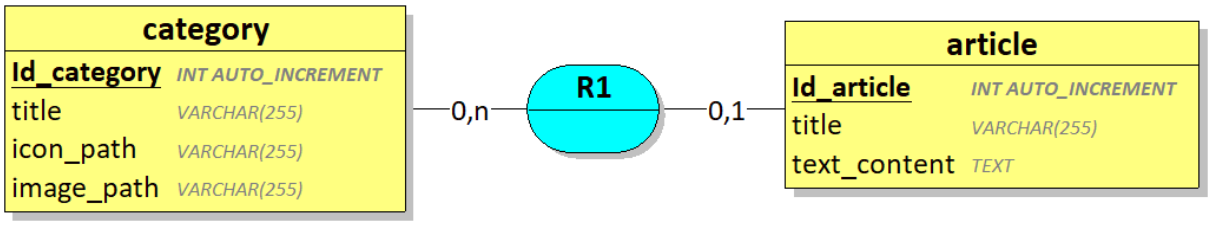
TD Symfony

Étape 6 : Création d'entités

Dans l'étape précédente, nous avons codé un premier template (page d'accueil) en dur. Nous allons voir comment y intégrer des données pour la rendre dynamique.

La page d'accueil doit afficher les catégories d'article ainsi que les derniers articles publiés. Nous allons donc créer 2 entités qui correspondront à 2 tables en base de données.

MCD première version



Nous allons utiliser Maker pour générer nos entités. Tapez la ligne de commande suivant dans le terminal VSCode

```
\td_symfony> php bin/console make:entity
```

Choisissez le nom de l'entité : Category

```
Class name of the entity to create or update (e.g. VictoriousElephant):
> Category

created: src/Entity/Category.php
created: src/Repository/CategoryRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.
```

2 classes ont été créées : src/Entity/Categroy et src/Entity/CategoryRepository. Allez y jeter un coup d'œil avant de continuer … Il reste à créer les champs de notre entité (colonnes de la table en base de donnée)

```
New property name (press <return> to stop adding fields):
> title
```

Par défaut, le type des champs est string (varchar), la longueur est 255 et le champs est non-nullable. Tapez enter pour garder ces valeurs. Les 3 champs title, icon_path et image_path sont de type : string, 255, Not Null

```
Field type (enter ? to see all types) [string]:
>

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
>
```

```
updated: src/Entity/Category.php

Add another property? Enter the property name (or press <return> to stop adding fields):
```

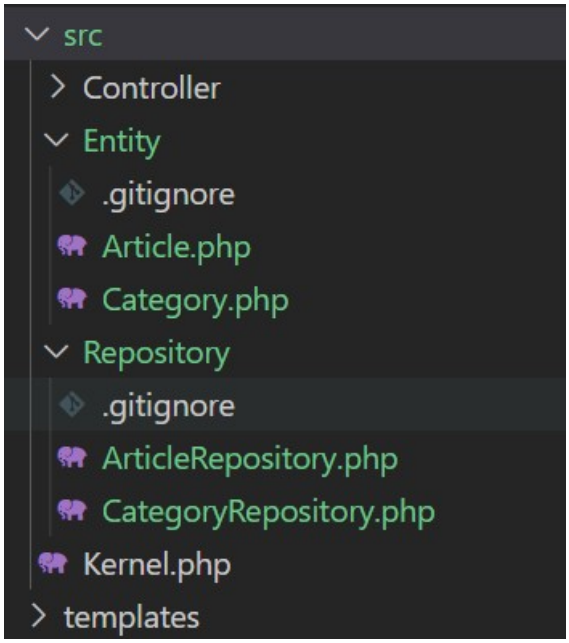
A chaque champ ajouté, notre entité Category (src/Entity/Category) est mise à jour. Une fois tous nos champs créés, tapez ENTER pour terminer la création de l'entité.

```
updated: src/Entity/Category.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>


 Success! 

Next: When you're ready, create a migration with php bin/console make:migration
```

Les champs ne seront pas toujours de type string 255 Not Null, si vous tapez « ? » en réponse à la question « Field type » vous pouvez visualiser tous les types possibles (et les relations)

```
 Field type (enter ? to see all types) [string]:
 > ?

Main types
  * string
  * text
  * boolean
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard will help you build the re
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval

Other Types
```

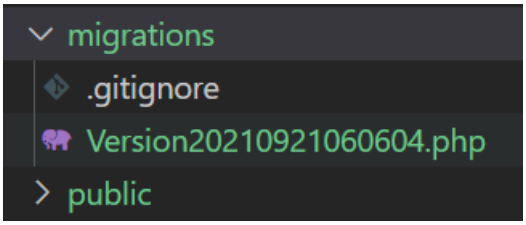
Créons la 2ème entité Article (incomplète pour l'instant, elle ne contient qu'un titre et un texte)



Il nous reste à créer les tables en base de données. Nous utilisons Maker pour générer un fichier de migration

```
\td_symfony> php bin/console make:migration
```

```
Success!

Next: Review the new migration "migrations/Version20210921060604.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```



Puis nous utilisons doctrine (ORM) pour exécuter cette migration (créer les 2 tables en base de données)

```
\td_symfony> php bin/console doctrine:migrations:migrate
```

```
 WARNING! You are about to execute a migration in database "td_symfony_db" that could
ata loss. Are you sure you wish to continue? (yes/no) [yes]:
 >

[notice] Migrating up to DoctrineMigrations\Version20210921060604
[notice] finished in 385.1ms, used 20M memory, 1 migrations executed, 2 sql queries
```

Dans phpMyAdmin :

Ajoutons la relation OneToMany/ManyToOne entre category et article

```
PS C:\Users\LAURENT\Workspace\td_symfony> php bin/console make:entity Article

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > category

 Field type (enter ? to see all types) [string]:
 > ManyToOne

 What class should this entity be related to?:
 > Category

 Is the Article.category property allowed to be null (nullable)? (yes/no) [yes]:
 >

 Do you want to add a new property to Category so that you can access/update Arti
y->getArticles()? (yes/no) [yes]:
 >

 A new property will also be added to the Category class so that you can access t

 New field name inside Category [articles]:
 >

 updated: src/Entity/Article.php
 updated: src/Entity/Category.php
```

Nos entités ont été modifiées.

Réaliser ensuite la migration :

```
\td_symfony> php bin/console make:migration
```

Puis la modification de la base :

```
\td_symfony> php bin/console doctrine:migrations:migrate
```

Doctrine à ajouté la contrainte de clé étrangère dans la table article.

Dans notre classe Entity/Article nous avons accès directement à la category

src > Entity > Article.php > Article

```
29
30      /**
31       * @ORM\ManyToOne(targetEntity=Category::class, inversedBy="articles")
32       */
33      private $category;
```

src > Entity > Article.php > Article

```
64      public function getCategory(): ?Category
65      {
66          return $this->category;
67      }
68
69      public function setCategory(?Category $category): self
70      {
71          $this->category = $category;
72
73          return $this;
74      }
```

La relation inverse (OneToMany) entre category et article est également créée dans la classe Entity/Category

src > Entity > Category.php > Category

```
37      /**
38       * @ORM\OneToMany(targetEntity=Article::class, mappedBy="category")
39       */
40      private $articles;
41
42      public function __construct()
43      {
44          $this->articles = new ArrayCollection();
45      }
```

src > Entity > Category.php > Category

```php
    /**
     * @return Collection|Article[]
     */
    public function getArticles(): Collection
    {
        return $this->articles;
    }

    public function addArticle(Article $article): self
    {
        if (!$this->articles->contains($article)) {
            $this->articles[] = $article;
            $article->setCategory($this);
        }

        return $this;
    }

    public function removeArticle(Article $article): self
    {
        if ($this->articles->removeElement($article)) {
            // set the owning side to null (unless already changed)
            if ($article->getCategory() === $this) {
                $article->setCategory(null);
            }
        }

        return $this;
    }
```