Appunti 13 Maggio

Il costruttore serve per "instanziare" l'oggetto nella classe, le informazioni inserite all'interno del construct sono obbligatorie quando si crea l'oggetto

```
public function __construct($_name, $_price)
{
$this→name = $_name;
$this→price = $_price;
}
```

da index.php includo le classi con

```
require o require_once __DIR__ . '/classes/product.class.php';
```

Dopo una funzione si può aggiungere una "firma" postponendo i due punti

```
public function->getPrice(): string (o float) (o int)
{
return $this->price;
}
```

Per includere una nuova classe che include quella precedente

```
class Son extends Father
```

Trait

I trait possono esser usati per creare proprietà da applicare a più classi

```
trait Position
{
  public $lat;
  public $lng;

  public function getAddress()
```

```
  {
  return $address;
  }
}
```

il trait si collega con use

```
class Son extends Father
{
  use Position;

  public $name;
  public $lastname;
}
```

per includere il file trait nell'index di usa il require.

Più trait si concatenano con la virgola

```
use Position, Info;
```

Il controller

E' alla base del pattern MVC. E' un file php che viene richiamato dal file Routes, si occuperà di prendere i dati, gestirli e restituirli sotto forma di view o altro all'utente. Si può creare un controller tramite terminale con il comando

```
php artisan make:controller NomeController
```

che crea un file in app/http/controllers.

Per convenzione i nomi dei controller sono al singolare e scritti in PascalCase.

esempio

```
Route::get('/home', 'HomeController@index');
```

La rotta avrà come url /home, gestita dall'HomeController, più precisamente dalla public function index() all'interno di HomeController.

```
public function index()
{
  return view('home.index');
}
```

Database su Laravel

Andare su PhpMyAdmin e creare un nuovo database vuoto.

Su laravel si usa il comando da terminale

```
php artisan make:migration nomeDellaMigration
```

Così si crea in automatico il nome della tabella, creerà l'ID e aggiungerà altri dati:

```
php artisan make:migration create_users_table
```

Se volessimo aggiornare una tabella potremo usare questo nome che aiuterà laravel a capire quale tabella modificare:

```
php artisan make:migration update_users_table
--table=users
```

Schema è la classe che si occupa di creare la tabella, il metodo create() crea la tabella, id e timestamps sono colonne aggiunte in automatico da laravel

```
Schema::create('nomeTabella', function( Blueprint $table)
{
  $table->id();
  $table->timestamps();
}
```

Per creare una nuova colonna basta aggiungere:

```
$table->string('nomeColonna', 100);
```

con nome della colonna e numero di caratteri massimi che può accettare.

Per tutte gli altri dati accettati consultare la tabella all'indirizzo:

https://laravel.com/docs/7.x/migrations#columns

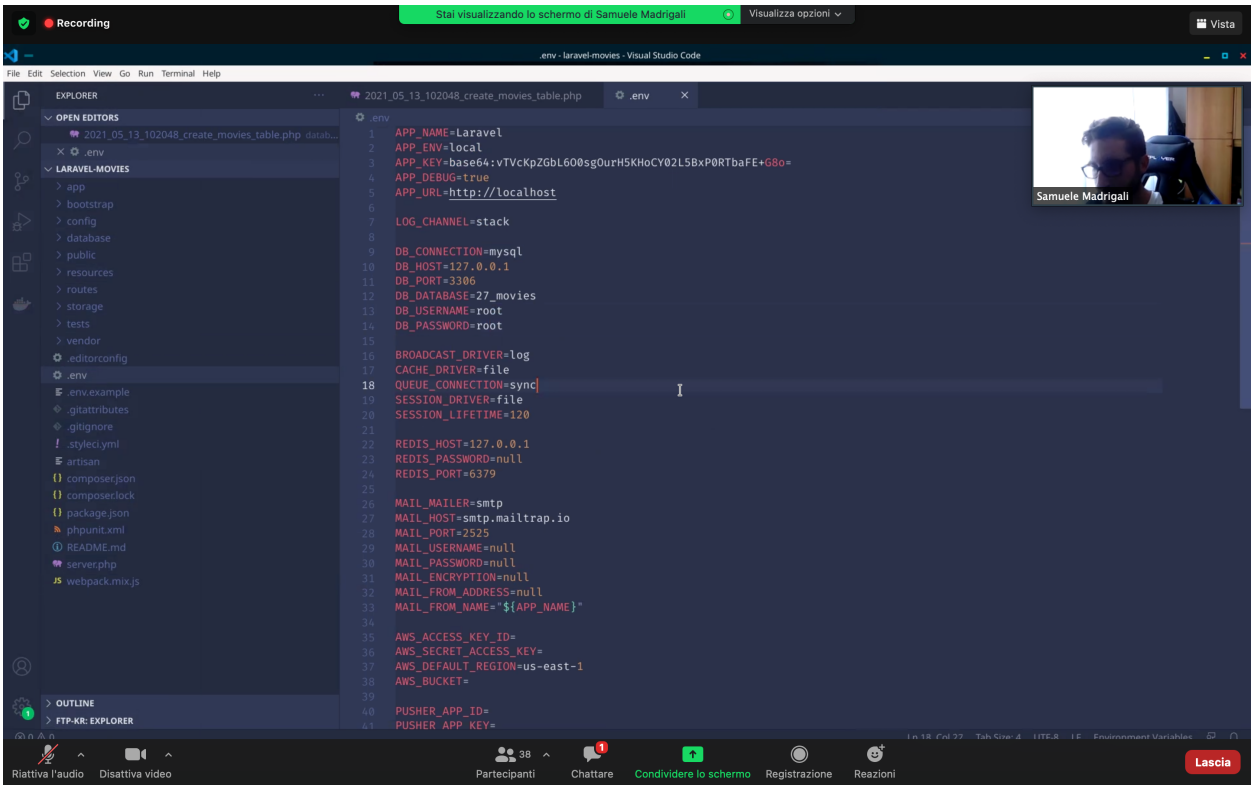
Una volta create tutte le colonne per eseguire tutte le migration ancora non eseguite da terminale si può lanciare il comando da terminale

```
php artisan migrate
```

per tornare indietro invece si può fare

```
php artisan migrate:rollback
```

Ma prima bisogna andare nel file .env e connettere il database.



Terminata la modifica del file .env bisogna lanciare il comando

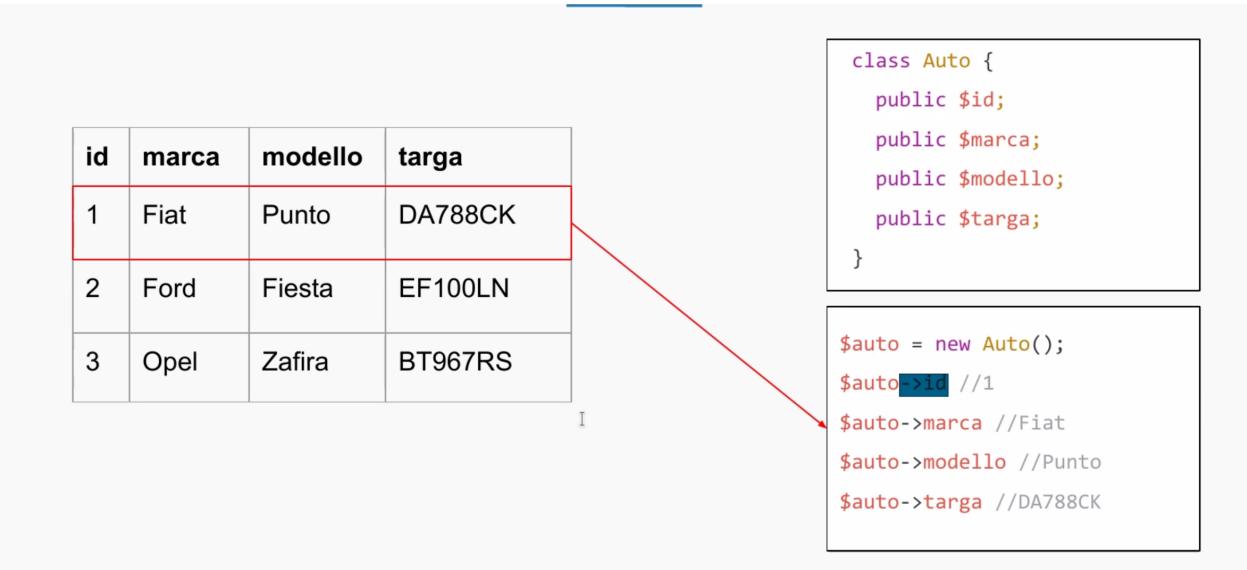
```
composer dump-autoload
```

Dopo aver aggiunto i dati al database che ci servono si può andare a collegare l'home al controller.

Il Model

Per collegarci al nostro database utilizzeremo una classe che funziona da interfaccia agli oggetti: l'ORM (Object Relationship Mapping).

Il modello sarà di questo tipo:



L'ORM di Laravel è Eloquent.

Il modello si crea col comando da terminale:

```
php artisan make:model MyModel
```

Dove MyModel è il nome del model, per convenzione è al singolare. Il model sarà nella cartella App.

Supponendo di avere una tabella movies e un model Movie si possono leggere i dati andando nel movie controller e importare la classe

```
use app\Movie;
```

e usando

```
$movie = Movie::all();
```

prendiamo tutto ciò che si trova all'interno della tabella Movies.