigationService;

\_apiService = apiService;

LoadUser();

LoadMenus();

}

…

public static SoccerMasterDetailPageViewModel GetInstance()

{

return \_instance;

}

public async void ReloadUser()

{

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

return;

}

UserResponse user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

EmailRequest emailRequest = new EmailRequest

{

Email = user.Email

};

Response response = await \_apiService.GetUserByEmail(url, "api", "/Account/GetUserByEmail", "bearer", token.Token, emailRequest);

UserResponse userResponse = (UserResponse)response.Result;

Settings.User = JsonConvert.SerializeObject(userResponse);

LoadUser();

}

Modificamos **ModifyUserPageViewModel**

Settings.User = JsonConvert.SerializeObject(User);

SoccerMasterDetailPageViewModel.GetInstance().ReloadUser();

await App.Current.MainPage.DisplayAlert(

"Ok",

"El Usuario fue actualizado.",

"Aceptar");

# Modificar Password desde la App

Agregamos en **IApiService**

Task<Response> ChangePasswordAsync(string urlBase, string servicePrefix, string controller, ChangePasswordRequest changePasswordRequest, string tokenType, string accessToken);

Agregamos en **ApiService**

public async Task<Response> ChangePasswordAsync(string urlBase, string servicePrefix, string controller, ChangePasswordRequest changePasswordRequest, string tokenType, string accessToken)

{

try

{

string request = JsonConvert.SerializeObject(changePasswordRequest);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

Response obj = JsonConvert.DeserializeObject<Response>(answer);

return obj;

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Agregamos la **ChangePasswordPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"

ios:Page.UseSafeArea="true"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.ChangePasswordPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

Text="Password actual:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="0"

IsPassword="True"

Placeholder="Ingrese Password actual..."

Text="{Binding CurrentPassword}"/>

<Label Grid.Column="0"

Grid.Row="1"

Text="Nuevo Password:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="1"

IsPassword="True"

Placeholder="Ingrese Nuevo Password..."

Text="{Binding NewPassword}"/>

<Label Grid.Column="0"

Grid.Row="2"

Text="Confirme Password:"

VerticalOptions="Center"/>

<Entry Grid.Column="1"

Grid.Row="2"

IsPassword="True"

Placeholder="Ingrese Confirm. de Passsword..."

Text="{Binding PasswordConfirm}"/>

</Grid>

<Button Command="{Binding ChangePasswordCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Cambiar Password"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cambiando Password..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

Modificamos la **ChangePasswordViewModel**

using System.Threading.Tasks;

using Taxi.Common.Services;

using Prism.Commands;

using Prism.Navigation;

using Taxi.Prism.Helpers;

namespace Taxi.Prism.ViewModels

{

public class ChangePasswordPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_changePasswordCommand;

public ChangePasswordPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

IsEnabled = true;

Title = "Cambio de Password";

}

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePasswordAsync));

public string CurrentPassword { get; set; }

public string NewPassword { get; set; }

public string PasswordConfirm { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void ChangePasswordAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(CurrentPassword))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese el Password actual",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(NewPassword) || NewPassword?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese un nuevo Password",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese la confirmación del nuevo Password",

"Aceptar");

return false;

}

if (NewPassword != PasswordConfirm)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"El nuevo Password y su confirmación no son iguales.",

"Aceptar");

return false;

}

return true;

}

}

}

Modificamos **ModifyUserPageViewModel**

private DelegateCommand \_changePasswordCommand;

…

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePasswordAsync));

…

private async void ChangePasswordAsync()

{

await \_navigationService.NavigateAsync(nameof(ChangePasswordPage));

}

Modificamos **ChangePasswordPageViewModel**

private async void ChangePasswordAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

UserResponse user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

ChangePasswordRequest request = new ChangePasswordRequest

{

Email = user.Email,

NewPassword = NewPassword,

OldPassword = CurrentPassword,

};

string url = App.Current.Resources["UrlAPI"].ToString();

Response response = await \_apiService.ChangePasswordAsync(url, "api", "/Account/ChangePassword", request, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

# Comenzar un Viaje

Creamos la **StartTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.StartTripPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos la **StartTripPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class StartTripPageViewModel : ViewModelBase

{

public StartTripPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Comenzar Viaje";

}

}

}

Modificamos la **HomePage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.HomePage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<StackLayout>

<maps:Map x:Name="MyMap"

MapType="Street"/>

<StackLayout Padding="5">

<Button

Command="{Binding StartTripCommand}"

Margin="10,0"

Text="Iniciar Viaje"/>

</StackLayout>

</StackLayout>

</ContentPage>

Modificamos la **HomePageViewModel**

using Prism.Commands;

using Prism.Navigation;

using Taxi.Common.Helpers;

using Taxi.Prism.Views;

namespace Taxi.Prism.ViewModels

{

public class HomePageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_startTripCommand;

public HomePageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Taxi Qualifier";

}

public DelegateCommand StartTripCommand => \_startTripCommand ?? (\_startTripCommand = new DelegateCommand(StartTripAsync));

private async void StartTripAsync()

{

if (Settings.IsLogin)

{

await \_navigationService.NavigateAsync(nameof(StartTripPage));

}

else

{

await \_navigationService.NavigateAsync(nameof(LoginPage));

}

}

}

}

Modificamos la **StartTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

xmlns:i18n="clr-namespace:Taxi.Prism.Helpers"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.StartTripPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Row="0"

Grid.Column="0"

Text="Patente:"

VerticalOptions="Center"/>

<Entry Grid.Row="0"

Grid.Column="1"

FontSize="48"

HorizontalOptions="FillAndExpand"

Placeholder="Ingrese Patente..."

Text="{Binding Plaque}"/>

<Label Grid.Row="1"

Grid.Column="0"

HorizontalOptions="FillAndExpand"

Text="Origen:"

VerticalOptions="Center"/>

<StackLayout Grid.Row="1"

Grid.Column="1"

Orientation="Horizontal">

<Entry HorizontalOptions="FillAndExpand"

Placeholder="Ingrese Origen..."

Text="{Binding Source}"/>

<ImageButton Command="{Binding GetAddressCommand}"

Source="ic\_airport\_shuttle"/>

</StackLayout>

</Grid>

<maps:Map x:Name="MyMap"

MapType="Street"/>

<StackLayout Orientation="Horizontal">

<Button Command="{Binding StartTripCommand}"

IsEnabled="{Binding IsEnabled}"

Text="{Binding ButtonLabel}"/>

<Button Command="{Binding CancelTripCommand}"

IsVisible="{Binding IsSecondButtonVisible}"

Style="{StaticResource DangerButton}"

Text="Cancelar"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos el Code Behind de la página **StartTripPage** …. o sea el **StartTripPage.xaml.cs**

using Plugin.Permissions;

using Plugin.Permissions.Abstractions;

using System.Threading.Tasks;

using Taxi.Common.Services;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.Views

{

public partial class StartTripPage : ContentPage

{

private readonly IGeolocatorService \_geolocatorService;

private static StartTripPage \_instance;

public StartTripPage(IGeolocatorService geolocatorService)

{

InitializeComponent();

\_instance = this;

\_geolocatorService = geolocatorService;

}

public static StartTripPage GetInstance()

{

return \_instance;

}

public void AddPin(Position position, string address, string label, PinType pinType)

{

MyMap.Pins.Add(new Pin

{

Address = address,

Label = label,

Position = position,

Type = pinType

});

}

protected override void OnAppearing()

{

base.OnAppearing();

MoveMapToCurrentPositionAsync();

}

private async void MoveMapToCurrentPositionAsync()

{

bool isLocationPermision = await CheckLocationPermisionsAsync();

if (isLocationPermision)

{

MyMap.IsShowingUser = true;

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

Position position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

MoveMap(position);

}

}

}

private void MoveMap(Position position)

{

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(

position,

Distance.FromKilometers(.2)));

}

public void DrawLine(Position a, Position b)

{

if (Device.RuntimePlatform == Device.Android)

{

Polygon polygon = new Polygon

{

StrokeWidth = 10,

StrokeColor = Color.FromHex("#8D07F6"),

FillColor = Color.FromHex("#8D07F6"),

Geopath = { a, b }

};

MyMap.MapElements.Add(polygon);

}

else

{

AddPin(b, string.Empty, string.Empty, PinType.SavedPin);

}

MoveMap(b);

}

private async Task<bool> CheckLocationPermisionsAsync()

{

PermissionStatus permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

PermissionStatus permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);

PermissionStatus permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);

bool isLocationEnabled = permissionLocation == PermissionStatus.Granted ||

permissionLocationAlways == PermissionStatus.Granted ||

permissionLocationWhenInUse == PermissionStatus.Granted;

if (isLocationEnabled)

{

return true;

}

await CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);

permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);

permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);

return permissionLocation == PermissionStatus.Granted ||

permissionLocationAlways == PermissionStatus.Granted ||

permissionLocationWhenInUse == PermissionStatus.Granted;

}

}

}

Modificamos la **StartTripPageViewModel**

using Prism.Commands;

using Prism.Navigation;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using Taxi.Common.Services;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.ViewModels

{

public class StartTripPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IGeolocatorService \_geolocatorService;

private string \_source;

private string \_buttonLabel;

private bool \_isSecondButtonVisible;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_getAddressCommand;

private DelegateCommand \_startTripCommand;

public StartTripPageViewModel(INavigationService navigationService, IGeolocatorService geolocatorService)

: base(navigationService)

{

\_navigationService = navigationService;

\_geolocatorService = geolocatorService;

Title = "Comenzar Viaje";

ButtonLabel = "Iniciar Viaje";

LoadSourceAsync();

}

public DelegateCommand GetAddressCommand => \_getAddressCommand ?? (\_getAddressCommand = new DelegateCommand(LoadSourceAsync));

public DelegateCommand StartTripCommand => \_startTripCommand ?? (\_startTripCommand = new DelegateCommand(StartTripAsync));

public string Plaque { get; set; }

public bool IsSecondButtonVisible

{

get => \_isSecondButtonVisible;

set => SetProperty(ref \_isSecondButtonVisible, value);

}

public string ButtonLabel

{

get => \_source;

set => SetProperty(ref \_source, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public string Source

{

get => \_buttonLabel;

set => SetProperty(ref \_buttonLabel, value);

}

private async void LoadSourceAsync()

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

Position position = new Position(\_geolocatorService.Latitude, \_geolocatorService.Longitude);

Geocoder geoCoder = new Geocoder();

IEnumerable<string> sources = await geoCoder.GetAddressesForPositionAsync(position);

List<string> addresses = new List<string>(sources);

if (addresses.Count > 0)

{

Source = addresses[0];

}

}

}

private async void StartTripAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Patente",

"Aceptar");

return false;

}

Regex regex = new Regex(@"^([A-Za-z]{3}\d{3})$");

if (!regex.IsMatch(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"La Patente debe tener 3 Letras y 3 Números",

"Aceptar");

return false;

}

return true;

}

}

}

Nos aseguramos que figure esta línea en el **MainActivity**

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

CrossCurrentActivity.Current.Init(this, bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

Xamarin.FormsMaps.Init(this, bundle);

new SfRatingRenderer();

Xamarin.FormsMaps.Init(this, bundle);

new SfBusyIndicatorRenderer();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

LoadApplication(new App(new AndroidInitializer()));

}

Agregamos al **IApiService**

Task<Response> NewTripAsync(string urlBase, string servicePrefix, string controller, TripRequest model, string tokenType, string accessToken);

Agregamos al **ApiService**

public async Task<Response> NewTripAsync(string urlBase, string servicePrefix, string controller, TripRequest model, string tokenType, string accessToken)

{

try

{

string request = JsonConvert.SerializeObject(model);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

TripResponse trip = JsonConvert.DeserializeObject<TripResponse>(answer);

return new Response

{

IsSuccess = true,

Result = trip,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos **StartTripPage.xaml.cs**

private static StartTripPage \_instance;

…

public StartTripPage(IGeolocatorService geolocatorService)

{

InitializeComponent();

\_geolocatorService = geolocatorService;

\_instance = this;

}

…

public static StartTripPage GetInstance()

{

return \_instance;

}

public void AddPin(Position position, string address, string label, PinType pinType)

{

MyMap.Pins.Add(new Pin

{

Address = address,

Label = label,

Position = position,

Type = pinType

});

}

Modificamos **StartTripPageViewModel**

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Views;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.ViewModels

{

public class StartTripPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IGeolocatorService \_geolocatorService;

private readonly ApiService \_apiService;

private string \_source;

private string \_buttonLabel;

private bool \_isSecondButtonVisible;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_getAddressCommand;

private DelegateCommand \_startTripCommand;

private TripResponse \_tripResponse;

private Position \_position;

public StartTripPageViewModel(INavigationService navigationService, IGeolocatorService geolocatorService, ApiService apiService)

: base(navigationService)

{

\_navigationService = navigationService;

\_geolocatorService = geolocatorService;

\_apiService = apiService;

Title = "Comenzar Viaje";

ButtonLabel = "Iniciar Viaje";

IsEnabled = true;

LoadSourceAsync();

}

public DelegateCommand GetAddressCommand => \_getAddressCommand ?? (\_getAddressCommand = new DelegateCommand(LoadSourceAsync));

public DelegateCommand StartTripCommand => \_startTripCommand ?? (\_startTripCommand = new DelegateCommand(StartTripAsync));

public string Plaque { get; set; }

public bool IsSecondButtonVisible

{

get => \_isSecondButtonVisible;

set => SetProperty(ref \_isSecondButtonVisible, value);

}

public string ButtonLabel

{

get => \_source;

set => SetProperty(ref \_source, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public string Source

{

get => \_buttonLabel;

set => SetProperty(ref \_buttonLabel, value);

}

private async void LoadSourceAsync()

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

\_position = new Position(\_geolocatorService.Latitude, \_geolocatorService.Longitude);

Geocoder geoCoder = new Geocoder();

IEnumerable<string> sources = await geoCoder.GetAddressesForPositionAsync(\_position);

List<string> addresses = new List<string>(sources);

if (addresses.Count > 0)

{

Source = addresses[0];

}

}

}

private async void StartTripAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

UserResponse user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

TripRequest tripRequest = new TripRequest

{

Address = Source,

Latitude = \_geolocatorService.Latitude,

Longitude = \_geolocatorService.Longitude,

Plaque = Plaque,

UserId = new Guid(user.Id)

};

Response response = await \_apiService.NewTripAsync(url, "api", "/Trips", tripRequest, "bearer", token.Token);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

\_tripResponse = (TripResponse)response.Result;

IsSecondButtonVisible = true;

ButtonLabel = "Finalizar viaje";

StartTripPage.GetInstance().AddPin(\_position, Source, "Iniciar Viaje", PinType.Place);

IsRunning = false;

IsEnabled = true;

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una Patente",

"Aceptar");

return false;

}

Regex regex = new Regex(@"^([A-Za-z]{3}\d{3})$");

if (!regex.IsMatch(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"La Patente debe tener 3 Letras y 3 Números",

"Aceptar");

return false;

}

return true;

}

}

}

Agregamos el Nugget **Xamarin.Essentials** a todos los proyectos Prims

Agregamos en **Prism/Helpers** la Clase **UnitOfLength**

namespace Taxi.Prism.Helpers

{

public class UnitOfLength

{

public static UnitOfLength Kilometers = new UnitOfLength(1.609344);

public static UnitOfLength NauticalMiles = new UnitOfLength(0.8684);

public static UnitOfLength Miles = new UnitOfLength(1);

private readonly double \_fromMilesFactor;

private UnitOfLength(double fromMilesFactor)

{

\_fromMilesFactor = fromMilesFactor;

}

public double ConvertFromMiles(double input)

{

return input \* \_fromMilesFactor;

}

}

}

Agregamos en **Prism/Helpers** la Clase **GeoHelper** (sirve para calcular la distancia entre dos puntos)

using System;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.Helpers

{

public static class GeoHelper

{

public static double GetDistance(Position baseCoordinates, Position targetCoordinates, UnitOfLength unitOfLength)

{

double baseRad = Math.PI \* baseCoordinates.Latitude / 180;

double targetRad = Math.PI \* targetCoordinates.Latitude / 180;

double theta = baseCoordinates.Longitude - targetCoordinates.Longitude;

double thetaRad = Math.PI \* theta / 180;

double dist =

Math.Sin(baseRad) \* Math.Sin(targetRad) + Math.Cos(baseRad) \*

Math.Cos(targetRad) \* Math.Cos(thetaRad);

dist = Math.Acos(dist);

dist = dist \* 180 / Math.PI;

dist = dist \* 60 \* 1.1515;

return unitOfLength.ConvertFromMiles(dist);

}

}

}

Modificamos la **HomePage.xaml.cs** (y hacemos lo mismo en **StartTripPage.xaml.cs**)

private async void MoveMapToCurrentPositionAsync()

{

bool isLocationPermision = await CheckLocationPermisionsAsync();

if (isLocationPermision)

{

MyMap.IsShowingUser = true;

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

Position position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

MoveMap(position);

}

}

}

private void MoveMap(Position position)

{

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(

position,

Distance.FromKilometers(.2)));

}

Agregamos al **IApiService**

Task<Response> AddTripDetailsAsync(string urlBase, string servicePrefix, string controller, TripDetailsRequest model, string tokenType, string accessToken);

Agregamos al **ApiService**

public async Task<Response> AddTripDetailsAsync(string urlBase, string servicePrefix, string controller, TripDetailsRequest model, string tokenType, string accessToken)

{

try

{

string request = JsonConvert.SerializeObject(model);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos la **StartTripPage.xaml.cs**

using Plugin.Permissions;

using Plugin.Permissions.Abstractions;

using System.Threading.Tasks;

using Taxi.Common.Services;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.Views

{

public partial class StartTripPage : ContentPage

{

private readonly IGeolocatorService \_geolocatorService;

private static StartTripPage \_instance;

public StartTripPage(IGeolocatorService geolocatorService)

{

InitializeComponent();

\_instance = this;

\_geolocatorService = geolocatorService;

}

public static StartTripPage GetInstance()

{

return \_instance;

}

public void AddPin(Position position, string address, string label, PinType pinType)

{

MyMap.Pins.Add(new Pin

{

Address = address,

Label = label,

Position = position,

Type = pinType

});

}

protected override void OnAppearing()

{

base.OnAppearing();

MoveMapToCurrentPositionAsync();

}

private async void MoveMapToCurrentPositionAsync()

{

bool isLocationPermision = await CheckLocationPermisionsAsync();

if (isLocationPermision)

{

MyMap.IsShowingUser = true;

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

Position position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

MoveMap(position);

}

}

}

private void MoveMap(Position position)

{

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(

position,

Distance.FromKilometers(.2)));

}

public void DrawLine(Position a, Position b)

{

if (Device.RuntimePlatform == Device.Android)

{

Polygon polygon = new Polygon

{

StrokeWidth = 10,

StrokeColor = Color.FromHex("#8D07F6"),

FillColor = Color.FromHex("#8D07F6"),

Geopath = { a, b }

};

MyMap.MapElements.Add(polygon);

}

else

{

AddPin(b, string.Empty, string.Empty, PinType.SavedPin);

}

MoveMap(b);

}

private async Task<bool> CheckLocationPermisionsAsync()

{

PermissionStatus permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

PermissionStatus permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);

PermissionStatus permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);

bool isLocationEnabled = permissionLocation == PermissionStatus.Granted ||

permissionLocationAlways == PermissionStatus.Granted ||

permissionLocationWhenInUse == PermissionStatus.Granted;

if (isLocationEnabled)

{

return true;

}

await CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);

permissionLocation = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

permissionLocationAlways = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationAlways);

permissionLocationWhenInUse = await CrossPermissions.Current.CheckPermissionStatusAsync(Permission.LocationWhenInUse);

return permissionLocation == PermissionStatus.Granted ||

permissionLocationAlways == PermissionStatus.Granted ||

permissionLocationWhenInUse == PermissionStatus.Granted;

}

}

}

Modificamos la **StartTripPageViewModel**

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Timers;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Helpers;

using Taxi.Prism.Views;

using Xamarin.Essentials;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.ViewModels

{

public class StartTripPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IGeolocatorService \_geolocatorService;

private readonly IApiService \_apiService;

private string \_source;

private string \_buttonLabel;

private bool \_isSecondButtonVisible;

private bool \_isRunning;

private bool \_isEnabled;

private Position \_position;

private TripResponse \_tripResponse;

private UserResponse \_user;

private TokenResponse \_token;

private string \_url;

private Timer \_timer;

private Geocoder \_geoCoder;

private TripDetailsRequest \_tripDetailsRequest;

private DelegateCommand \_getAddressCommand;

private DelegateCommand \_startTripCommand;

public StartTripPageViewModel(INavigationService navigationService, IGeolocatorService geolocatorService, IApiService apiService)

: base(navigationService)

{

\_navigationService = navigationService;

\_geolocatorService = geolocatorService;

\_apiService = apiService;

\_tripDetailsRequest = new TripDetailsRequest { TripDetails = new List<TripDetailRequest>() };

Title = "Iniciar Viaje";

ButtonLabel = "Iniciar Viaje";

IsEnabled = true;

LoadSourceAsync();

}

public DelegateCommand GetAddressCommand => \_getAddressCommand ?? (\_getAddressCommand = new DelegateCommand(LoadSourceAsync));

public DelegateCommand StartTripCommand => \_startTripCommand ?? (\_startTripCommand = new DelegateCommand(StartTripAsync));

public string Plaque { get; set; }

public bool IsSecondButtonVisible

{

get => \_isSecondButtonVisible;

set => SetProperty(ref \_isSecondButtonVisible, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public string ButtonLabel

{

get => \_source;

set => SetProperty(ref \_source, value);

}

public string Source

{

get => \_buttonLabel;

set => SetProperty(ref \_buttonLabel, value);

}

private async void LoadSourceAsync()

{

IsEnabled = false;

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude == 0 && \_geolocatorService.Longitude == 0)

{

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Error de Geolocalización",

"Aceptar");

await \_navigationService.GoBackAsync();

return;

}

\_position = new Position(\_geolocatorService.Latitude, \_geolocatorService.Longitude);

Geocoder geoCoder = new Geocoder();

IEnumerable<string> sources = await geoCoder.GetAddressesForPositionAsync(\_position);

List<string> addresses = new List<string>(sources);

if (addresses.Count > 1)

{

Source = addresses[0];

}

IsEnabled = true;

}

private async void StartTripAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

\_url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(\_url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión aInternet",

"Aceptar");

return;

}

\_user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

\_token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

TripRequest tripRequest = new TripRequest

{

Address = Source,

Latitude = \_geolocatorService.Latitude,

Longitude = \_geolocatorService.Longitude,

Plaque = Plaque,

UserId = new Guid(\_user.Id)

};

Response response = await \_apiService.NewTripAsync(\_url, "api", "/Trips", tripRequest, "bearer", \_token.Token);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

\_tripResponse = (TripResponse)response.Result;

IsSecondButtonVisible = true;

ButtonLabel = "Finalizar viaje";

StartTripPage.GetInstance().AddPin(\_position, Source, "Iniciar Viaje", PinType.Place);

IsRunning = false;

IsEnabled = true;

\_timer = new Timer

{

Interval = 1000

};

\_timer.Elapsed += Timer\_Elapsed;

\_timer.Start();

}

private async void Timer\_Elapsed(object sender, ElapsedEventArgs e)

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude == 0 && \_geolocatorService.Longitude == 0)

{

return;

}

Position previousPosition = new Position(\_position.Latitude, \_position.Longitude);

\_position = new Position(\_geolocatorService.Latitude, \_geolocatorService.Longitude);

double distance = GeoHelper.GetDistance(previousPosition, \_position, UnitOfLength.Kilometers);

if (distance < 0.003 || double.IsNaN(distance))

{

return;

}

MainThread.BeginInvokeOnMainThread(() =>

{

StartTripPage.GetInstance().DrawLine(previousPosition, \_position);

});

\_tripDetailsRequest.TripDetails.Add(new TripDetailRequest

{

Latitude = \_position.Latitude,

Longitude = \_position.Longitude,

TripId = \_tripResponse.Id

});

if (\_tripDetailsRequest.TripDetails.Count > 9)

{

SendTripDetailsAsync();

}

}

private async Task SendTripDetailsAsync()

{

TripDetailsRequest tripDetailsRequestCloned = CloneTripDetailsRequest(\_tripDetailsRequest);

\_tripDetailsRequest.TripDetails.Clear();

await \_apiService.AddTripDetailsAsync(\_url, "api", "/Trips/AddTripDetails", tripDetailsRequestCloned, "bearer", \_token.Token);

}

private TripDetailsRequest CloneTripDetailsRequest(TripDetailsRequest tripDetailsRequest)

{

TripDetailsRequest tripDetailsRequestCloned = new TripDetailsRequest

{

TripDetails = tripDetailsRequest.TripDetails.Select(d => new TripDetailRequest

{

Address = d.Address,

Latitude = d.Latitude,

Longitude = d.Longitude,

TripId = d.TripId

}).ToList()

};

return tripDetailsRequestCloned;

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese una Patente",

"Aceptar");

return false;

}

Regex regex = new Regex(@"^([A-Za-z]{3}\d{3})$");

if (!regex.IsMatch(Plaque))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"La Patente debe tener 3 letras y 3 números",

"Aceptar");

return false;

}

return true;

}

}

}

# Mejorar Chequeo de Conexión a Internet

Instalar el Nuget **Xamarin.Essentials** en todos los Proyectos Prism

Deinstalar el Nuget **Xamarin.Plugin.Connectivity** en el Proyecto Common

En **IApiService** modificamos el método **CheckConnectionAsync**

bool CheckConnection();

En **ApiService** modificamos el método **CheckConnectionAsync**

public bool CheckConnection()

{

return Connectivity.NetworkAccess == NetworkAccess.Internet;

}

Modificamos el llamado al método **CheckConnectionAsync** en todos los lugares donde esté por:

if (!\_apiService.CheckConnection())

# Finalizar viaje

Agregamos la **EndTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.EndTripPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos la **EndTripPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class EndTripPageViewModel : ViewModelBase

{

public EndTripPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Finalizar viaje";

}

}

}

Modificamos la **StartTripPageViewModel**

private async void StartTripAsync()

{

bool isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

if (IsSecondButtonVisible)

{

await EndTripAsync();

}

else

{

await BeginTripAsync();

}

}

private async Task EndTripAsync()

{

await \_navigationService.NavigateAsync(nameof(EndTripPage));

}

private async Task BeginTripAsync()

{

IsRunning = true;

IsEnabled = false;

\_url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(\_url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión aInternet",

"Aceptar");

return;

}

\_user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

\_token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

TripRequest tripRequest = new TripRequest

{

Address = Source,

Latitude = \_geolocatorService.Latitude,

Longitude = \_geolocatorService.Longitude,

Plaque = Plaque,

UserId = new Guid(\_user.Id)

};

Response response = await \_apiService.NewTripAsync(\_url, "api", "/Trips", tripRequest, "bearer", \_token.Token);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

\_tripResponse = (TripResponse)response.Result;

IsSecondButtonVisible = true;

ButtonLabel = "Finalizar viaje";

StartTripPage.GetInstance().AddPin(\_position, Source, "Iniciar Viaje", PinType.Place);

IsRunning = false;

IsEnabled = true;

\_timer = new Timer

{

Interval = 1000

};

\_timer.Elapsed += Timer\_Elapsed;

\_timer.Start();

}

En **Common/Models** agregamos la Clase **Comment**

namespace Taxi.Common.Models

{

public class Comment

{

public int Id { get; set; }

public string Name { get; set; }

}

}

Agregamos al **Prism/CombosHelper**

public static List<Comment> GetComments()

{

return new List<Comment>

{

new Comment { Id = 1, Name = "Muy buen servicio"},

new Comment { Id = 2, Name = "Conductor muy amigable"},

new Comment { Id = 2, Name = "Taxi limpio"},

new Comment { Id = 2, Name = "Mal conductor"},

new Comment { Id = 2, Name = "Cobro mayor a lo esperado"},

new Comment { Id = 2, Name = "Taxi sucio o en mal estado"}

};

}

Agregamos al **App.xml**

<Style TargetType="Editor">

<Setter Property="BackgroundColor" Value="{StaticResource ColorFontInverse}" />

</Style>

Agregamos a **StartTripPageViewModel**

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (IsSecondButtonVisible && \_timer != null)

{

\_timer.Start();

}

}

Modificamos **StartTripPageViewModel**

private async Task EndTripAsync()

{

\_timer.Stop();

if (\_tripDetailsRequest.TripDetails.Count > 0)

{

await SendTripDetailsAsync();

}

NavigationParameters parameters = new NavigationParameters

{

{ "tripId", \_tripResponse.Id },

};

await \_navigationService.NavigateAsync(nameof(EndTripPage), parameters);

}

Modificamos la **EndTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:rating="clr-namespace:Syncfusion.SfRating.XForms;assembly=Syncfusion.SfRating.XForms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:i18n="clr-namespace:Taxi.Prism.Helpers"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.EndTripPage"

BackgroundColor="{StaticResource ColorBackgroud}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<ScrollView>

<StackLayout>

<rating:SfRating HorizontalOptions="Center"

Margin="0,5"

Precision="Half"

Value="{Binding Qualification}">

<rating:SfRating.RatingSettings>

<rating:SfRatingSettings RatedFill="{StaticResource ColorDanger}"/>

</rating:SfRating.RatingSettings>

</rating:SfRating>

<StackLayout HorizontalOptions="Center"

Orientation="Horizontal">

<Label FontAttributes="Bold"

FontSize="Large"

Text="Calificación"/>

<Label FontAttributes="Bold"

FontSize="Large"

Text="{Binding Qualification, StringFormat='{0:N2}'}"/>

</StackLayout>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Row="0"

Grid.Column="0"

Text="Comentario genérico"

VerticalOptions="Center"/>

<Picker Grid.Row="0"

Grid.Column="1"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding Comments}"

SelectedItem="{Binding Comment}"

Title="Ingrese un comentario..."/>

<Label Grid.Row="1"

Grid.Column="0"

Text="Comentario"

VerticalOptions="Center"/>

<Editor Grid.Row="1"

Grid.Column="1"

HeightRequest="150"

Text="{Binding Remark}"/>

<Label Grid.Row="2"

Grid.Column="0"

Text="Distancia recorrida"

VerticalOptions="Center"/>

<Label Grid.Row="2"

Grid.Column="1"

FontSize="Medium"

FontAttributes="Bold"

Text="{Binding Distance, StringFormat='{0:N2}'}"

VerticalTextAlignment="Center"/>

<Label Grid.Row="3"

Grid.Column="0"

Text="Tiempo"

VerticalOptions="Center"/>

<Label Grid.Row="3"

Grid.Column="1"

FontSize="Medium"

FontAttributes="Bold"

Text="{Binding Time}"

VerticalTextAlignment="Center"/>

<Label Grid.Row="4"

Grid.Column="0"

Text="Estimado"

VerticalOptions="Center"/>

<Label Grid.Row="4"

Grid.Column="1"

FontSize="Medium"

FontAttributes="Bold"

Text="{Binding Value}"

VerticalTextAlignment="Center"/>

</Grid>

</StackLayout>

</ScrollView>

<Button Command="{Binding EndTripCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Finalizar viaje"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorAccent}"

HorizontalOptions="Center"

TextColor="{StaticResource ColorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

En **Common/Models** agregamos la Clase **TripSummary**

using System;

namespace Taxi.Common.Models

{

public class TripSummary

{

public double Distance { get; set; }

public TimeSpan Time { get; set; }

public decimal Value { get; set; }

}

}

Agregamos a **GeoHelper**

public static TripSummary GetTripSummary(TripResponse trip)

{

try

{

double distance = 0;

if (trip.TripDetails == null || trip.TripDetails.Count < 2)

{

return new TripSummary();

}

List<TripDetailResponse> details = trip.TripDetails.ToList();

for (int i = 0; i < details.Count - 1; i++)

{

Position a = new Position(details[i].Latitude, details[i].Longitude);

Position b = new Position(details[i + 1].Latitude, details[i + 1].Longitude);

distance += GetDistance(a, b, UnitOfLength.Kilometers) \* 1000;

}

decimal value = (decimal)(3600 + Math.Truncate(distance / 78) \* 110);

return new TripSummary

{

Distance = distance,

Time = details[details.Count - 1].Date.Subtract(details[0].Date),

Value = value < 5600 ? 5600 : value

};

}

catch

{

return new TripSummary { Value = 5600 };

}

}

Agregamos a **IapiService**

Task<Response> GetTripAsync(string urlBase, string servicePrefix, string controller, int id, string tokenType, string accessToken);

Task<Response> CompleteTripAsync(string urlBase, string servicePrefix, string controller, CompleteTripRequest model, string tokenType, string accessToken);

Agregamos a **ApiService**

public async Task<Response> GetTripAsync(string urlBase, string servicePrefix, string controller, int id, string tokenType, string accessToken)

{

try

{

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}/{id}";

HttpResponseMessage response = await client.GetAsync(url);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

TripResponse trip = JsonConvert.DeserializeObject<TripResponse>(answer);

return new Response

{

IsSuccess = true,

Result = trip,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> CompleteTripAsync(string urlBase, string servicePrefix, string controller, CompleteTripRequest model, string tokenType, string accessToken)

{

try

{

string request = JsonConvert.SerializeObject(model);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos la **EndTripPageViewModel**

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

using Taxi.Prism.Helpers;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.ViewModels

{

public class EndTripPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private readonly IGeolocatorService \_geolocatorService;

private int \_tripId;

private bool \_isRunning;

private bool \_isEnabled;

private float \_qualification;

private Comment \_comment;

private ObservableCollection<Comment> \_comments;

private string \_remark;

private double \_distance;

private string \_time;

private string \_value;

private DelegateCommand \_endTripCommand;

public EndTripPageViewModel(INavigationService navigationService, IApiService apiService, IGeolocatorService geolocatorService)

: base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

\_geolocatorService = geolocatorService;

Title = "Finalizar viaje";

IsEnabled = true;

Comments = new ObservableCollection<Comment>(CombosHelper.GetComments());

}

public DelegateCommand EndTripCommand => \_endTripCommand ?? (\_endTripCommand = new DelegateCommand(EndTripAsync));

public string Value

{

get => \_value;

set => SetProperty(ref \_value, value);

}

public double Distance

{

get => \_distance;

set => SetProperty(ref \_distance, value);

}

public string Time

{

get => \_time;

set => SetProperty(ref \_time, value);

}

public string Remark

{

get => \_remark;

set => SetProperty(ref \_remark, value);

}

public Comment Comment

{

get => \_comment;

set

{

Comment comment = value;

Remark += string.IsNullOrEmpty(Remark) ? $"{comment.Name}" : $", {comment.Name}";

SetProperty(ref \_comment, value);

}

}

public ObservableCollection<Comment> Comments

{

get => \_comments;

set => SetProperty(ref \_comments, value);

}

public float Qualification

{

get => \_qualification;

set => SetProperty(ref \_qualification, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

\_tripId = parameters.GetValue<int>("tripId");

LoadTripAsync(\_tripId);

}

private async void LoadTripAsync(int id)

{

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión de Internet",

"Aceptar");

return;

}

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

Response response = await \_apiService.GetTripAsync(url, "api", "/Trips", id, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

TripResponse trip = (TripResponse)response.Result;

TripSummary tripSummary = GeoHelper.GetTripSummary(trip);

Distance = tripSummary.Distance;

Time = $"{tripSummary.Time.ToString().Substring(0, 8)}";

Value = $"{tripSummary.Value:C0}";

}

private async void EndTripAsync()

{

if (Qualification == 0)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Ingrese una Calificación",

"Aceptar");

return;

}

IsRunning = true;

IsEnabled = false;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

await \_geolocatorService.GetLocationAsync();

Position position = new Position();

string address = string.Empty;

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

position = new Position(\_geolocatorService.Latitude, \_geolocatorService.Longitude);

Geocoder geoCoder = new Geocoder();

IEnumerable<string> sources = await geoCoder.GetAddressesForPositionAsync(position);

List<string> addresses = new List<string>(sources);

if (addresses.Count > 1)

{

address = addresses[0];

}

}

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

CompleteTripRequest completeTripRequest = new CompleCompleteTripteTripRequest

{

Qualification = Qualification,

Remarks = Remark,

Target = address,

TargetLatitude = position.Latitude,

TargetLongitude = position.Longitude,

TripId = \_tripId

};

\_apiService.CompleteTripAsync(url, "api", "/Trips/CompleteTrip", completeTripRequest, "bearer", token.Token);

await \_navigationService.GoBackToRootAsync();

}

}

}

# Cancel Trip

Agregamos al **IApiService**

Task<Response> DeleteAsync(string urlBase, string servicePrefix, string controller, int id, string tokenType, string accessToken);

Agregamos al **ApiService**

public async Task<Response> DeleteAsync(string urlBase, string servicePrefix, string controller, int id, string tokenType, string accessToken)

{

try

{

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}/{id}";

HttpResponseMessage response = await client.DeleteAsync(url);

string answer = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

Modificamos **StartTripPageViewModel**

private DelegateCommand \_cancelTripCommand;

…

public DelegateCommand CancelTripCommand => \_cancelTripCommand ?? (\_cancelTripCommand = new DelegateCommand(CancelTripAsync));

…

private async void CancelTripAsync()

{

bool answer = await App.Current.MainPage.DisplayAlert(

"Atención!!",

"Está seguro de cancelar este viaje?",

"Sí",

"No");

if (!answer)

{

return;

}

IsRunning = true;

IsEnabled = false;

bool connection = await \_apiService.CheckConnectionAsync(\_url);

if (!connection)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

\_timer.Stop();

\_apiService.DeleteAsync(\_url, "api", "/Trips", \_tripResponse.Id, "bearer", \_token.Token);

IsRunning = false;

IsEnabled = true;

await \_navigationService.GoBackToRootAsync();

}

# See My Trips

Agregamos los íconos **location\_on** and **refresh**:

Agregamos la View **MyTripsPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"

ios:Page.UseSafeArea="true"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.MyTripsPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos **MyTripsPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class MyTripsPageViewModel : ViewModelBase

{

public MyTripsPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Mis Viajes";

}

}

}

Modificamos el **Menu**

namespace Taxi.Common.Models

{

public class Menu

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

public bool IsLoginRequired { get; set; }

}

}

Modificamos **TaxiMasterDetailPageViewModel**

new Menu

{

Icon = "ic\_action\_local\_taxi",

PageName = "TaxiHistoryPage",

Title = "Ver historial del Taxi"

},

new Menu

{

Icon = "ic\_location\_on",

PageName = "MyTripsPage",

Title = "Mis Viajes",

IsLoginRequired = true

},

new Menu

{

Icon = "ic\_action\_people",

PageName = "GroupPage",

Title = "Administrar mi grupo de usuarios"

},

List<Menu> menus = new List<Menu>

{

new Menu

{

Icon = "ic\_action\_airport\_shuttle",

PageName = "HomePage",

Title = "Nuevo viaje",

IsLoginRequired=false

},

new Menu

{

Icon = "ic\_action\_local\_taxi",

PageName = "TaxiHistoryPage",

Title = "Ver historial del Taxi",

IsLoginRequired=false

},

new Menu

{

Icon = "ic\_location\_on",

PageName = "MyTripsPage",

Title = "Mis Viajes",

IsLoginRequired = true

},

new Menu

{

Icon = "ic\_action\_people",

PageName = "GroupPage",

Title = "Administrar mi grupo de usuarios",

IsLoginRequired=true

},

new Menu

{

Icon = "ic\_action\_account\_circle",

PageName = "ModifyUserPage",

Title = "Modificar Usuario",

IsLoginRequired=true

},

new Menu

{

Icon = "ic\_action\_report",

PageName = "ReportPage",

Title = "Informar un incidente",

IsLoginRequired=false

},

new Menu

{

Icon = "ic\_action\_exit\_to\_app",

PageName = "LoginPage",

Title = Settings.IsLogin ? "Cerrar sesión" : "Iniciar sesión",

IsLoginRequired=false

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title,

IsLoginRequired=m.IsLoginRequired

}).ToList());

Modificamos **MenuItemViewModel**

if (IsLoginRequired && !Settings.IsLogin)

{

await \_navigationService.NavigateAsync($"/TaxiMasterDetailPage/NavigationPage/LoginPage");

}

else

{

await \_navigationService.NavigateAsync($"/TaxiMasterDetailPage/NavigationPage/{PageName}");

}

Agregamos al **IApiService**

Task<Response> GetMyTrips(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, MyTripsRequest model);

Agregamos al **ApiService**

public async Task<Response> GetMyTrips(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, MyTripsRequest model)

{

try

{

string request = JsonConvert.SerializeObject(model);

StringContent content = new StringContent(request, Encoding.UTF8, "application/json");

HttpClient client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

string url = $"{servicePrefix}{controller}";

HttpResponseMessage response = await client.PostAsync(url, content);

string result = await response.Content.ReadAsStringAsync();

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

List<TripResponse> trips = JsonConvert.DeserializeObject<List<TripResponse>>(result);

return new Response

{

IsSuccess = true,

Result = trips

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

Agregamos la View **MyTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"

ios:Page.UseSafeArea="true"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.MyTripPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

</ContentPage>

Modificamos la **MyTripPageViewModel**

using Prism.Navigation;

namespace Taxi.Prism.ViewModels

{

public class MyTripPageViewModel : ViewModelBase

{

public MyTripPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Mi Viaje";

}

}

}

Modificamos **TripItemViewModel**

private DelegateCommand \_selectTrip2Command;

…

public DelegateCommand SelectTrip2Command => \_selectTrip2Command ?? (\_selectTrip2Command = new DelegateCommand(SelectTrip2Async));

…

private async void SelectTrip2Async()

{

NavigationParameters parameters = new NavigationParameters

{

{ "trip", this }

};

await \_navigationService.NavigateAsync(nameof(MyTripPage), parameters);

}

Modificamos **MyTripsPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"

ios:Page.UseSafeArea="true"

x:Class="Taxi.Prism.Views.MyTripsPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

FontAttributes="Bold"

Text="Desde:"/>

<Label Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

Text="Hasta:"/>

<DatePicker Grid.Column="0"

Grid.Row="1"

Date="{Binding StartDate}"/>

<DatePicker Grid.Column="1"

Grid.Row="1"

Date="{Binding EndDate}"/>

<ImageButton Grid.Column="2"

Grid.Row="0"

Grid.RowSpan="2"

Command="{Binding RefreshCommand}"

Source="ic\_refresh"/>

</Grid>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="2\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="4\*" />

<ColumnDefinition Width="48" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

FontAttributes="Bold"

Text="Fecha:"/>

<Label Grid.Column="1"

FontAttributes="Bold"

MaxLines="1"

LineBreakMode="TailTruncation"

Text="Calif"/>

<Label Grid.Column="2"

FontAttributes="Bold"

Text="Origen:"/>

</Grid>

<CollectionView ItemsSource="{Binding Trips}">

<CollectionView.ItemsLayout>

<GridItemsLayout Orientation="Vertical"/>

</CollectionView.ItemsLayout>

<CollectionView.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectTrip2Command}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="2\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="4\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Text="{Binding StartDateLocal, StringFormat='{0:yyyy/MM/dd HH:mm}'}"/>

<Label Grid.Column="1"

Text="{Binding Qualification, StringFormat='{0:N2}'}"/>

<Label Grid.Column="2"

LineBreakMode="TailTruncation"

MaxLines="2"

Text="{Binding Source}"/>

<Image Grid.Column="3"

Source="ic\_more\_vert"/>

</Grid>

</DataTemplate>

</CollectionView.ItemTemplate>

</CollectionView>

</StackLayout>

<busyindicator:SfBusyIndicator AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource ColorDanger}"

HorizontalOptions="Center"

TextColor="White"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ContentPage>

Modificamos **MyTripsPageViewModel**

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using System;

using System.Collections.Generic;

using System.Linq;

using Taxi.Common.Helpers;

using Taxi.Common.Models;

using Taxi.Common.Services;

namespace Taxi.Prism.ViewModels

{

public class MyTripsPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private List<TripItemViewModel> \_trips;

private DelegateCommand \_refreshCommand;

public MyTripsPageViewModel(INavigationService navigationService, IApiService apiService)

: base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Mis Viajes";

StartDate = DateTime.Today.AddDays(-7);

EndDate = DateTime.Today;

LoadTripsAsync();

}

public DelegateCommand RefreshCommand => \_refreshCommand ?? (\_refreshCommand = new DelegateCommand(LoadTripsAsync));

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public List<TripItemViewModel> Trips

{

get => \_trips;

set => SetProperty(ref \_trips, value);

}

private async void LoadTripsAsync()

{

IsRunning = true;

string url = App.Current.Resources["UrlAPI"].ToString();

bool connection = await \_apiService.CheckConnectionAsync(url);

if (!connection)

{

IsRunning = false;

await App.Current.MainPage.DisplayAlert(

"Error",

"Revise su conexión a Internet",

"Aceptar");

return;

}

TokenResponse token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

UserResponse user = JsonConvert.DeserializeObject<UserResponse>(Settings.User);

MyTripsRequest request = new MyTripsRequest

{

EndDate = EndDate.AddDays(1).ToUniversalTime(),

StartDate = StartDate.ToUniversalTime(),

UserId = user.Id

};

Response response = await \_apiService.GetMyTrips(url, "api", "/Trips/GetMyTrips", "bearer", token.Token, request);

IsRunning = false;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Aceptar");

return;

}

List<TripResponse> trips = (List<TripResponse>)response.Result;

Trips = trips.Select(t => new TripItemViewModel(\_navigationService)

{

EndDate = t.EndDate,

Id = t.Id,

Qualification = t.Qualification,

Remarks = t.Remarks,

Source = t.Source,

SourceLatitude = t.SourceLatitude,

SourceLongitude = t.SourceLongitude,

StartDate = t.StartDate,

Target = t.Target,

TargetLatitude = t.TargetLatitude,

TargetLongitude = t.TargetLongitude,

TripDetails = t.TripDetails,

User = t.User

}).ToList();

}

}

}

Modificamos **MyTripPage**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:prism="http://prismlibrary.com"

xmlns:rating="clr-namespace:Syncfusion.SfRating.XForms;assembly=Syncfusion.SfRating.XForms"

xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

ios:Page.UseSafeArea="true"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="Taxi.Prism.Views.MyTripPage"

BackgroundColor="{StaticResource ColorBackground}"

Title="{Binding Title}">

<AbsoluteLayout>

<StackLayout AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="5">

<ScrollView>

<StackLayout >

<rating:SfRating HorizontalOptions="Center"

IsEnabled="False"

Margin="0,5"

Precision="Exact"

Value="{Binding Trip.Qualification}">

<rating:SfRating.RatingSettings>

<rating:SfRatingSettings RatedFill="{StaticResource ColorDanger}"/>

</rating:SfRating.RatingSettings>

</rating:SfRating>

<StackLayout HorizontalOptions="Center"

Orientation="Horizontal">

<Label FontAttributes="Bold"

FontSize="Large"

Text="Calificación"/>

<Label FontAttributes="Bold"

FontSize="Large"

Text="{Binding Trip.Qualification, StringFormat='{0:N2}'}"/>

</StackLayout>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

FontAttributes="Bold"

Text="Fecha Inicio}"/>

<Label Grid.Column="1"

Grid.Row="0"

Text="{Binding Trip.StartDateLocal, StringFormat='{0:yyyy/MM/dd HH:mm}'}"/>

<Label Grid.Column="0"

Grid.Row="1"

FontAttributes="Bold"

Text="Fecha Fin"/>

<Label Grid.Column="1"

Grid.Row="1"

Text="{Binding Trip.EndDateLocal, StringFormat='{0:yyyy/MM/dd HH:mm}'}"/>

<Label Grid.Column="0"

Grid.Row="2"

FontAttributes="Bold"

Text="Origen"/>

<Label Grid.Column="1"

Grid.Row="2"

Text="{Binding Trip.Source, StringFormat='{0:yyyy/MM/dd HH:mm}'}"/>

<Label Grid.Column="0"

Grid.Row="3"

FontAttributes="Bold"

Text="Destino"/>

<Label Grid.Column="1"

Grid.Row="3"

Text="{Binding Trip.Target, StringFormat='{0:yyyy/MM/dd HH:mm}'}"/>

<Label Grid.Row="4"

Grid.Column="0"

FontAttributes="Bold"

Text="Distancia recorrida"

VerticalOptions="Center"/>

<Label Grid.Row="4"

Grid.Column="1"

Text="{Binding Distance, StringFormat='{0:N2}'}"/>

<Label Grid.Row="5"

Grid.Column="0"

FontAttributes="Bold"

Text="Tiempo"/>

<Label Grid.Row="5"

Grid.Column="1"

Text="{Binding Time}"/>

<Label Grid.Row="6"

Grid.Column="0"

FontAttributes="Bold"

Text="Monto aproximado"/>

<Label Grid.Row="6"

Grid.Column="1"

Text="{Binding Value}"/>

<Label Grid.Column="0"

Grid.Row="7"

FontAttributes="Bold"

Text="Comentarios"/>

<Label Grid.Column="1"

Grid.Row="7"

Text="{Binding Trip.Remarks}"/>

</Grid>

<maps:Map x:Name="MyMap"

HeightRequest="500"

MapType="Street"/>

</StackLayout>

</ScrollView>

</StackLayout>

</AbsoluteLayout>

</ContentPage>

Modificamos el Code Behind **MyTripPage.xaml.cs**

using Taxi.Common.Models;

using Taxi.Prism.Helpers;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace Taxi.Prism.Views

{

public partial class MyTripPage : ContentPage

{

private static MyTripPage \_instance;

public MyTripPage()

{

InitializeComponent();

\_instance = this;

}

public static MyTripPage GetInstance()

{

return \_instance;

}

public void DrawMap(TripResponse trip)

{

if (trip.SourceLatitude != 0 && trip.SourceLongitude != 0)

{

Position position = new Position(trip.SourceLatitude, trip.SourceLongitude);

AddPin(position, trip.Source, "Inicio de viaje", PinType.Place);

MoveMap(position);

}

if (trip.TargetLatitude != 0 && trip.TargetLongitude != 0)

{

Position position = new Position(trip.TargetLatitude, trip.TargetLongitude);

AddPin(position, trip.Target, "Fin de viaje", PinType.Place);

MoveMap(position);

}

for (int i = 0; i < trip.TripDetails.Count - 1; i++)

{

Position a = new Position(trip.TripDetails[i].Latitude, trip.TripDetails[i].Longitude);

Position b = new Position(trip.TripDetails[i + 1].Latitude, trip.TripDetails[i + 1].Longitude);

DrawLine(a, b);

}

}

public void DrawLine(Position a, Position b)

{

if (Device.RuntimePlatform == Device.Android)

{

Polygon polygon = new Polygon

{

StrokeWidth = 10,

StrokeColor = Color.FromHex("#8D07F6"),

FillColor = Color.FromHex("#8D07F6"),

Geopath = { a, b }

};

MyMap.MapElements.Add(polygon);

}

else

{

AddPin(b, string.Empty, string.Empty, PinType.SavedPin);

}

MoveMap(b);

}

public void AddPin(Position position, string address, string label, PinType pinType)

{

MyMap.Pins.Add(new Pin

{

Address = address,

Label = label,

Position = position,

Type = pinType

});

}

private void MoveMap(Position position)

{

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position, Distance.FromKilometers(.5)));

}

}

}

Modificamos **MyTripPageViewModel**

using Prism.Navigation;

using Taxi.Common.Models;

using Taxi.Prism.Helpers;

using Taxi.Prism.Views;

namespace Taxi.Prism.ViewModels

{

public class MyTripPageViewModel : ViewModelBase

{

private TripResponse \_trip;

private double \_distance;

private string \_time;

private string \_value;

public MyTripPageViewModel(INavigationService navigationService)

: base(navigationService)

{

Title = "Mi Viaje";

}

public string Value

{

get => \_value;

set => SetProperty(ref \_value, value);

}

public double Distance

{

get => \_distance;

set => SetProperty(ref \_distance, value);

}

public string Time

{

get => \_time;

set => SetProperty(ref \_time, value);

}

public TripResponse Trip

{

get => \_trip;

set => SetProperty(ref \_trip, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

Trip = parameters.GetValue<TripResponse>("trip");

TripSummary tripSummary = GeoHelper.GetTripSummary(Trip);

Distance = tripSummary.Distance;

Time = $"{tripSummary.Time.ToString().Substring(0, 8)}";

if (tripSummary.Value == 5600)

{

Value = $"{tripSummary.Value:C0}";

}

else

{

Value = $"{tripSummary.Value:C0}";

}

MyTripPage.GetInstance().DrawMap(Trip);

}

}

}

# Hasta acá llegué