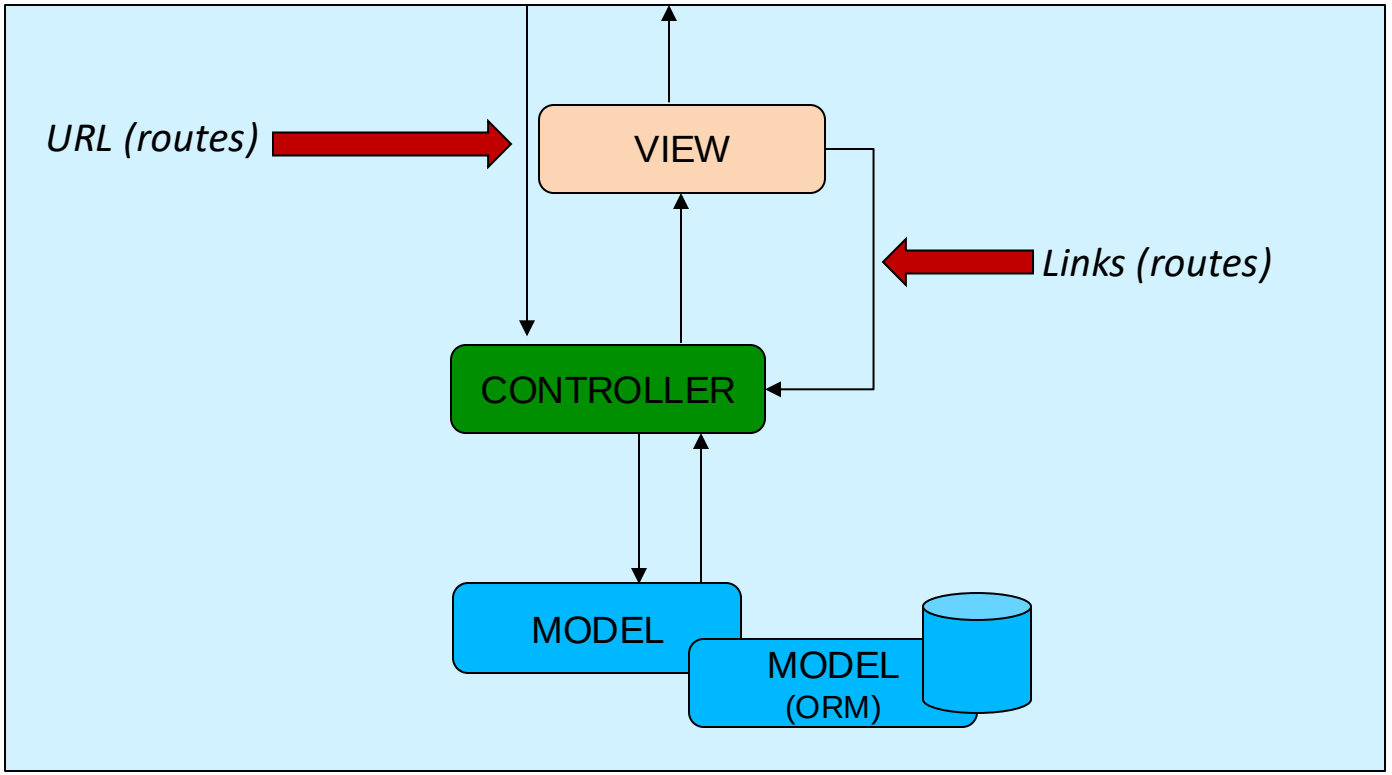
PROGRAMMAZIONE WEB

OBJECT-RELATIONAL MAPPING ELOQUENT

Prof. Ada Bagozi
ada.bagozi@unibs.it

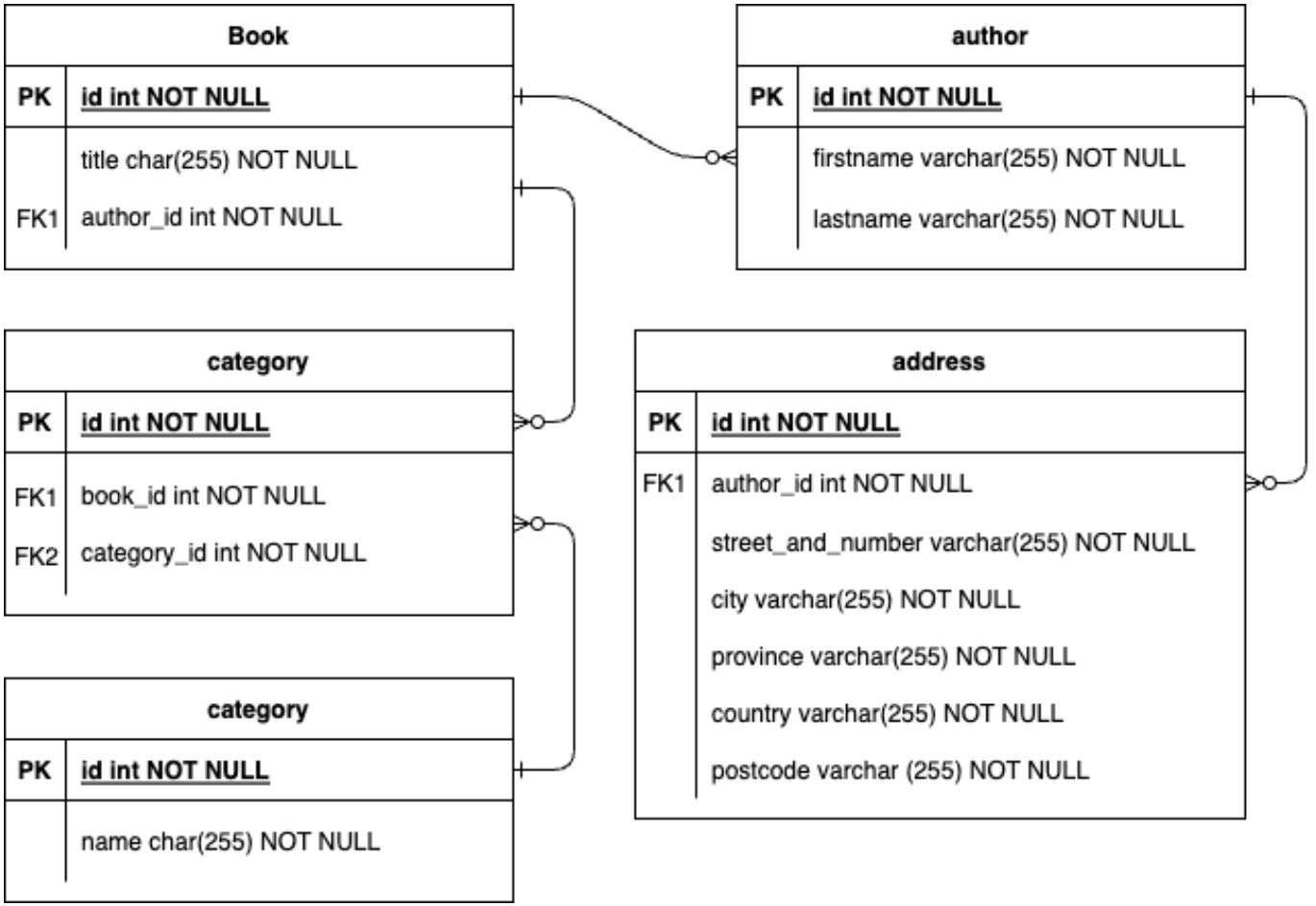
MVC design pattern



Creazione del Model (I)

- ✓ La componente **`Model`** include tutte le classi PHP in cui viene codificata la logica di business dell'applicazione, così come le classi per l'interazione con il database (ORM)
- ✓ Preparazione dei parametri di configurazione del database (nel file **`.env`**)

DB del running example



Creazione del Model (II)


```
php artisan make:model ModelName
```

Crea un file **`Model.php`** all'interno della cartella **`app/`**

Per convenzione, corrisponde ad una tabella nel database il cui nome è ottenuto da quello del modello:

- ✓ Trasformato da CamelCase a snake_case e pluralizzato (**`MyAuthor`** -> **`my_authors`**)
- ✓ A meno che non si usi la proprietà **`$table`**

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    //protected $table = 'alter_name_for_the_book_table';
}
```

Creazione del Model (III)

Per convenzione, la chiave primaria:

- ✓ È associata al campo **`id`** autoincrementale e intero
- ✓ A meno che non si usi la proprietà **`$primaryKey`**
- ✓ Ogni oggetto della classe **`.php`** generata presenta implicitamente una serie di campi corrispondenti alle colonne della tabella corrispondente nel DB
- ✓ Altre opzioni disponibili (**`id`** non incrementale, non intero, etc.)

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    //protected $table = 'alter_name_for_the_book_table';
    //protected $primaryKey = 'alter_field_as_primary_key';
}
```

Creazione del Model (IV)

- ✓ Per default, Eloquent si aspetta in ogni tabella del database due colonne aggiuntive **`created_at`** e **`updated_at`** per tenere traccia delle firme temporali di creazione e modifica di ogni istanza
- ✓ A meno che non si usi la proprietà **`public $timestamps = false`**
- ✓ Altre opzioni disponibili (come la possibilità di cambiare il nome delle colonne che dovrebbero contenere le firme temporali di creazione e di modifica)
- ✓ Esiste anche la proprietà **`deleted_at`** per implementare il cosiddetto «soft delete»
  - ✓ Per abilitare questa opzione nel modello va esplicitato **`use SoftDeletes`**

Relazioni uno-a-uno

Modellata tramite i metodi **`hasOne()`** e **`belongsTo()`**

```php
// Method of Author model
0 references | 0 overrides
public function address()
{
    // the property $author->address returns an object of type Address
    return $this->hasOne(Address::class,'author_id','id');
}

// Method of Address model
0 references | 0 overrides
public function author()
{
    // the property $address->author returns an object of type Author
    return $this->belongsTo(Author::class,'author_id','id');
}
```

La convenzione richiede una colonna **`author_id`** nella tabella **`addresses`**, in tal caso il secondo e il terzo parametro dei metodi sono evitabili

Relazioni uno-a-molti

Modellata tramite i metodi **`hasMany()`** e **`belongsTo()`**

```php
// Method of Author model
0 references | 0 overrides
public function books()
{
    // the property $author->books returns an array of Books
    return $this->hasMany(Book::class,'author_id','id');
}

// Method of Book model
0 references | 0 overrides
public function author()
{
    // the property $book->author returns an object of type Author
    return $this->belongsTo(Author::class,'author_id','id');
}
```

La convenzione richiede una colonna **`author_id`** nella tabella **`books`**, che punta alla colonna id della tabella authors, in tal caso il secondo e il terzo parametro dei metodi sono evitabili

Relazioni molti-a-molti

Modellata tramite il metodo **`belongsToMany()`**

```php
// Method of Book model
5 references | 0 overrides
public function categories()
{
    // the property $book->categories returns an array of Category
    return $this->belongsToMany(Category::class,'book_category','book_id','category_id');
}

// Method of Category model
0 references | 0 overrides
public function books()
{
    // the property $category->books returns an array of Book
    return $this->belongsToMany(Book::class,'book_category','category_id','book_id');
}
```

La convenzione richiede una tabella **`book_category`** con le due colonne **`book_id`** e **`category_id`**; in tal caso, i parametri aggiuntivi possono essere omessi

Creazione delle tabelle nel DB (I)

Laravel offre il meccanismo delle *migration* – Si tratta di script che sono posizionati nella cartella **`database/migrations/`** e si occupano del versioning del database tramite i metodi **`up()`** e **`down()`**

```
php artisan make:migration book_table
```

A questo punto, il comando `php artisan migrate` genererà le tabelle nel DB
Per annullare le migration eseguite sull'applicazione il comando da utilizzare è `php artisan migrate:rollback`

Creazione delle tabelle nel DB (II)


```php
Schema::create('author', function (Blueprint $table) {
    $table->id();
    $table->string('firstname');
    $table->string('lastname');
    $table->timestamps();
});

Schema::create('book', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->unsignedBigInteger('author_id');
    $table->timestamps();
});
```

```php
Schema::table('book', function (Blueprint $table) {
    $table->foreign('author_id')->references('id')->on('author');
});
```

Popolamento delle tabelle nel DB

Laravel offre anche gli strumenti (denominati *Factory* e *Seed*) per popolare le tabelle del DB con dati di test

- Le factory sono classi che si occupano di costruire istanze di particolari modelli inserendo dati di test nelle loro proprietà (cartella **`database/factories`**)
- I seed sono script per il popolamento del database (che possono far uso delle factory) attraverso il comando `php artisan make:seeder ${nome_seeder}`, che genererà uno script `${nome_seeder}` nella cartella **`database/seeds`**; il seeder viene invocato con il comando `php artisan db:seed --class=${nome_seeder}`

Seeder e factories possono essere create anche contemporaneamente al modello

Factories - Esempio


```
php artisan make:model Model --factory
```

```php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Book;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Book>
 */
0 references | 0 implementations
class BookFactory extends Factory
{
    0 references
    protected $model = Book::class;
    0 references | 0 overrides
    public function definition(): array
    {
        return [
            'title' => $this->faker->sentence(rand(1, 5))
        ];
    }
}
```

Seeders - Esempio (I)


```php
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    0 references | 0 overrides
    public function run(): void
    {
        $this->populateDB();
        $this->createUsers();
    }

    1 reference
    private function populateDB()
    {

        // Create 100 authors with their corresponding address
        Author::factory()->count(100)->create()->each(function ($author) {
            Address::factory()->count(1)->create(['author_id' => $author->id]);
        });

        // Randomly select a subset of 50 authors and, for each of them, create a set of books (from 1 to 5, randomly generated)
        $authors = Author::all();
        $authorsWithBooks = $authors->random(50);

        foreach($authorsWithBooks as $singleAuthor) {
            $numberOfBooks = rand(1,5);
            for($b=0; $b<$numberOfBooks; $b++) {
                Book::factory()->count(1)->create(['author_id' => $singleAuthor->id]);
            }
        }
```

Seeders - Esempio (II)


```
    // Create 10 book categories
    Category::factory()->count(1)->create(['name' => 'Romanzi classici']);
    Category::factory()->count(1)->create(['name' => 'Fantasy']);
    Category::factory()->count(1)->create(['name' => 'Gialli']);
    Category::factory()->count(1)->create(['name' => 'Thriller']);
    Category::factory()->count(1)->create(['name' => 'Saggi']);
    Category::factory()->count(1)->create(['name' => 'Poesie']);
    Category::factory()->count(1)->create(['name' => 'Psicologia']);
    Category::factory()->count(1)->create(['name' => 'Fantascienza']);
    Category::factory()->count(1)->create(['name' => 'Viaggi']);
    Category::factory()->count(1)->create(['name' => 'Arte e fotografia']);

    // Randomly associate a book to a subset of categories (from 1 to 5)
    $books = Book::all();
    $categories = Category::all();

    foreach($books as $singleBook)
    {
        $numberOfCategories = rand(1,5);
        $selectedCategories = $categories->random($numberOfCategories);
        $singleBook->categories()->attach($selectedCategories);
    }
}
```

Per l'inserimento massivo…

Proprietà modificabili in modalità *massiva* (o non modificabili):

```php
class Book extends Model
{
    protected $table = 'book';
    //protected $primaryKey = 'alter_field_as_primary_key';
    //use SoftDeletes;
    public $timestamps = false;
    use HasFactory;

    protected $fillable = ['title', 'author_id'];
```

Eloquent – Azioni CRUD (I)


```php
class DataLayer
{
    1 reference | 0 overrides
    public function listBooks() {
        $books = Book::orderBy('title','asc')->get();
        return $books;
    }

    3 references | 0 overrides
    public function findBookById($id) {
        return Book::find($id);
    }

    2 references | 0 overrides
    public function listAuthors() {
        $authors = Author::orderBy('lastname','asc')->orderBy('firstname','asc')->get();
        return $authors;
    }

    0 references | 0 overrides
    public function findAuthorById($id) {
        return Author::find($id);
    }

    2 references | 0 overrides
    public function getAllCategories() {
        return Category::orderBy('name','asc')->get();
    }
```

Eloquent – Azioni CRUD (II)


```
1 reference | 0 overrides
public function addBook($title, $author_id, $categories) {
    $book = new Book;
    $book->title = $title;
    $book->author_id = $author_id;
    $book->save();
    foreach($categories as $cat) {
        $book->categories()->attach($cat);
    }
    // massive creation (only with fillable property enabled on Book):
    // Book::create(['title' => $title, 'author_id' => $author_id, 'user_id' => $user]);
}

0 references | 0 overrides
public function addAuthor($first_name, $last_name) {
    $author = new Author;
    $author->firstname = $first_name;
    $author->lastname = $last_name;
    $author->save();

    //use the factory to randomly generate an address
    Address::factory()->count(1)->create(['author_id' => $author->id]);

    // massive creation (only with fillable property enabled on Author):
    // Author::create(['firstname' => $first_name, 'lastname' => $last_name, 'user_id' => $user]);
}
```

Eloquent – Azioni CRUD (III)


```
1 reference | 0 overrides
public function editBook($id, $title, $author_id, $categories) {
    $book = Book::find($id);
    $book->title = $title;
    $book->author_id = $author_id;
    $book->save();

    // Cancel the previous list of categories
    $prevCategories = $book->categories;
    foreach($prevCategories as $prevCat) {
        $book->categories()->detach($prevCat->id);
    }

    // Update the list of categories
    foreach($categories as $cat) {
        $book->categories()->attach($cat);
    }
    // massive update (only with fillable property enabled on Book):
    // Book::find($id)->update(['title' => $title, 'author_id' => $author_id]);
}

0 references | 0 overrides
public function editAuthor($id, $first_name, $last_name) {
    $author = Author::find($id);
    $author->firstname = $first_name;
    $author->lastname = $last_name;
    $author->save();
    // massive update (only with fillable property enabled on Author):
    // Author::find($id)->update(['firstname' => $first_name, 'lastname' => $last_name]);
}
```

Eloquent – Azioni CRUD (IV)


```php
1 reference | 0 overrides
public function deleteBook($id) {
    $book = Book::find($id);
    $categories = $book->categories;
    foreach($categories as $cat) {
        $book->categories()->detach($cat->id);
    }
    $book->delete();
}

0 references | 0 overrides
public function deleteAuthor($id) {
    $author = Author::find($id);
    $author->address->delete();
    $author->delete();
}
```

Running example v5 (I)

Home My Library

Home / Library / Books

Biblios :: Books' List

Create new book

Title	Author			
Ab itaque quod qui.	Adams	Details	Edit	Delete
Aliquid adipisci dolor.	Boyer	Details	Edit	Delete
Amet laboriosam temporibus enim.	Rutherford	Details	Edit	Delete

```
Route::get('/book', [BookController::class, 'index'])->name('book.index'); // Display the list of books
```

Running example v5 (II)

Home My Library

Home / Library / Books / Ab itaque quod qui.

Biblios :: book details

Title: Ab itaque quod qui.

Author: Adams, Fritz

Categories: Arte e fotografia
Gialli
Thriller
Viaggi

Edit

Delete

Back

```
Route::get('/book/{id}', [BookController::class, 'show'])->name('book.show'); // Display a single book
```

Running example v5 (III)

Home My Library

Home / Library / Books / Edit book

Biblios :: Edit Book

Categories

Arte e fotografia
Fantascienza
Fantasy
Gialli

Title

Ab itaque quod qui.

Author

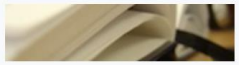
Adams

Save

```
Route::get('/book/{id}/edit', [BookController::class, 'edit'])->name('book.edit'); // Display the edit form
```

```
@if(isset($book->id))
    <form class="form-horizontal" name="book" method="post" action="{{ route('book.update', ['book' => $book->id]) }}">
    <!--<input type="hidden" name="_method" value="PUT">-->
    @method('PUT')
@else
    <form class="form-horizontal" name="book" method="post" action="{{ route('book.store') }}">
@endif
@csrf
```

Running example v5 (IV)



Home | My Library

Home / Library / Books / Delete book

Deleting book "Ab itaque quod qui." from the list

Deleting book. Confirm?

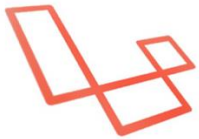
Revert	Confirm
The book **will not be removed** from the data base	The book **will be permanently removed** from the data base
Cancel	Delete

```
Route::delete('/book/{id}', [BookController::class, 'destroy'])->name('book.destroy'); // Delete the book from ID
Route::get('/book/{id}/destroy/confirm', [BookController::class, 'confirmDestroy'])->name('book.destroy.confirm');
```

```
<form name="book" method="post" action="{{ route('book.destroy', ['book' => $book->id]) }}">
    @method('DELETE')
    @csrf
    <label for="mySubmit" class="btn btn-danger"><i class="bi bi-trash"></i> Delete</label>
    <input id="mySubmit" class="d-none" type="submit" value="Delete">
</form>
```

Link utili

La documentazione ufficiale di Eloquent inclusa in quella di Laravel: https://laravel.com/docs/12.x/eloquent



laravel

PROGRAMMAZIONE WEB

OBJECT-RELATIONAL MAPPING ELOQUENT

Prof. Ada Bagozi
ada.bagozi@unibs.it