Model

Representa as entidades presentes na base de dados, neste momento o projecto ainda inclui as sample classes anteriores, são elas:

Attendance, Person, Room, Session, TimeSlot, Track

Meti na drop o ficheiro CodeCamper.sdf, é um ficheiro db SQL Server CompactEdition que pode ser aberto com o programa SQL Compact Query Analyzer (aconselho a sacares para veres os campos de cada tabela, e de que maneira se relacionam com as classes C#).

Relativamente ao HortiHoje, estas são as nossas entidades:

Activity, FieldNote, **FieldNoteReporter**, Location, MediaFile, **MediaFileTag**, Reporter, Tag, Task, **TaskAllowedReporter**, **TaskAllocatedReporter**

As entidades a negrito representam ligações N-para-N, e como tal têm um significado especial para as tabelas que referenciam.

Para cada uma destas entidades tem que ser criada uma classe C# com esse nome **exacto**, esta parte é mais importante do que o que parece, observa a convenção de nomes não só nas classes mas também nos seus atributos e segue-a à risca.

A construção destas classes parte primeiro dos atributos da sua respectiva entidade na base de dados, estes atributos têm ainda que ser compatíveis da base de dados para as classes, nvarchar para string por exemplo, segue esta convenção.

No caso da Location por exemplo:

```
id   : int
lat  : nvarchar(25)
long : nvarchar(25)
```

A sua classe C# fica:

```
public class Location
{

    public int Id { get; set; }

    [Required, MaxLength(25)]
    public string Lat { get; set; }

    [Required, MaxLength(25)]
    public string Long { get; set; }
}
```

As anotações [] pertencem ao System.ComponentModel.DataAnnotations, e são utilizadas por vários componentes no cliente e no servidor, são utilizadas para validar os modelos quer no cliente quer no servidor, no entanto, **não são obrigatórias**, mas se puderes escreve pelo menos estas(onde fizer sentido):

- **Required** – Indicates that the property is a required field
- **MaxLength** – Defines a maximum length for a string field
- **Range** – Gives a maximum and minimum value for a numeric field

Podes observar estas utilizações nas classes sample, como a `Attendance` e `Person`

No caso de uma entidade com uma **foreign key**, como a `Activity`

```
id          : int
description : text
name        : nvarchar(50)
idManager   : int
```

C#:

```
public class Activity
{
    public int Id { get; set; }
    [Required, MaxLength(50)]
    public string Name { get; set; }
    public string Description { get; set; }
    public int IdManager { get; set; }

    public virtual Reporter Reporter { get; set; }

    public virtual ICollection<Task>
        TaskList
    { get; set; }
}
```

Neste caso temos dois atributos adicionais, `Reporter e TaskList,` com os seus tipos respectivos, `Reporter` , e uma colecção de `Task`, duas classes da nossa base de dados que eu já criei.

Podes definir atributos destes quando “tens maneira de chegar a esse elemento”, neste caso podemos incluir uma referência a um `Reporter` porque temos o seu id(**idManager**).

Estas ligações não são imediatas, têm que ser explicitadas no módulo **DataAccess**.

São bastante úteis porque significa que não tens que fazer mais queries, se tens uma actividade e queres saber o nome do manager que a inseriu não tens que fazer uma segunda query SELECT * FROM Reporter WHERE id = idManager

No caso da TaskList podemos incluir uma referência a uma colecção de `Task` porque temos uma relação **1** para **N** da tabela Activity para a Task. Novamente, esta relação não é imediata, é definida no **DataAccess**.

No caso de tabelas **N** para **N** penso que o mais fácil é olhares para as classes Attendance e Person, e ainda dentro da Person, o atributo SpeakerSessions.

DataAccess

Tens 2 ficheiros relevantes, DbContext e Repository.
DbContext tem todas as configurações da ligação à base de dados, assim como as funções que devolvem a informação num formato DbSet.

Repository exporta a informação sob forma de funções, é aqui que o cliente liga (através do controlador Breeze, se quiseres podes ver esse comportamento no ficheiro BreezeController.cs no módulo **Web**).

Só precisas de saber que ao adicionares uma nova classe Model tens que fazer 3 alterações no módulo DataAccess:

1. Criar uma nova classe de configuração, se estiveres a adicionar o modelo `MediaFile` por exemplo, `MediaFileConfiguration`

   Nessa classe de configuração precisas de definir o relacionamento entre essa entidade e as suas relacionadas, faz copy paste do esqueleto de uma já existente e altera o código no constructor.

   Esse código não é bem C#, é Fluent API da EntityFramework, aconselho-te **vivamente** a dares uma olhada por este tutorial (o código referido nos exemplos está no fundo da página).

   Algumas entidades poderão não precisar de configuração, como por exemplo a `Tag` , mas cria na mesma o ficheiro Configuration e segue os restantes passos, se mais tarde nos apercebermos que precisamos de configurar ao menos já está tudo feito e só basta escrever o código Fluent.

2. Vais ao DbContext e na função OnModelCreating adicionas a tua entrada ao modelBuilder.configurations

   ```
   modelBuilder.Configurations.Add(new MediaFileConfiguration());
   ```

   De seguida, onde tens as `// HortiHoje Entities` adicionas a tua entrada:

   ```
   public DbSet<MediaFile> MediaFiles { get; set; }
   ```

   Toma atenção à diferença entre a função MediaFiles e o nome da classe modelo `MediaFile`, e ainda como uma está no plural e a outra em singular, isto é **obrigatório.**

3. No Repository adicionas a tua entrada nas `// HortiHoje Entities`

   ```
   public IQueryable<MediaFile> MediaFiles
   {
       get { return Context.MediaFiles; }

   }
   ```

Nenhuma destas alterações quer no Model quer no DataAccess deve produzir resultados visíveis no website. No entanto, ao executares a aplicação se acederes ao link http://localhost:xxxxx/breeze/breeze/metadata

Deves conseguir fazer CTRL+F e procurar pelo modelo que adicionaste, deves conseguir encontrar referências a esse modelo quer no HortiHoje.Model quer no HortiHoje.DataAccess.

Deves ainda conseguir ver os seus atributos, e os seus relacionamentos com outras tabelas.

Sei que não é propriamente legível mas uma vez que as views não estão construídas não há maneira de observar esses modelos em uso.