

# PlantUMLGen

## [Manual de Usuario]

Version:1.0 | 07/10/24 | Brian Pando

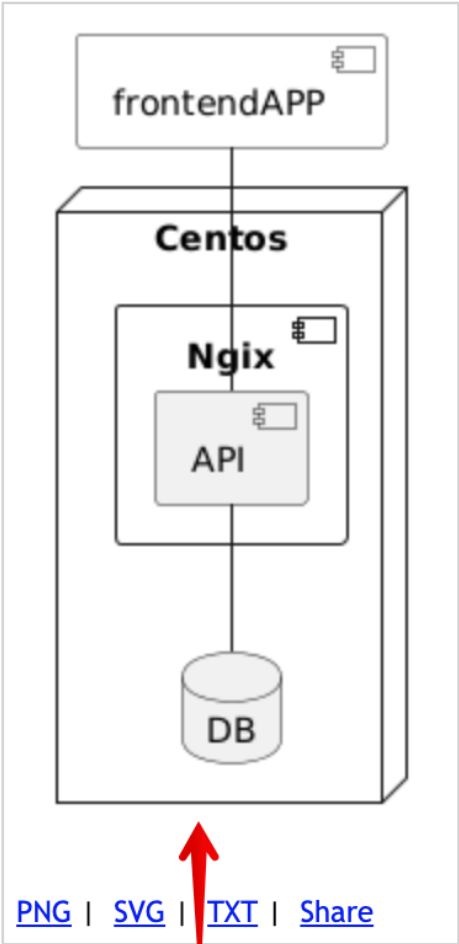
### Que es PlantUML?

PlantUML es una herramienta de diagramación que permite crear diagramas basado en UML. A diferencia de otras herramientas, PlantUML se crea escribiendo texto, por lo que se vuelve reproducible y rapidamente modificable.

 **PlantText** The expert's design tool

File ManagerDefault DiagramSamplesRefresh

```
1 @startuml
2 
3 component frontendAPP : #fff
4 
5 node Centos {
6   component Ngix {
7     [API]
8   }
9   database "DB"
10 }
11 
12 
13 ' Definimos las relaciones
14 frontendAPP -- API
15 API -- DB
16 
17 @enduml
```



[PNG](#) | [SVG](#) | [TXT](#) | [Share](#)

Sintaxis PlantUML

Visualización PlantUML

Existen varios editores para escribir la sintaxis PLantUML. En este ejemplo se usa el editor en linea llamado [PlantText](#).

### La libreria

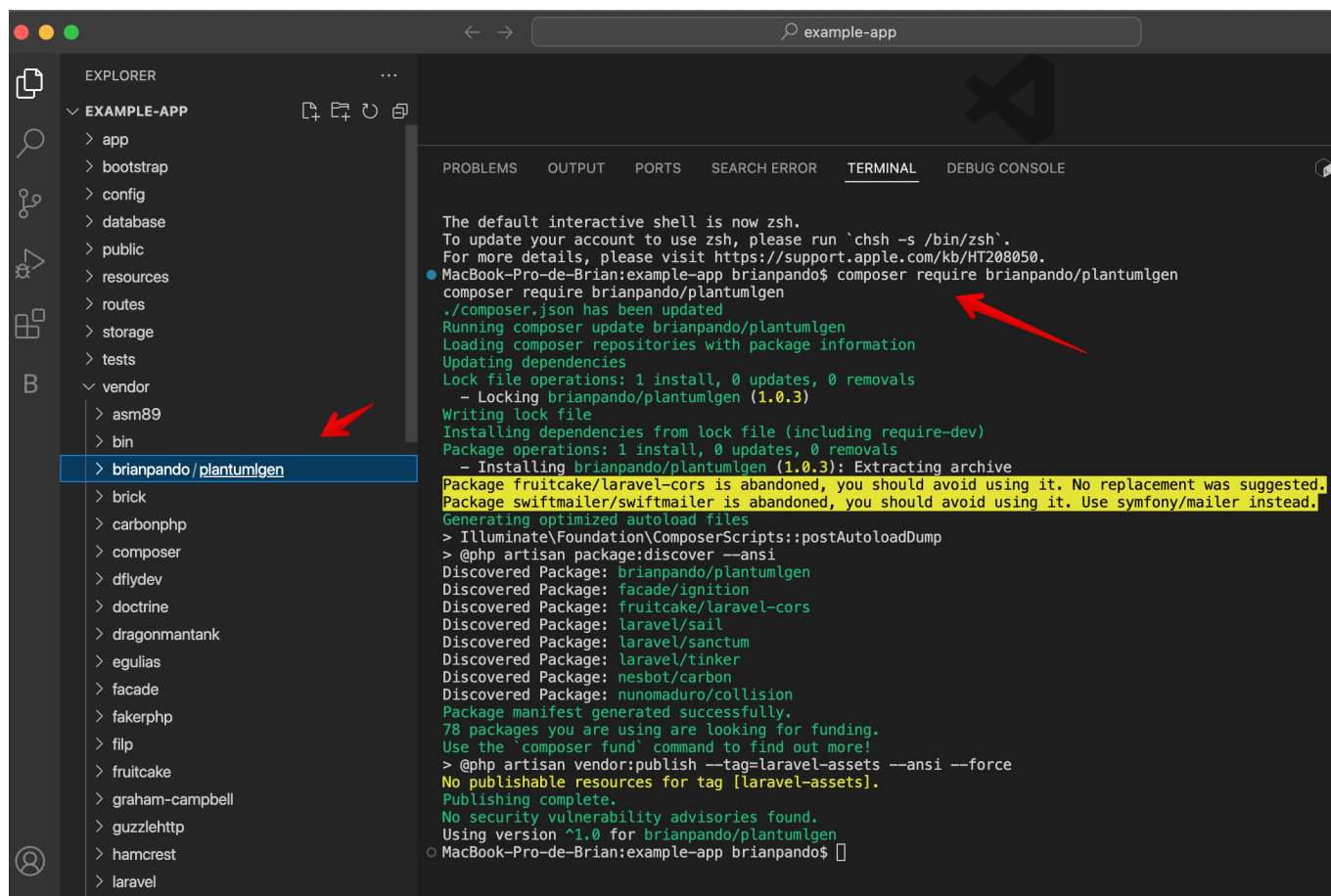
Esta librería o conocido también como paquete Laravel, se basa en el [diagrama de clases UML](#) para crear las clases de un proyecto bajo el patrón de arquitectura Modelo Vista Controlador (MVC) en Laravel. Para elaborar un diagrama de primero tener un poco de teoría sobre este tipo de diagramas.

## Instalar la librería.

La librería está disponible para ser usada dentro de un proyecto Laravel, el gestor de paquetes para un proyecto de este tipo es Composer, por tanto, para instalar esta librería debe ejecutar dentro del directorio de su proyecto:

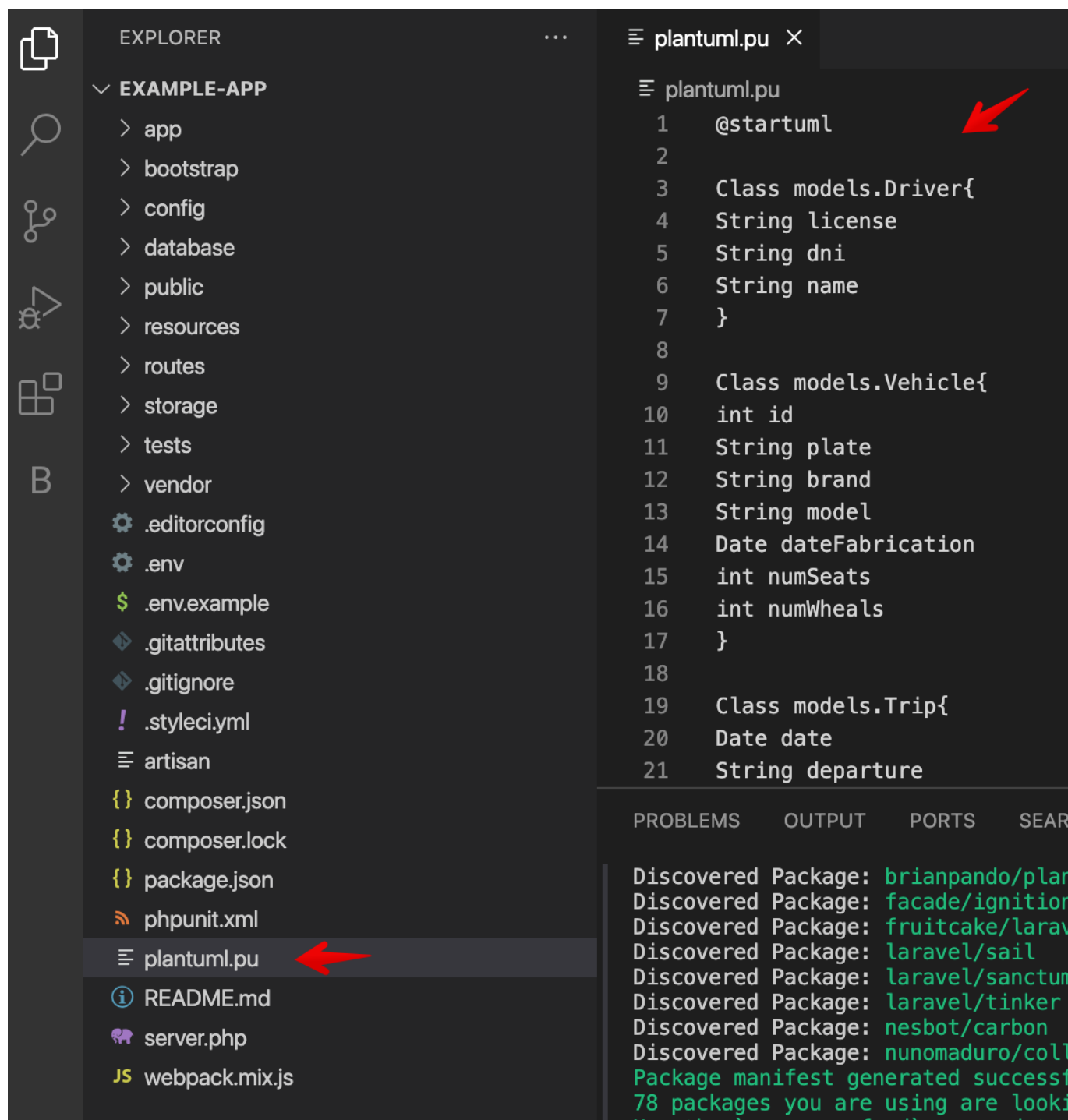
```
composer require brianpando/plantumlgen
```

Si todo salió correctamente, en el directorio `/vendor` de Laravel se habrá agregado el nuevo paquete "brianpando/plantumlgen".



## Crear el archivo PlantUML.

Para empezar a usar la librería, en la ruta raíz de su proyecto Laravel debe crear el archivo `./plantuml.pu`. En este archivo debe estar código plantUML del diagrama de clase.



Basandose en el patrón MVC, debe considerar los namespaces Models y Controllers para que la librería pueda interpretar los que intentas crear.

```
≡ plantuml.pu ×
≡ plantuml.pu
20   Date date
21   String departure
22   String arrive
23   }
24
25   Class models.Passenger{
26   String dni
27   String name
28   String lastname
29   }
30
31   models.Trip "*" o-- "*" models.Passenger
32   models.Trip "*" *-- "1" models.Driver
33   models.Trip "*" *-- "1" models.Vehicle
34
35   Class controllers.DriverController{
36   |   list()
37   |   store()
38   |   delete()
39   }
40
41   Class controllers.TripController{
42   |   list()
43   |   store()
44   |   delete()
45   }
```

Crear las clases Models.

La librería buscará en el archivo plantuml.pu, todas las clases que pertenecen a la capa Model. También, ubicará las relaciones entre estos Models.

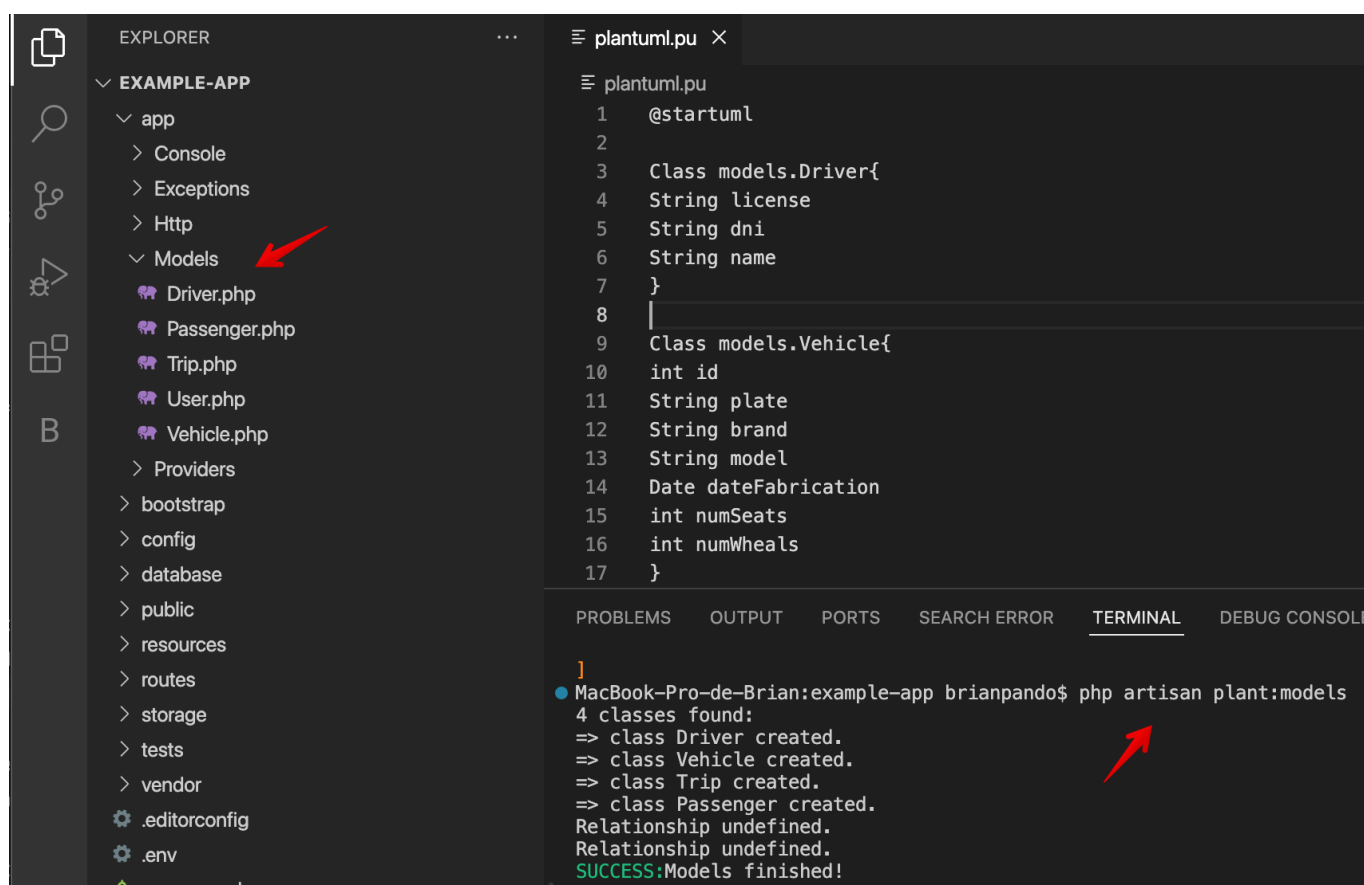
```
≡ plantuml.pu ×  
≡ plantuml.pu  
1  @startuml  
2  
3  Class models.Driver{ ←  
4  String license  
5  String dni  
6  String name  
7  }  
8  
9  Class models.Vehicle{ ←  
10 int id  
11 String plate  
12 String brand  
13 String model  
14 Date dateFabrication  
15 int numSeats  
16 int numWheels  
17 }  
18  
19 Class models.Trip{ ←  
20 Date date  
21 String departure  
22 String arrive  
23 }  
24  
25 Class models.Passenger{ ←  
26 String dni  
27 String name  
28 String lastname  
29 }  
30
```

```
30
31 models.Trip "*" o-- "*" models.Passenger
32 models.Trip "*" *-- "1" models.Driver
33 models.Trip "*" *-- "1" models.Vehicle
34
```

Para poder crear estos Models en el proyecto, debe ejecutar el siguiente comando:

```
php artisan plant:models
```

Resultado de ese comando, se muestra un mensaje de modelos creados. Puede verificar que se han creado archivos en el directorio Models/

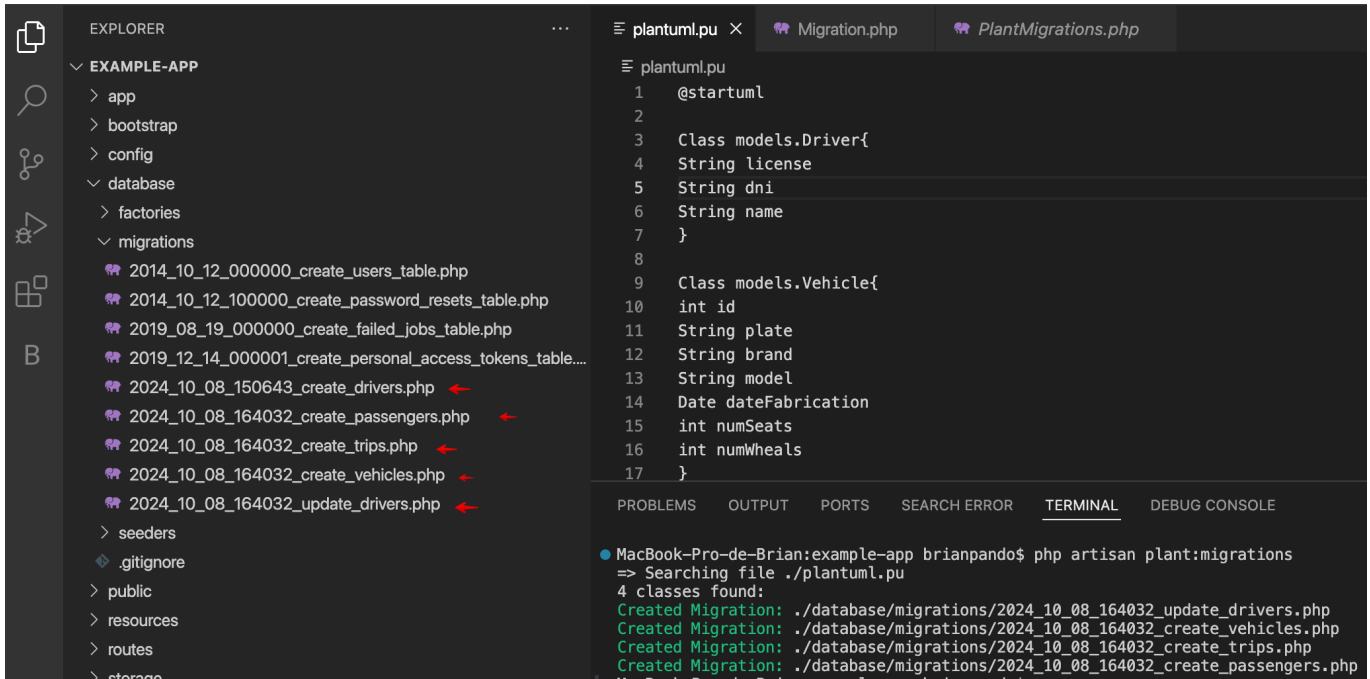


## Crear los archivos Migrations.

Laravel conectado a una base de datos relacional como Mysql, PostgreSQL y otros, se basa en archivos de migraciones, estos archivos permiten tener organizada la estructura de la base de datos desde el código laravel. Por tanto, esta librería permite crear los archivos Migrations a partir del diagrama de clase. Para crear estos archivos debe ejecutar el comando:

```
php artisan plant:migrations
```

Resultado de la ejecución se crean los archivos de migraciones con los campos y relaciones tomadas del diagrama de clase.



The screenshot shows the VS Code interface with the Explorer panel on the left, the Migration.php file in the center, and the PlantMigrations.php file on the right. The terminal at the bottom shows the output of the command `php artisan plant:migrations`, which successfully created 4 migrations.

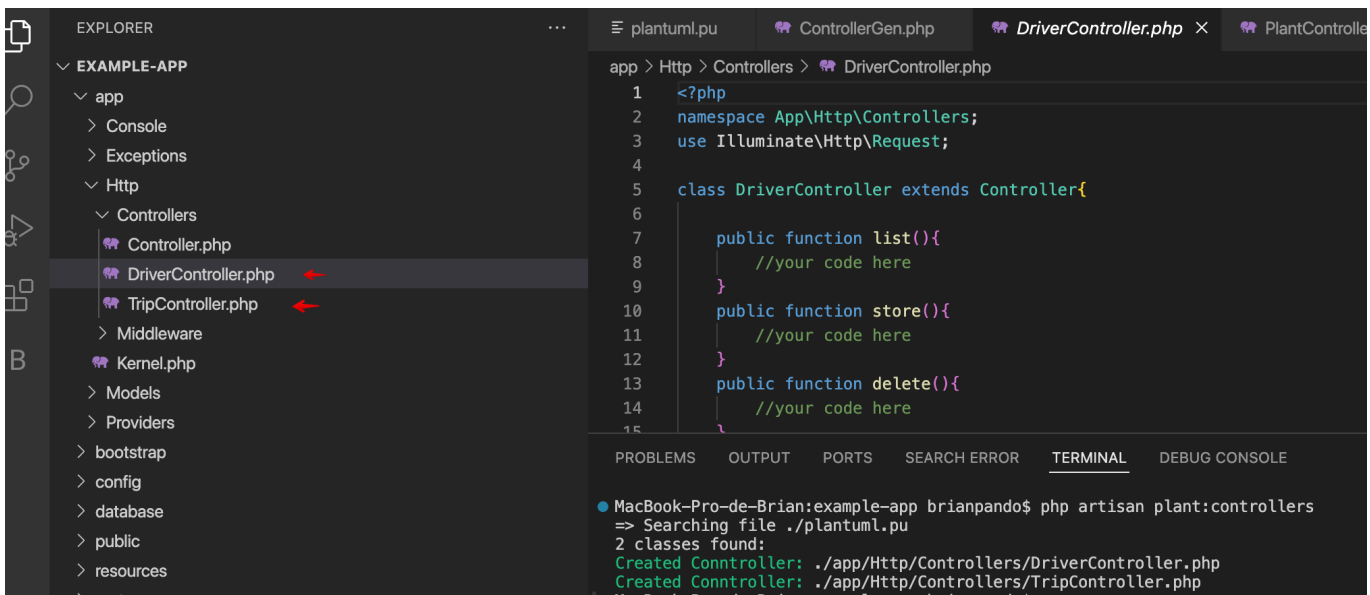
```
MacBook-Pro-de-Brian:example-app brianpando$ php artisan plant:migrations
=> Searching file ./plantuml pu
4 classes found:
Created Migration: ./database/migrations/2024_10_08_164032_update_drivers.php
Created Migration: ./database/migrations/2024_10_08_164032_create_vehicles.php
Created Migration: ./database/migrations/2024_10_08_164032_create_trips.php
Created Migration: ./database/migrations/2024_10_08_164032_create_passengers.php
```

## Crear las clases Controllers

Finalmente, hemos llegado a la parte superior de la arquitectura. Para crear los controladores debe ejecutar el siguiente comando:

```
php artisan plant:controllers
```

Al ejecutar este comando, se habran creados las clases controladores dentro del directorio `./app/http/controllers/` siguiendo el patron de Laravel.



The screenshot shows the VS Code interface with the Explorer panel on the left, the ControllerGen.php file in the center, and the DriverController.php and PlantController.php files on the right. The terminal at the bottom shows the output of the command `php artisan plant:controllers`, which successfully created 2 controllers.

```
MacBook-Pro-de-Brian:example-app brianpando$ php artisan plant:controllers
=> Searching file ./plantuml pu
2 classes found:
Created Controller: ./app/Http/Controllers/DriverController.php
Created Controller: ./app/Http/Controllers/TripController.php
```

Ahora podria codificar dentro de las clases controladores y models sin problemas, si luego debe agregar alguna propiedad a las clases modelos o quiere crear mas modelos, debe hacerlo en el diagrama y luego

volver a correr el comando para Models.

Si quiere crear mas controladores, hagalo en el diagrama y vuelva a correr los comandos del controlador. En ambas capas, puede escribir codigo y no se perderan cuando vuelva a correr estos comandos.

FIN.-