Pour affecter la table principale avec l'utilisateurs il faut inclure la proprieté:

ApplicationUser dans la classe;

-----BOOTSTRAP

ajouter le liens de la view créé dans le \_layout.cshtml et supprimer les inutiles;

création de la view Forme Create xxx ajouter la view depuis le contrôleur;

pour crée une forme le raccourci c'est :

"form>div.form-group\*5"

Ensuite remplacer le tag form html par le helpeur :

@using (Html.BeginForm("Create", "Gigs"))

classique il faut ajouter les helpeurs entre les div:

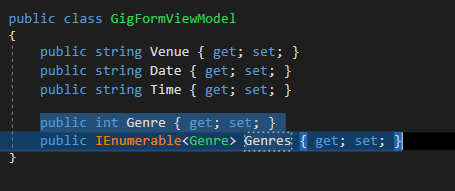
@Html.LabelFor(m=>m.Venue)

@Html.TextBoxFor(m=>m.Venue ,new {@class="form-control"})

pour convertir le DateTime sur les Forme il faut créé un ViewModel et ajouter les proprietés string Time et string Date

ensuite sur la view changer le @model de la pase par la view model

pour un dropdownlist il faut une propriété dans la viewModel en INT et une liste exemple

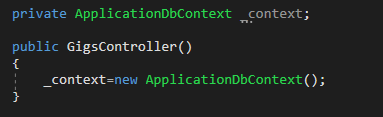


ensuite dans la view :

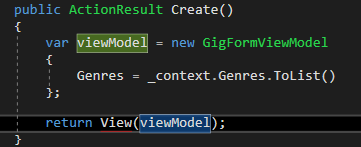
@Html.DropDownListFor(m => m.Genre, new SelectList(Model.Genres, "Id", "Name"),

           "", new { @class = "form-control" })

Il faut toujours initialiser le ADBC context Private READONLY !!! dans chaque contrôleur



Et ensuite pour que le dropdownlist marche il faut faire un lien entre le viewmodel et la table en question



Ensuite ajouter un bouton :

<button type="submit" class="btn btn-info">Sauvegarder</button>

Sauvegarder une FOrme:

[Authorize]

       [HttpPost]

       public ActionResult Create(GigFormViewModel viewModel)

       {

           var userId = User.Identity.GetUserId();

           var artist = \_context.Users.Single(u => u.Id == userId);

           var genre = \_context.Genres.Single(g => g.Id == viewModel.Genre);

           var gig = new Gig

           {

               Artist = artist,

               DateTime = DateTime.Parse(string.Format("{0} {1}", viewModel.Date, viewModel.Time)),

               Genre = genre,

               Venue = viewModel.Venue

           };

           \_context.Gigs.Add(gig);

           \_context.SaveChanges();

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

       }

Fluent API

Pour changer les noms des FK il faut ajouter required :

public GigConfiguration()

       {

           Property(p => p.Venue)

               .IsRequired()

               .HasMaxLength(255);

           Property(p => p.GenreId)

               .IsRequired();

           Property(p => p.ArtistId)

               .IsRequired();

public class Gig

    {

        public int Id { get; set; }

        public DateTime DateTime { get; set; }

        public string Venue { get; set; }

        public string ArtistId { get; set; }

        public ApplicationUser Artist { get; set; }

        public byte GenreId { get; set; }

        public Genre Genre { get; set; }

    }

La propriété GenreId e tArtistId n’existe pas dans la class du coup impossible de l’affecter dans l’action du contrôleur est pour cette raison qu’il faut l’affecter dans la classe et ensuite dans Fluent isRequired !!!

Et ensuite après refactoring de la ActionResult on obtiens :

[Authorize]

        [HttpPost]

        public ActionResult Create(GigFormViewModel viewModel)

        {

            var gig = new Gig

            {

                ArtistId = User.Identity.GetUserId(),

                DateTime = DateTime.Parse(string.Format("{0} {1}", viewModel.Date, viewModel.Time)),

                GenreId = viewModel.Genre,

                Venue = viewModel.Venue

            };

            \_context.Gigs.Add(gig);

            \_context.SaveChanges();

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

        }

Pour finir le refactoring il faut enlever le parse de datetime du contrôleur et le déplacer dans la ViewModel :

 public class GigFormViewModel

    {

        public string Venue { get; set; }

        public string Date { get; set; }

        public string Time { get; set; }

        public byte Genre { get; set; }

        public IEnumerable<Genre> Genres { get; set; }

        public DateTime DateTime

        {

            get { return DateTime.Parse(string.Format("{0} {1}", Date, Time)); }

        }

    }

}

Ensuite il faut transformer le parse en Methode !!! si non un msg d’erreur de null :

public DateTime GetDateTime()

       {

           return DateTime.Parse(string.Format("{0} {1}", Date, Time));

       }

Et dans le ActionResult

DateTime=viewModel.DateTime

var gig = new Gig

            {

                ArtistId = User.Identity.GetUserId(),

                DateTime = viewModel.GetDateTime(),

                GenreId = viewModel.Genre,

                Venue = viewModel.Venue

            };

Pour la validation coté serveur il faut mettre Required pour les propriétés de la viewModel

public class GigFormViewModel

   {

       [Required]

       public string Venue { get; set; }

       [Required]

       public string Date { get; set; }

       [Required]

       public string Time { get; set; }

       [Required]

       public byte Genre { get; set; }

       public IEnumerable<Genre> Genres { get; set; }

       public DateTime DateTime

       {

           get { return DateTime.Parse(string.Format("{0} {1}", Date, Time)); }

       }

   }

Ensuite dans la View il décoré avec le helpeur validationMessageFor :

@Html.LabelFor(m => m.Venue)

        @Html.TextBoxFor(m => m.Venue, new { @class = "form-control" })

        @Html.ValidationMessageFor(m=>m.Venue)

Et dans le controleur il faut verifier le !ModelState.isvalid :

public ActionResult Create(GigFormViewModel viewModel)

        {

            if (!ModelState.IsValid)

                return View("Create", viewModel);

si n’est pas valide retourne sur la Forme et ne perdant pas les données

Ici pour eviter un msg. De null il faut remplir le Genre avec la base comme ceci

if (!ModelState.IsValid)

            {

                viewModel.Genres = \_context.Genres.ToList();

                return View("Create", viewModel);

            }

Pour la verification des Date futures il faut cree une classe :

public class FutureDage : ValidationAttribute

    {

        public override bool IsValid(object value)

        {

            return base.IsValid(value);

        }

    }

Et override IsValid et appliquer dans le viewModel précèdent comme annotation :

[Required]

        [FutureDate]

        public string Date { get; set; }

ensuite il faut verifier si la date est inferieure :

public override bool IsValid(object value)

        {

            DateTime dateTime;

            var isValid = DateTime.TryParseExact(Convert.ToString(value),

                "dd/MM/yyyy",

                CultureInfo.CurrentCulture,

                DateTimeStyles.None,

                out dateTime);

            return (isValid && dateTime > DateTime.Now);

        }

Et pour le temps :

[Required]

       [ValidTime]

       public string Time { get; set; }

public class ValidTime : ValidationAttribute

    {

        public override bool IsValid(object value)

        {

            DateTime dateTime;

            var isValid = DateTime.TryParseExact(Convert.ToString(value),

                "HH:mm",

                CultureInfo.CurrentCulture,

                DateTimeStyles.None,

                out dateTime);

            return (isValid);

        }

    }

Faire la validation coté client pour optimisation éviter les allez retour serveur :

Il faut ajouter sur la page view le script suivant

La validation de la date et heure va disparaitre apres la validation coté client .

@section scripts

{

    @Scripts.Render("~/bundles/jqueryval")

}

Pour conturner les attack il faut ajouter le code suivant dans la view :

@Html.AntiForgeryToken()

Et dans le controleur :

[ValidateAntiForgeryToken]

       [Authorize]

       [HttpPost]

       public ActionResult Create(GigFormViewModel viewModel)

       {

           if (!ModelState.IsValid)

           {

upload une image pendant l’enregistrement d’une annonce dans une Forme :

Ajoute d’une collonne PicUrl varchar(max) not null ;

Cree un controleur et une action AddNewProduct

sur la Forme il créé un input :

<input id= « ImageUpload » name= « ImageUpload » class= « form-control » type= « file » />

le JS

function AjaxPost(formData)