UPDATED!

Laravel

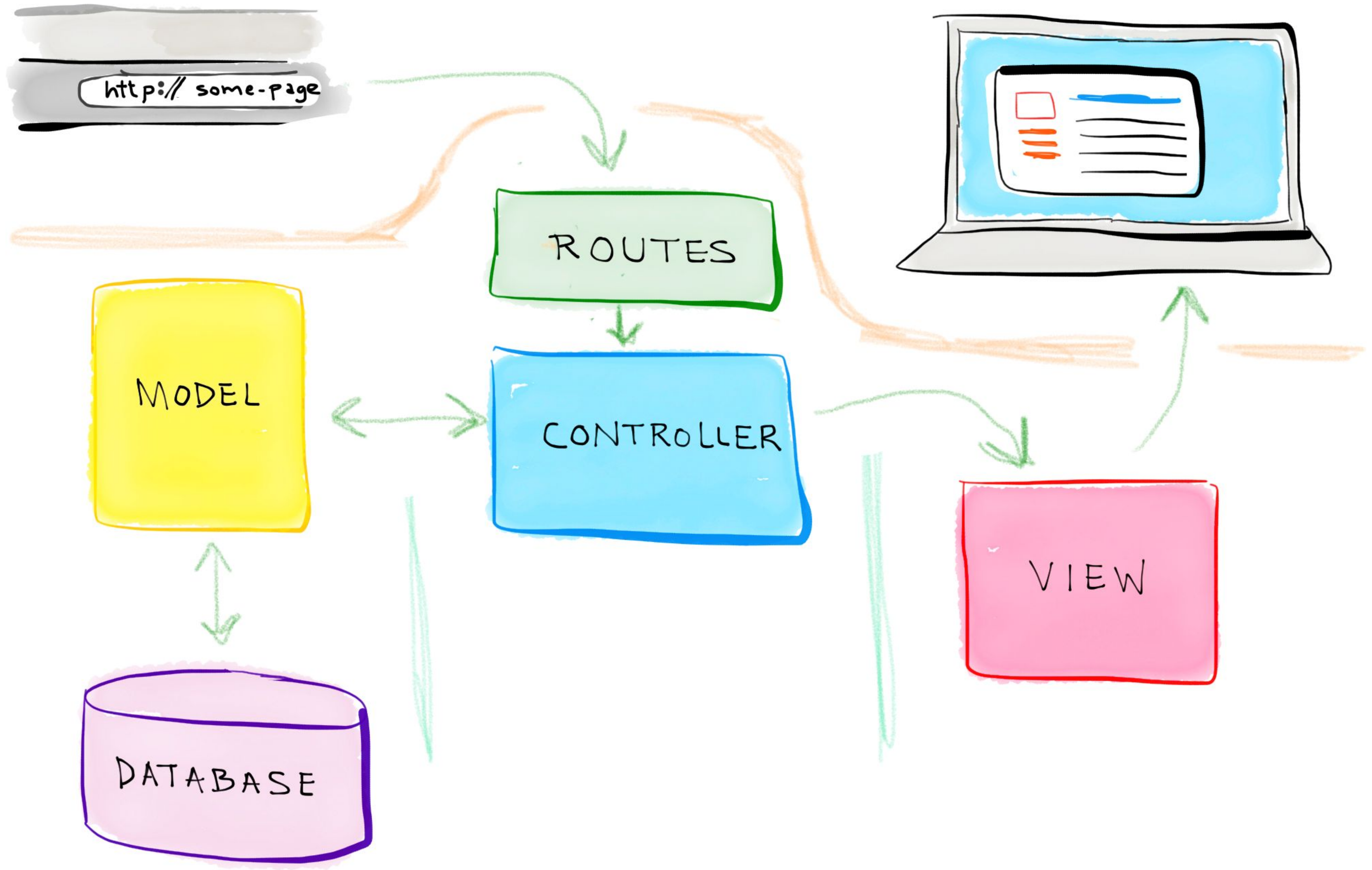
Faculty of Applied Science
Information Technology
2023-24 Fall Semester

Web Technologies
04 - Databases and Eloquent

Previous Lecture

- Collecting and Handling User Data
- Validation
- Blade Templating
- Template Inheritance

Contents


http:// some-page
ROUTES
MODEL
CONTROLLER
VIEW
DATABASE

Introduction

Laravel makes interacting with databases extremely simple across a variety of database backends using either **raw SQL**, the fluent **query builder**, and the **Eloquent ORM**. Currently, Laravel supports four databases:

- ➔ MySQL 5.6+
- ➔ PostgreSQL 9.4+
- ➔ SQLite 3.8.8+
- ➔ SQL Server 2017+

Introduction

- Raw SQL

```
DB::select('select * from users where id = :id', ['id' => 1]);
```

- Query Builder

```
DB::table('users')->where('name', 'John')->first();
```

- Eloquent ORM.

```
App\Flight::where('active', 1)
```

Raw SQL

Once you have configured your database connection, you may run queries using the DB facade. The DB facade provides methods for each type of query: **select**, **update**, **insert**, **delete**, and statement.

```
DB::select('select * from users where id = :id', ['id' => 1]);
```

```
DB::update('update users set votes = 100 where name = ?', ['John']);
```

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

```
DB::delete('delete from users');
```

Query Builder

Laravel's database query builder provides a convenient, fluent interface to creating and running database queries. It can be used to perform most database operations in your application and works on all supported database systems.

```
$users = DB::table('users')->get();
```

Query Builder

You may use the table method on the DB facade to begin a query. The table method returns a fluent query builder instance for the given table, allowing you to chain more constraints onto the query and then finally get the results using the get method:

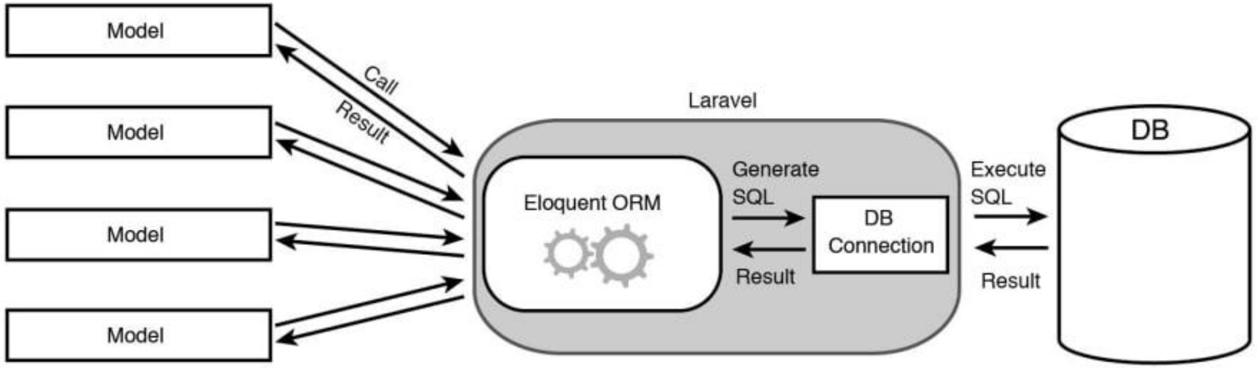
```
$users = DB::table('users')->get();
```

You may access each column's value by accessing the column as a property of the object.

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

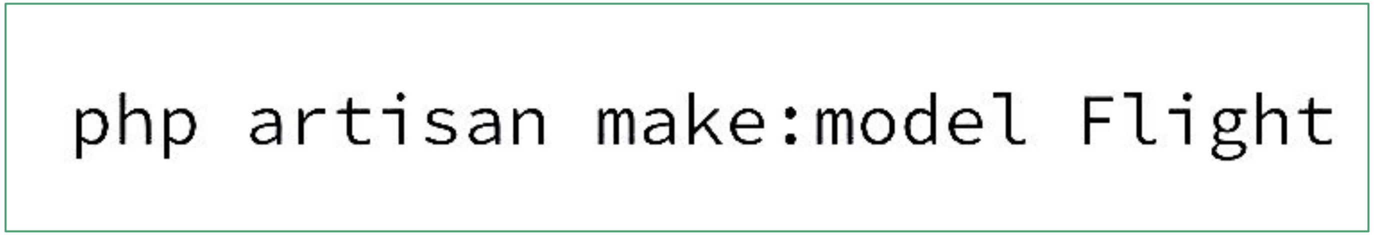
Eloquent ORM

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "**Model**" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.



Eloquent ORM

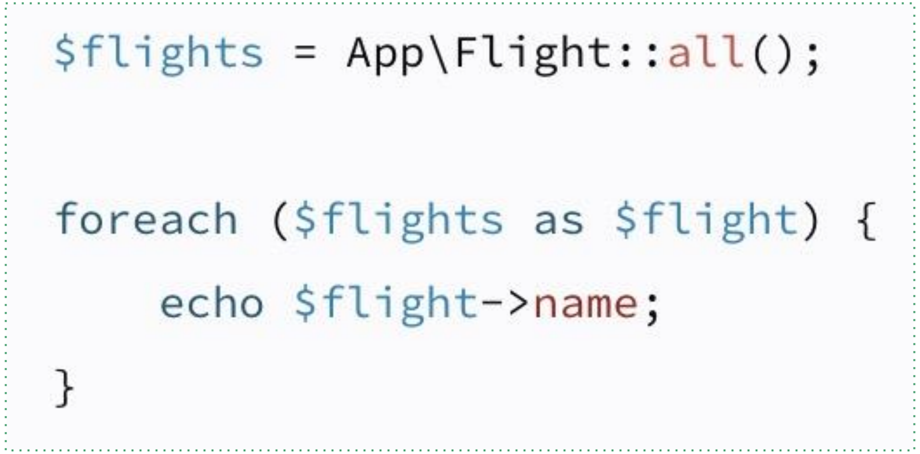
To get started, let's create an Eloquent model. Models typically live in the app directory.The easiest way to create a model instance is using the *make:model* Artisan command:



```
php artisan make:model Flight
```

Eloquent ORM

Once you have created a model and its associated database table, you are ready to start retrieving data from your database. Think of each Eloquent model as a powerful query builder allowing you to fluently query the database table associated with the model. For example:



```
$flights = App\Flight::all();


foreach ($flights as $flight) {
    echo $flight->name;
}
```

Retrieving

In addition to retrieving all of the records for a given table, you may also retrieve single records using find, first, or firstWhere. Instead of returning a collection of models, these methods return a single model instance:

```
// Retrieve a model by its primary key...
$flight = App\Flight::find(1);

// Retrieve the first model matching the query constraints...
$flight = App\Flight::where('active', 1)->first();

// Shorthand for retrieving the first model matching the query constraints...
$flight = App\Flight::firstWhere('active', 1);
```

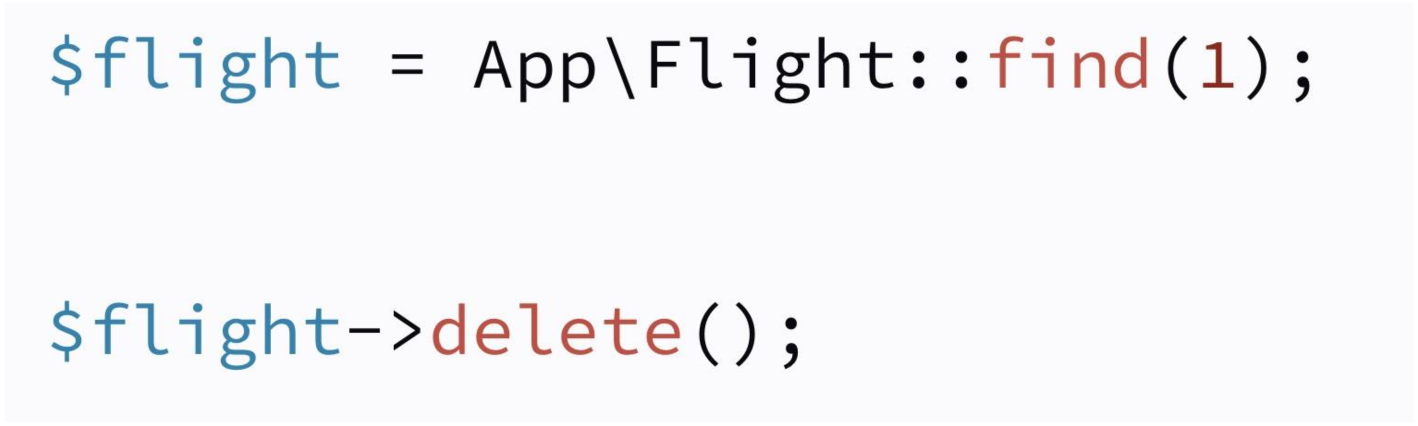
Inserts

To create a new record in the database, create a new model instance, set attributes on the model, then call the save method.

```
public function store(Request $request)
{
    // Validate the request...

    $flight = new Flight;

    $flight->name = $request->name;

    $flight->save();
}
```

Deletes

To delete a model, call the delete method on a model instance:



```
$flight = App\Flight::find(1);

$flight->delete();
```

Updates

The save method may also be used to update models that already exist in the database. To update a model, you should retrieve it, set any attributes you wish to update, and then call the save method

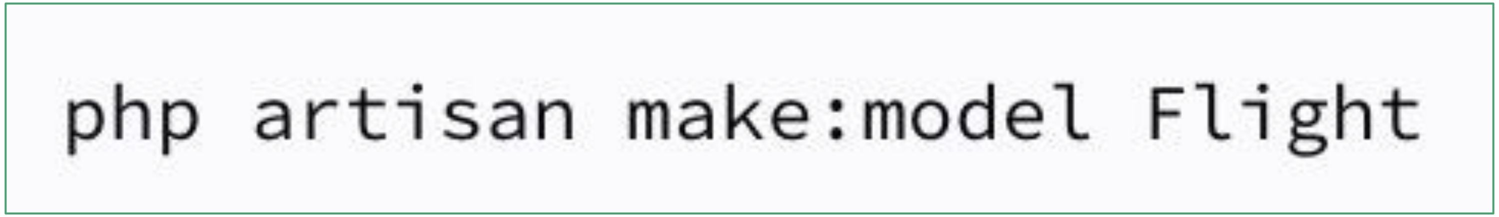
```
public function update($id){
        $flight = App\Flight::find($id);
        $flight->name = 'New Flight Name';
        $flight->save();
}
```

For more details on topics of this lecture: Read Chapter 05

Defining Models

To get started, let's create an Eloquent model. Models typically live in the app directory,

The easiest way to create a model instance is using the make:model Artisan command:



```
php artisan make:model Flight
```

Defining Migrations

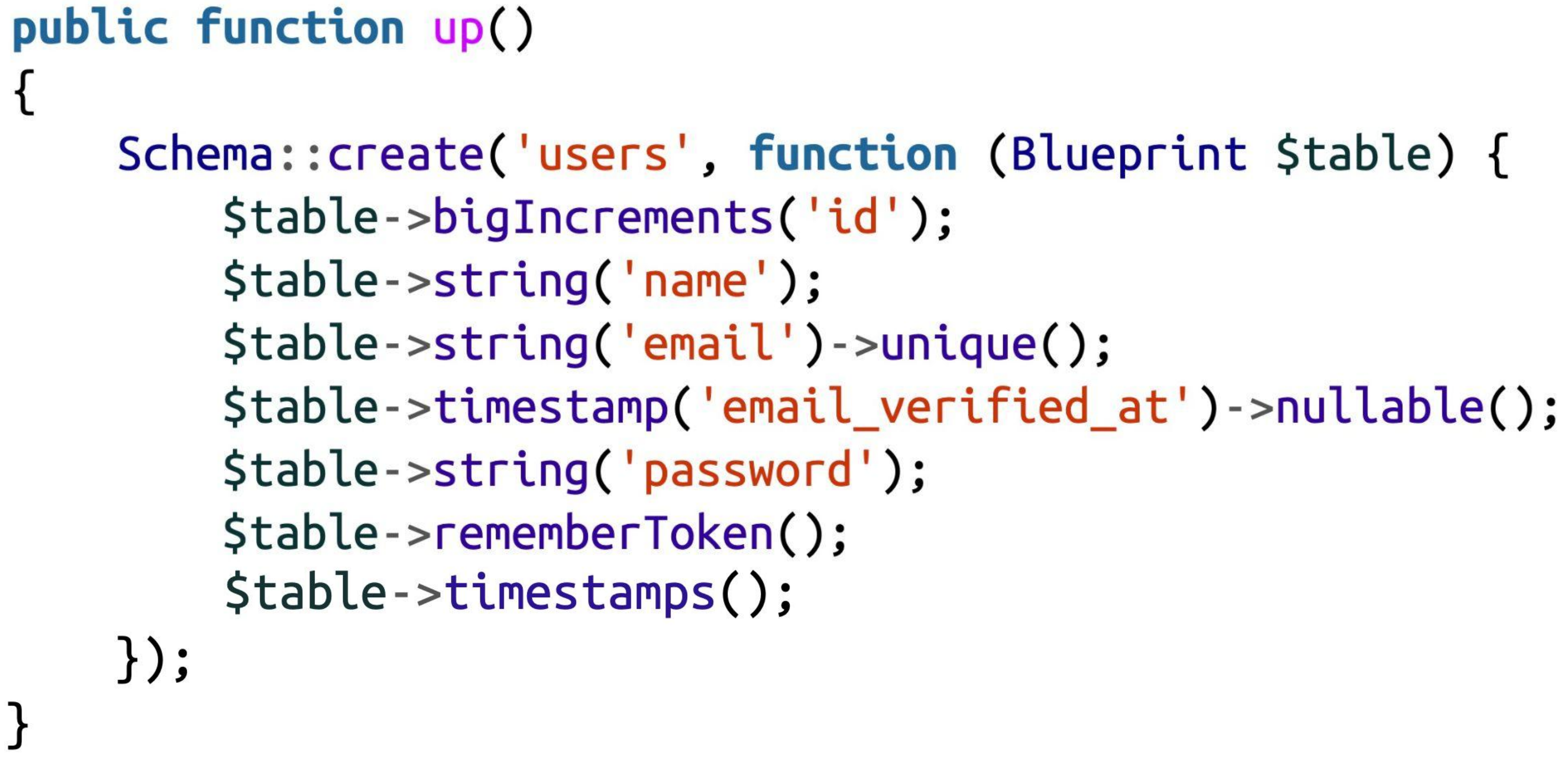
A migration is a single file that defines two things: the modifications desired when running this migration up and, optionally, the modifications desired when running this migration down

Creating a migration

Laravel provides a series of command-line tools you can use to interact with your app and generate boilerplate files. One of these commands allows you to create a migration file.

```
php artisan make:migration create_users_table
php artisan make:migration add_votes_to_users_table --table=users
php artisan make:migration create_users_table --create=users
```

Creating a migration



```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();
    });
}
```

Creating a migration

```
public function down()
{
    Schema::dropIfExists('users');
}
```

Running Migrations

Once you have your migrations defined, how do you run them? There's an Artisan command for that:

```
php artisan migrate
```

This command runs all "outstanding" migrations (by running the up() method on each). Laravel keeps track of which migrations you have run and which you haven't. Every time you run this command, it checks whether you've run all available migrations, and if you haven't, it'll run any that remain.

Running Migrations

migrate:refresh

Rolls back every database migration you've run on this instance, and then runs every migration available. It's the same as running migrate:reset and then migrate, one after the other.

migrate:fresh

Drops all of your tables and runs every migration again. It's the same as refresh but doesn't bother with the "down" migrations—it just deletes the tables and then runs the "up" migrations again.

Running Migrations

migrate:rollback

Rolls back just the migrations that ran the last time you ran migrate, or, with the added option --step=n, rolls back the number of migrations you specify.

migrate:status

Shows a table listing every migration, with a Y or N next to each showing whether or not it has run yet in this environment.

For more details on topics of this lecture:
Read Chapter 05

Activities and Next Week Topics

This Week:

- Study for the quiz.
- Read Chapter 05 of Laravel: Up & Running, for more information on Databases and Eloquent.
- Practice different types of databases and connections.
- Create simple database for your project.

Next Week:

- Database Migrations.
- Blade Templating.

References / Further Readings

- Matt Stauffer, 2019. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O’Reilly Media.
- Laravel.com : Laravel’s official Documentation.
- Dayle Rees, 2016. Laravel: Code Smart.