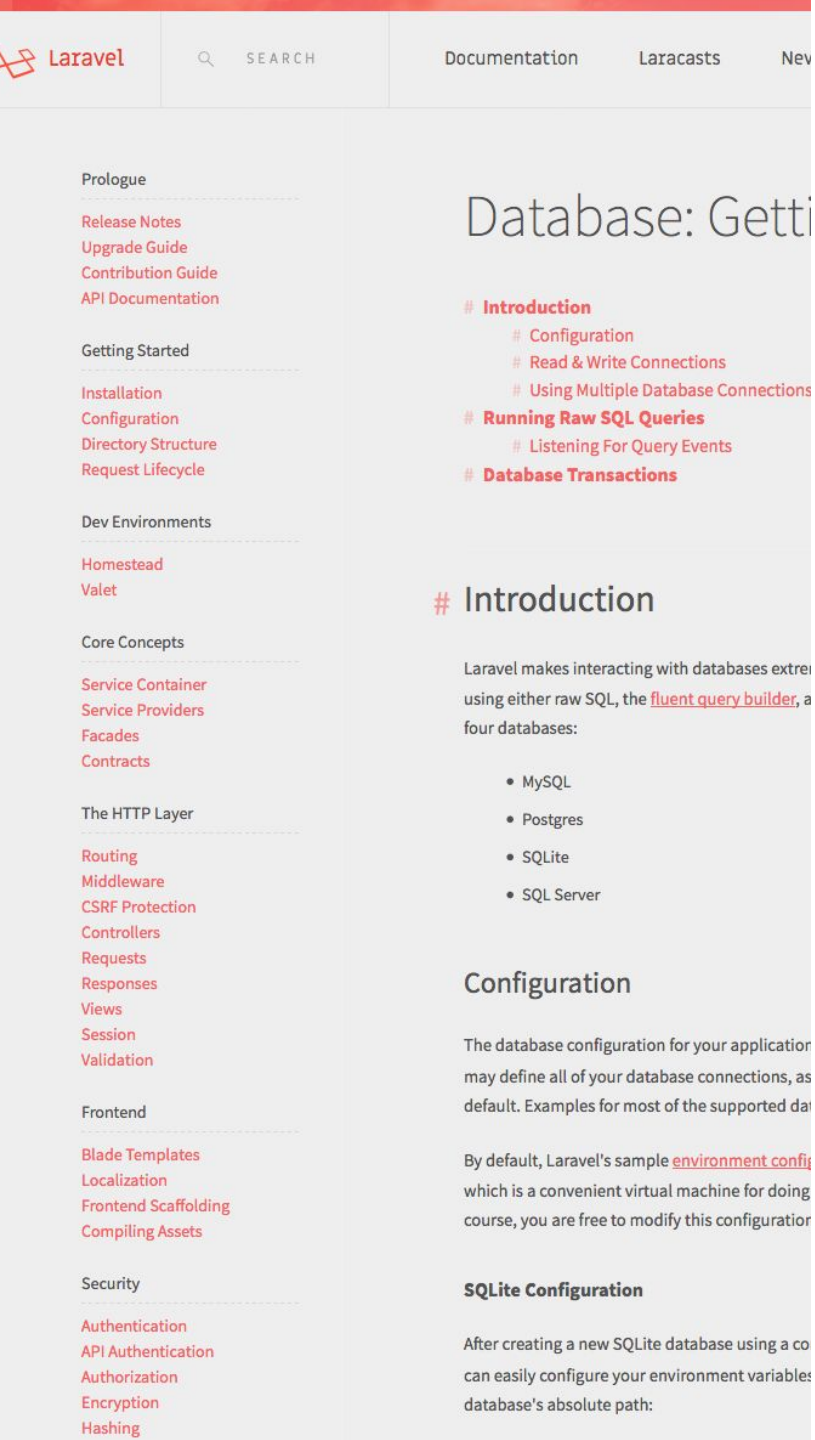


LARAVEL: ACCESO A DATOS

DISEÑO DE SISTEMAS SOFTWARE

Contenido

1. Configuración de la base de datos
2. Migraciones
3. Schema Builder
4. Query Builder
5. Database Seeding
6. Ejercicio

Configuración de un proyecto

- Laravel obtiene la mayor parte de su configuración de variables de entorno
- La carpeta `config/` contiene los scripts que cargan la configuración en memoria a partir de las variables de entorno cuando se inicia la aplicación, proporcionando un valor por defecto cuando no se encuentran:

```
'debug' => env('APP_DEBUG', false)
```

- Para facilitar la configuración y unificar entornos de desarrollo se pueden definir las variables de entorno en el archivo `.env` en la carpeta raíz:

```
APP_DEBUG=true
```

Configuración de la base de datos

- Laravel permite utilizar MySQL, PostgreSQL, SQLite y SQL Server
- La configuración de acceso a la BBDD está en el fichero `config/database.php`

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    ...
```

Configuración de la base de datos

- El fichero de configuración obtiene los valores de variables de entorno. Para mayor comodidad, Laravel permite especificar estos valores en el fichero "`.env`" en la carpeta raíz del proyecto

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=dss
DB_USERNAME=dss
DB_PASSWORD=dss
```

En los laboratorios se ha creado una base de datos "dss", a la que podéis acceder con usuario "dss" y contraseña "dss"

Configuración de la base de datos

- Modifica también el fichero **phpunit.xml** en la raíz del proyecto para que las pruebas automatizadas puedan acceder a la base de datos

```
    ...
    <server name="DB_CONNECTION" value="mysql"/>
    <server name="DB_DATABASE" value="dss"/>
    ...
```

- Puedes usar otra base de datos si quieres evitar que los tests modifiquen tu base de datos de desarrollo

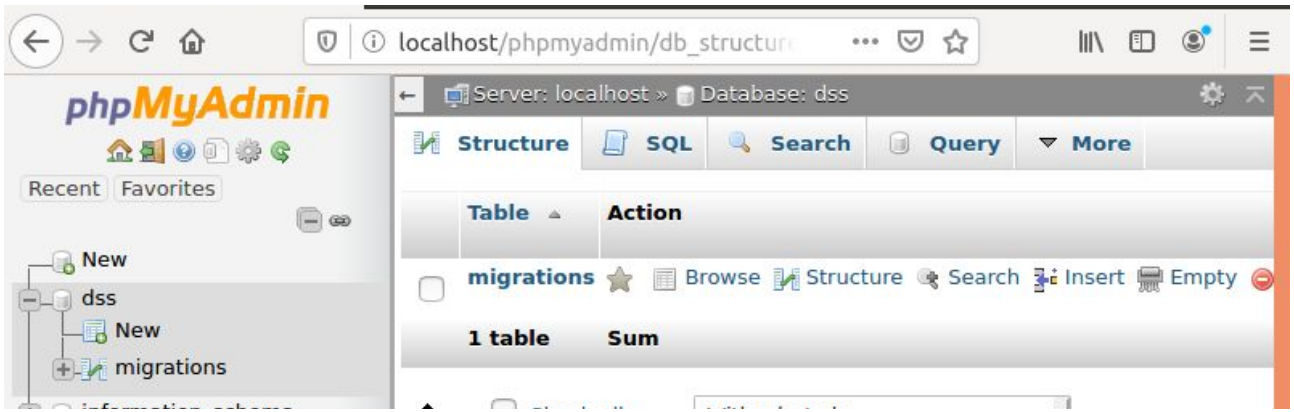
Inicialización de la base de datos

- Para inicializar la base de datos ejecutamos el comando de Artisan:

```
$ php artisan migrate:install
```

- Esto creará la tabla de migraciones en la base de datos, lo podemos comprobar usando phpMyAdmin en la URL:

http://localhost/phpmyadmin



Laravel: Acceso a datos

MIGRACIONES

Migraciones

- Sistema de control de versiones para la estructura de la BBDD
- Guardan un histórico de cambios y estado actual de la estructura
- Son archivos PHP guardados en la carpeta `database/migrations`
- Para cada tabla o cambio que queramos hacer en la BBDD creamos una migración, de esta forma se irá guardando un histórico
- Además podremos deshacer los cambios (rollback)

https://laravel.com/docs/6.x/migrations

Crear migraciones

- Para crear una nueva migración se utiliza la opción de Artisan `migrate:make`, por ejemplo, para una nueva tabla `products`:

```
$ php artisan make:migration create_products_table --create=products
```

Esto creará el fichero `database/migrations/<TIMESTAMP>_create_products_table.php`

- Para crear una migración que modifica una tabla ya existente se usa la opción `--table=nombre_tabla`

```
php artisan make:migration add_price_to_products_table --table=products
```

Crear migraciones

- Los comandos anteriores crean un archivo de migración con una estructura preparada para trabajar con la tabla indicada

```php
class CreateProductsTable extends Migration
{
    // Lanza la migración
    public function up() {
        Schema::create('products', function (Blueprint $table) {
            // Define la estructura de la tabla con Schema Builder
        });
    }
    // Deshace la migración
    public function down() {
        Schema::dropIfExists('products');  // Elimina la tabla
    }
}
```

Lanzar y deshacer migraciones

- Para lanzar ejecutar las últimas migraciones utilizamos:

```
$ php artisan migrate
```

- Para deshacer la última migración:

```
$ php artisan migrate:rollback
# O para deshacer todas las migraciones:
$ php artisan migrate:reset
```

- O para deshacer todas las migraciones y volver a lanzarlas:

```
$ php artisan migrate:fresh
```

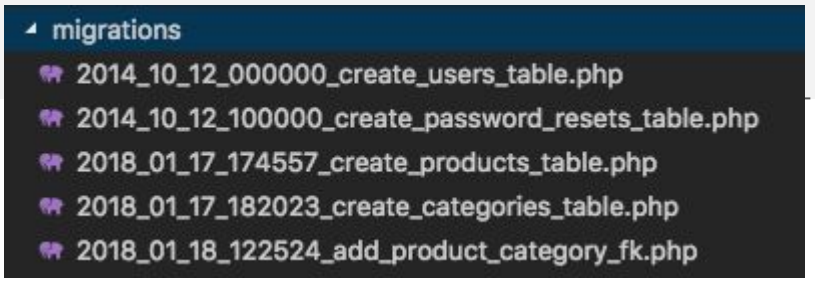
- También podemos comprobar el estado actual de las migraciones:

```
$ php artisan migrate:status
```

Orden de las migraciones

- Es importante no modificar migraciones anteriores para modificar la estructura de una tabla, de lo contrario puede haber errores al hacer un `rollback` de la base de datos
- Si se quiere añadir una clave ajena a otra tabla que se ha creado posteriormente, se debe crear una nueva migración que modifique la tabla para añadir la clave ajena

```php
public function up() {
  Schema::table('products', function (Blueprint $table) {
    $table->bigInteger('category_id')->unsigned();
    $table->foreign('category_id')->references('id')->on('categories');
  });
}
```



Laravel: Acceso a datos

SCHEMA BUILDER

Crear tablas

- Schema se utiliza de forma conjunta con las migraciones.
- Permite crear las tablas en el método `up` de la migración, por ejemplo para crear la tabla `users`:

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Eliminar tablas

- Y para eliminar la tabla `users` en el método `down`:

```
    // Deshace la migración
    public function down() {
        Schema::dropIfExists('users');  // Elimina la tabla
    }
```

Modificar tablas

- Con las migraciones también se puede modificar la estructura de una tabla existente:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('address');
    });
}

public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('address');
    });
}
```

Comando	Equivalencia en base de datos
$table->bigIncrements('id');	Autoincremental UNSIGNED BIGINT (**clave primaria**)
$table->unsignedBigInteger('votes');	UNSIGNED BIGINT (**usado para claves ajenas**)
$table->boolean('confirmed');	BOOLEAN
$table->date('created_at');	DATE
$table->dateTime('created_at', 0);	DATETIME con precisión opcional (dígitos decimales)
$table->double('amount', 8, 2);	DOUBLE con precisión (total de dígitos) y escala (dígitos decimales)
$table->float('amount', 8, 2);	FLOAT con precisión (total de dígitos) y escala (dígitos decimales)
$table->integer('votes');	INTEGER
$table->string('name', 100);	VARCHAR con **longitud opcional** (default=255)
$table->text('description');	TEXT
$table->time('sunrise', 0);	TIME con precisión opcional (dígitos decimales)
$table->timestamp('added_on', 0);	TIMESTAMP con precisión opcional (dígitos decimales)
->nullable($value = true)	Permite (o no) valores NULL **Las columnas que no tengan este modificador son obligatorias por defecto**
->default($value)	Especifica un valor por defecto para la columna

Tipos de datos

- Schema Builder se encarga de adaptar los tipos de datos al motor de BBDD que se esté utilizando

- Hay más opciones además de las mostradas en la tabla anterior, puedes encontrarlas todas en la siguiente dirección:

https://laravel.com/docs/6.x/migrations#creating-columns

Índices

- También permite añadir índices a los campos de una tabla.
- Podemos crearlos después de definir un campo, por ejemplo con:

```
$table->primary('id');     // Añadir una clave primaria
$table->primary(['first', 'last']);   // Primaria compuesta
$table->unique('email');   // Definir el campo como UNIQUE
$table->index('state');    // Añadir un índice a una columna
```

- O añadirlos a la vez que se crea el campo, por ejemplo:

```
$table->string('email')->unique();
```

- **IMPORTANTE:** al usar `$table->bigIncrements('id')` ya se crea una **clave principal** del tipo UNSIGNED BIG INTEGER auto-incremental.

Claves ajenas

- Para crear una clave ajenas utilizamos `foreign(...)->references(...)->on(...)`, de la forma:

```
$table->unsignedBigInteger('user_id');
$table->foreign('user_id')->references('id')->on('users');
```

- Es importante crear primero el campo de la referencia
- Podemos indicar que hacer en el `onDelete` on en `onUpdate`:

```
$table->foreign('user_id')->references('id')->on('users')
    ->onDelete('cascade');
```

- Para eliminar una clave ajena en el método "`down`" hacemos:

```
$table->dropForeign('posts_user_id_foreign');
// Siguiendo el patrón de nombre: <tabla>_<columna>_foreign
```

Laravel: Acceso a datos

DATABASE SEEDING

Database Seeding

- Permite la inserción de datos iniciales en la base de datos
- Muy útil para realizar pruebas en desarrollo o para rellenar tablas que ya tengan que contener datos inicialmente
- Los ficheros de semillas se encuentra en la carpeta `database/seeds`
- El método `run` de la clase `DatabaseSeeder` es el primero que se llama, y desde el cual podemos:
  - Ejecutar métodos privados de esta clase
  - Llamar a otros ficheros/clases de semillas separados

Control desde Artisan

- Para crear un nuevo fichero semilla podemos usar el siguiente comando de Artisan:

```
$ php artisan make:seeder UsersTableSeeder
```

- Una vez definidos los ficheros de semillas, para insertar esos datos en la BD usamos el comando de Artisan:

```
$ php artisan db:seed
```

- En desarrollo es probable que queramos restaurar la base de datos completamente, incluyendo las migraciones y las semillas:

```
$ php artisan migrate:fresh --seed
```

Database Seeding, ejemplo:

```php
class DatabaseSeeder extends Seeder {
    public function run() {
        // Llamamos a otro fichero de semillas
        $this->call( UserTableSeeder::class );
        // Mostramos información por consola
        $this->command->info('User table seeded!');
    }
}
```

Desde la clase principal podemos cargar otra clase externa de semillas o llamar a un método privado.

```php
class UserTableSeeder extends Seeder {
    public function run() {
        // Borramos los datos de la tabla
        DB::table('users')->delete();
        // Añadimos una entrada a esta tabla
        DB::table('users')->insert([
            'name' => 'Username',
            'email' => 'name@domain.com',
            'password' => 'strongpassword' ]);
    } // véase también el método insertGetId que devuelve el ID asignado
}
```

Primero eliminamos los datos de la tabla y después añadimos los datos que queramos.

Laravel: Acceso a datos

QUERY BUILDER

Query Builder

- Laravel incluye una serie de clases que nos facilita la construcción de consultas y otro tipo de operaciones con la base de datos
- Al utilizar estas clases obtenemos varias ventajas:
  - Es compatible con todos los tipos de bases de datos soportados por Laravel
  - Creamos una notación mucho más legible
  - Nos previene de cometer errores o de ataques por inyección de código SQL

Query Builder

- Por ejemplo, para realizar una consulta a la tabla `users` hacemos:

```php
$users = DB::table('users')->get(); // select * from users

foreach($users as $user)
{
    var_dump($user->name);
}
```

Query Builder

- Se puede recuperar un conjunto de datos con `get()` o un único valor con `first()`

```php
// Recupera todos los objetos de la tabla
$users = DB::table('users')->get();
// $users es de tipo Illuminate\Support\Collection
foreach ($users as $user) {
    // Los objetos son de tipo StdClass
    // y se puede acceder a sus propiedades
    echo $user->name;
}

// Recupera un único objeto
$user = DB::table('users')->where('name', 'John')->first();
echo $user->name;
```

Query Builder

- También podemos utilizar los métodos `orderBy`, `groupBy` y `having` en las consultas:

```
$users = DB::table('users')
                ->orderBy('name', 'desc')
                ->groupBy('count')
                ->having('count', '>', 100)
                ->get();
```

- Para más información (join, insert, update, delete, etc.) https://laravel.com/docs/6.x/queries

Laravel: Acceso a datos

EJERCICIO

Ejercicio

1. Configura tu proyecto para trabajar con la base de datos MySQL:
   - DB_DATABASE=dss
   - DB_USERNAME=dss
   - DB_PASSWORD=dss
2. Crea una migración para crear una tabla `categories` con los siguientes campos

Nombre	**Tipo de datos**
id	Clave principal, autoincremental
name	VARCHAR

3. Crea un *seeder* para insertar las categorías “Procesadores” y “Discos duros”. Modifica `database/seeds/DatabaseSeeder.php` para que ejecute el nuevo seeder

Ejercicio

4. Ejecuta la migración y el seeder
5. Accede a la base de datos con phpMyAdmin para comprobar que los datos se han introducido
6. Crea una migración para crear una tabla `products` con los siguientes campos

Nombre	**Tipo de datos**
id	Clave principal, autoincremental
name	VARCHAR
price	Float
category_id	Entero, clave ajena a categorías

7. Ejecuta la nueva migración

Ejercicio

8. Crea un nuevo seeder que introduzca varios productos para cada categoría, necesitarás recuperar el identificador de cada categoría antes de realizar las inserciones con `insert()` para pasar el valor correcto de la clave ajena
9. Vuelve a poblar la base de datos
10. Usando `tinker`, prueba las siguientes consultas
    a. Recupera todas las categorías
    b. Recupera todos los productos de la categoría “Procesadores”
    c. Recupera el precio del producto más caro de la categoría “Procesadores”
    d. Elimina todos los productos cuyo nombre empiece por una letra a tu elección