

Εισαγωγή στην Laravel

Πτυχιακή εργασία του Καρπούζη Κυριάκου

Αλεξάνδρειο Τ.Ε.Ι. Θεσσαλονίκης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Θεσσαλονίκη 2018

Μέρος Α'





Էլօգայոն

01



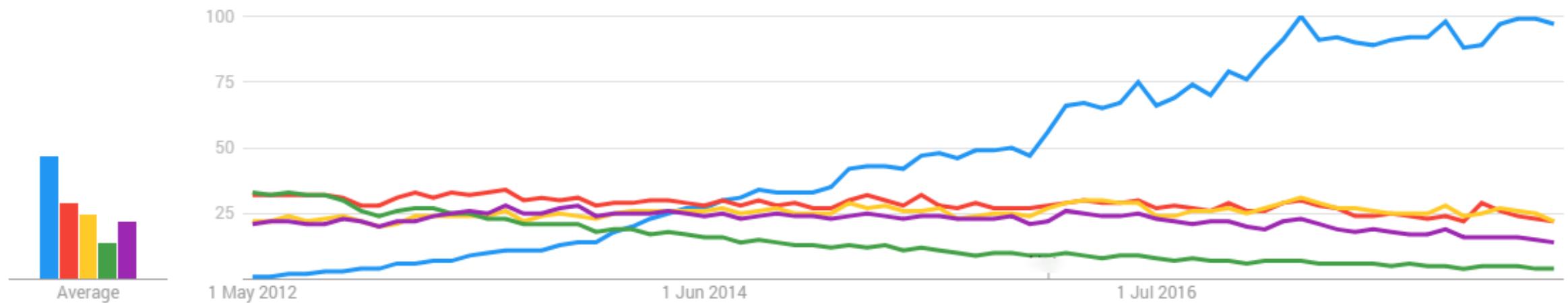
- PHP Framework ανοιχτού κώδικα, με εκφραστική και κομψή σύνταξη
- Δημιουργήθηκε από τον Taylor Otwell σαν μια εναλλακτική του CodeIgniter
- Προσφέρει ευκολία ανάπτυξης κοινών χαρακτηριστικών των μοντέρνων εφαρμογών διαδικτύου
- Παρέχει ισχυρά εργαλεία για την δημιουργία εφαρμογών μεγάλης κλίμακας, διατηρώντας ταυτόχρονα την απλότητα και την κομψότητα της σύνταξης

Ενδεικτικά...

- Εξαιρετικό Official Documentation και υποστήριξη (τόσο επίσημη όσο και Third Party)
- Σχεδιασμένη για Rapid Development
- Απλή και γρήγορη μηχανή δρομολόγησης
- Χρησιμοποιεί την αρχιτεκτονική MVC
- Κομψή δημιουργία και επικοινωνία με την Βάση Δεδομένων
- Εξαιρετικά δυνατό Dependency Injection
- Παροχή έτοιμων εργαλείων (π.χ. Αυθεντικοποίηση)
- Νούμερο 1 στα PHP Frameworks (άρα και μεγαλύτερη αγορά εργασίας)

Interest over time

Google Trends



Laravel

CodeIgniter

Symfony

Zend

Yii

- Δρομολόγηση (Routing)
- Διαχείριση Ρυθμίσεων
- Query Builder και ORM (**Object Relational Mapper**)
- Schema Builder, Migrations, Seeders
- Template Engine (Laravel Blade)
- Mails
- Authentication
- Events, Notifications, Queues
- Artisan
- Unit Testing
- Middleware

05

Ιστορικό Εκδόσεων

Έκδοση	Ημερομηνία
1.0	Ιούνιος 2011
2.0	Σεπτέμβριος 2011
3.0	22 Φεβρουαρίου 2012
3.1	27 Μαρτίου 2012
3.2	22 Μαΐου 2012
4.0	28 Μαΐου 2013
4.1	12 Δεκεμβρίου 2013
4.2	1 Ιουνίου 2014

Έκδοση	Ημερομηνία
5.0	4 Φεβρουαρίου 2015
5.1 LTS	9 Ιουνίου 2015
5.2	21 Δεκεμβρίου 2015
5.3	23 Αυγούστου 2016
5.4	24 Ιανουαρίου 2017
5.5 LTS	30 Αυγούστου 2017
5.6	7 Φεβρουαρίου 2018

Εγκατάσταση

02



- **Homestead** - To Laravel Homestead είναι ένα πακέτο Vagrant, στο οπόιο υπάρχουν όλες οι απαραίτητες βιβλιοθήκες και ρυθμίσεις που χρειάζονται από την Laravel.
- **Valet** – Το Valet είναι ένα περιβάλλον ανάπτυξης το οπόιο είναι συμβατό μόνο με Mac. Λειτουργεί μέσω του Homebrew.
- **Docker** – Πλατφόρμα ανοιχτού λογισμικού που υλοποιεί εικονοποίηση και χρησιμοποιεί Containers, με τα οποία αποφεύγεται η χρήση πόρων που θα απαιτούσε μια εικονική μηχανή.
- **Άλλες λύσεις** – Πρακτικά οποιοσδήποτε server ικανοποιεί τις απαιτήσεις της Laravel (PHP >= 7.1.3, OpenSSL PHP Extension, PDO PHP Extension, Mbstring PHP Extension, Tokenizer PHP Extension, XML PHP Extension, Ctype PHP Extension, JSON PHP Extension) π.χ XAMPP

- Το Composer είναι ένα Package Manager επιπέδου εφαρμογής για την PHP, το οποίο διαχειρίζεται τις εξαρτήσεις (dependencies) και τις βιβλιοθήκες των εφαρμογών.
- Λειτουργεί μέσω Command Line και αντλεί τα διαθέσιμα πακέτα από το Packagist (<https://packagist.org/>)
- Προσφέρει δυνατότητα autoloading για τις εφαρμογές οι οποίες παρέχουν την ανάλογη τεκμηρίωση, κάνοντας έτσι ευκολότερη την χρήση τους
- Είναι βασισμένο στην ιδέα και την λειτουργία του npm της Node.js



■ Μέσω του **Laravel Installer**

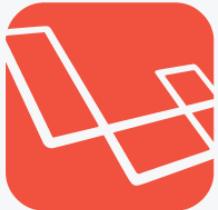
- composer global require "laravel/installer"
- Τοποθέτηση του composer στην μεταβλητή \$PATH
- laravel new blog

■ Μέσω του **Composer Create Project**

- composer create-project --prefer-dist laravel/laravel blog

Δομή Καταλόγων

03

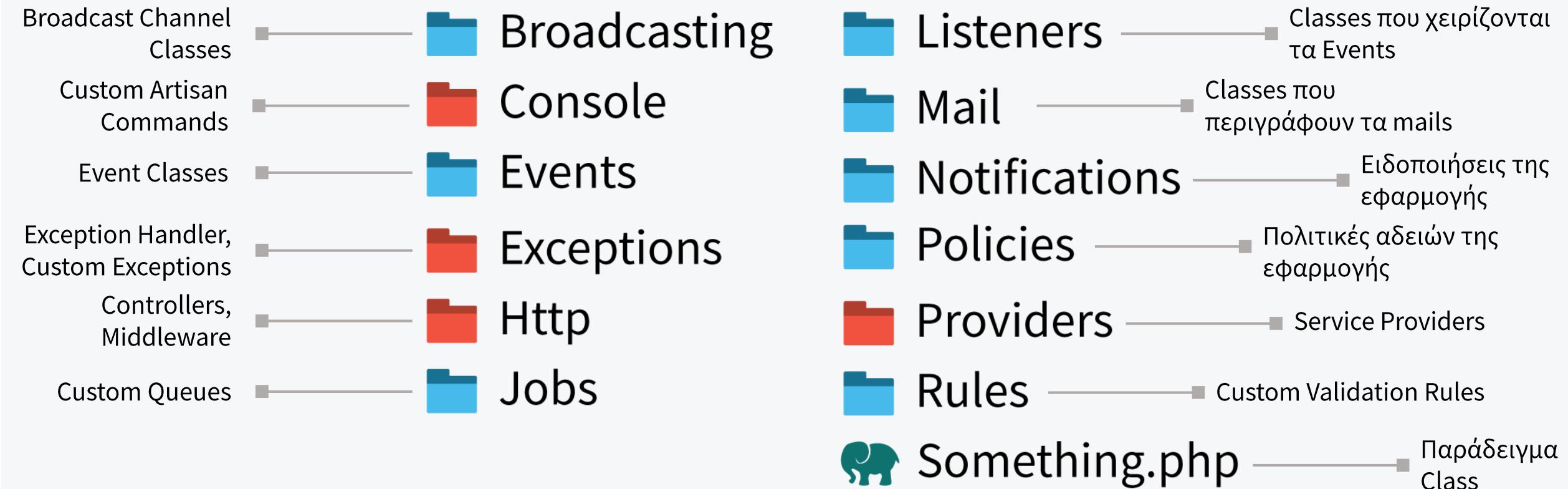


01

Root Directory

Ο πυρήνας της εφαρμογής		app
Bootstrap & Cache		bootstrap
Αρχεία Ρυθμίσεων		config
Αρχεία Βάσεων Δεδομένων		database
Το αρχικό σημείο κάθε Request στον server		public
Views, Uncompiled Assets, Languages		resources
Ορισμοί Δρομολόγησης		routes
Αποθηκευτικός Χώρος		storage
Αυτοματοποιημένα Tests		tests
Composer Dependencies		vendor

	.env	Μεταβλητές Περιβάλλοντος
	.gitignore	Αρχεία χωρίς ιχνηλασιμότητα στο GitHub
	artisan	Αρχείο για το artisan
	composer.json	Περιγραφή Composer Dependencies
	composer.lock	Τρέχοντα Composer Dependencies
	package.json	NPM Dependencies
	webpack.mix.js	Περιγραφή κανόνων για asset compiling



03

Αρχείο .env

- Αρχείο μεταβλητών περιβάλλοντος
- Μέρος του DotEnv PHP Library από τον Vance Lucas
- Δεν υποβάλλεται στο Source Control καθώς κάθε περιβάλλον (Local ή Production) περιέχει διαφορετικές μεταβλητές
- Για αυτό τον λόγο περιέχει και ευαίσθητες πληροφορίες της εφαρμογής (π.χ. Access Tokens)
- Στο Source Control υποβάλλεται μόνο το Boilerplate του αρχείου

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:5rpGD2snVRBnFdDfD2N9JXCoUsg60Hf7+GTci9+4E8M=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=testdatabase
DB_USERNAME=root
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
QUEUE_DRIVER=sync

MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

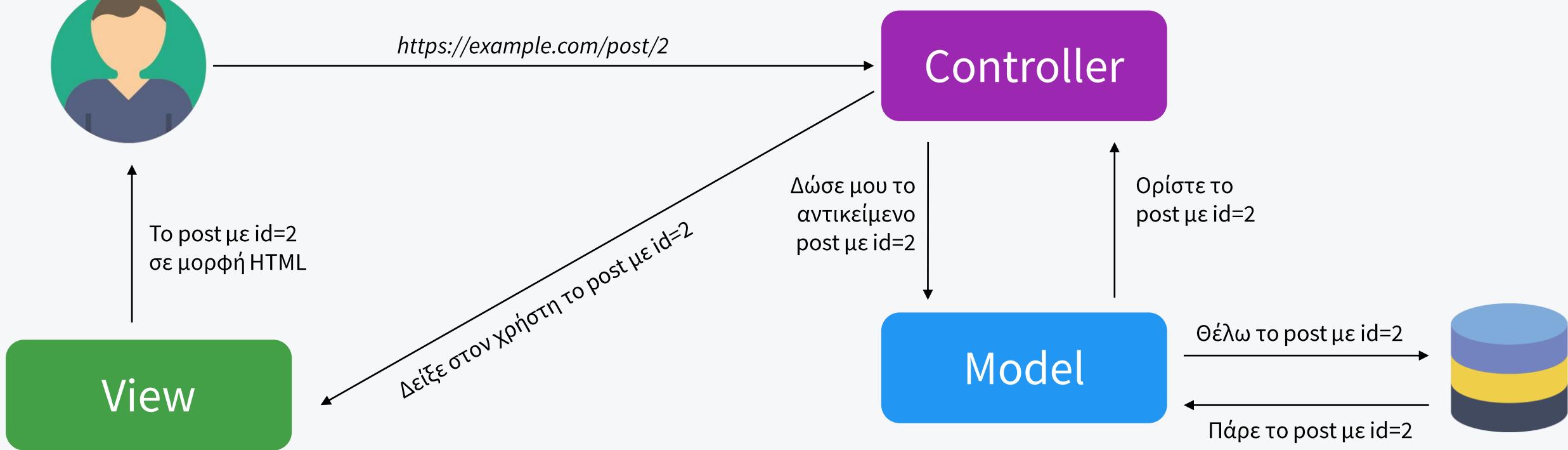
MY_CUSTOM_VARIABLE=abc
```

Αρχιτεκτονική

04

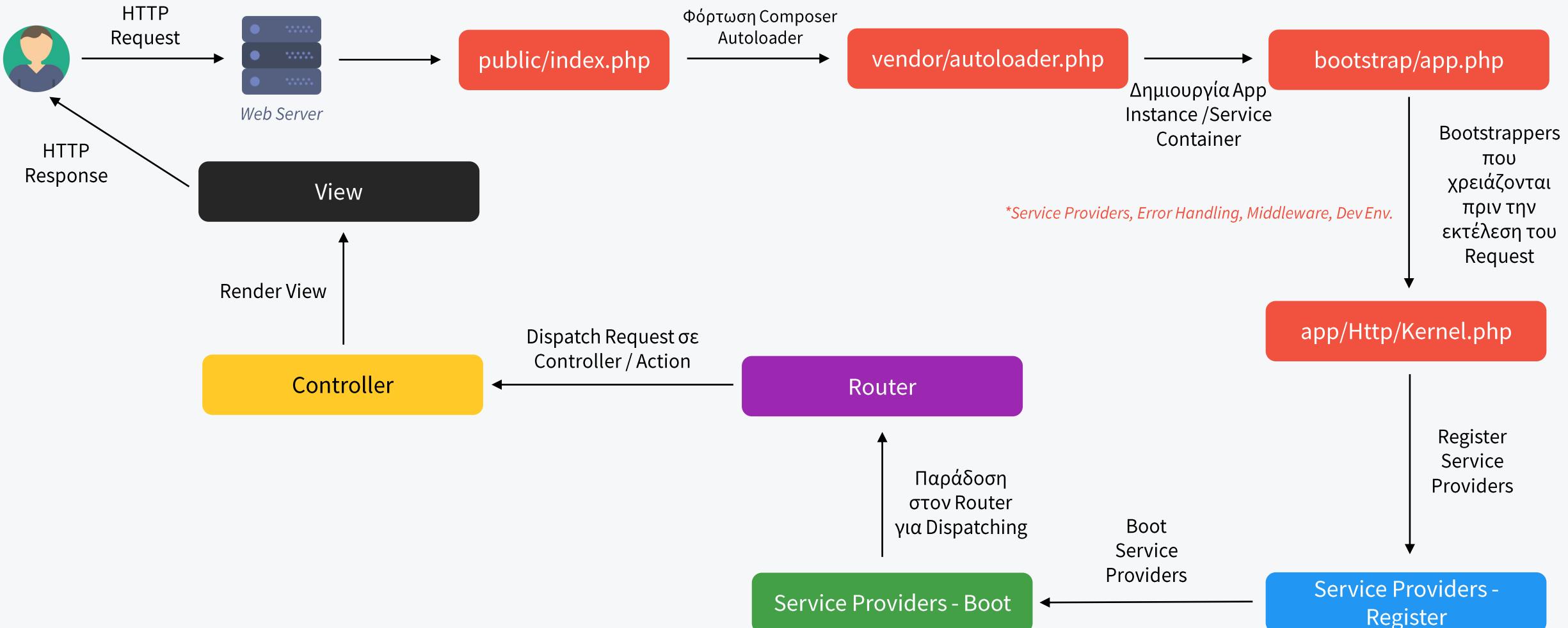


- **Model – View – Controller**
- Επιτρέπει επαναχρησιμοποίηση του κώδικα, παράλληλη ανάπτυξη και διατηρεί την οργάνωση της εφαρμογής.
- **Model** – Η περιγραφή μιας οντότητας (κλάση). Περιγράφει συνήθως την Βάση Δεδομένων
- **View** – Το αποτέλεσμα που αποτυπώνεται στην οθόνη του χρήστη και μπορεί να αλληλεπιδράσει μαζί του
- **Controller** - Η λογική της εφαρμογής. Συνδέει το Model με το View και κάνει τις καθορισμένες ενέργειες (π.χ. Διαχείριση Request, Εντολή για query στην Βάση Δεδομένων κ.τ.λ.)



02

Request Lifecycle



- To Service Container είναι ο τρόπος με τον οποίο η Laravel διαχειρίζεται τα Class Dependencies και εκτελεί το **Dependency Injection**^[1]
- Πρακτικά αποτελεί το instance της εφαρμογής
- Στο Service Container, γίνονται στην εφαρμογή μας injections τα οποία χρειάζονται για την λειτουργία της.

```
App::bind('App\\Billing\\Stripe', function() {  
    return new \App\\Billing\\Stripe('MySecretKey');  
});  
  
$stripe = resolve('App\\Billing\\Stripe');
```

[1]: (Οι εξαρτήσεις μιας κλάσης “εισέρχονται” στην κλάση μέσω ενός δομητή ή σπανιότερα μέσω μια setter μεθόδου)

- Οι Service Providers είναι το κεντρικό μέρος στο οποίο συμβάινει εξ' ολοκλήρου το απαραίτητο **Bootstraping^[1]** της εφαρμογής.
- Ένας Service Provider αποτελείται από τις μεθόδους **register()** και **boot()**
- **register()** – Το πέρασμα των dependencies στο Service Container
- **boot()** – Το έναυσμα για την εκκίνηση των υπηρεσιών. **Καλείται αφού έχουν γίνει register όλοι οι Service Providers,** δηλαδή η εφαρμογή έχει πρόσβαση σε όλα τα dependencies

[1]: To Register όλων των απαραίτητων πραγμάτων π.χ. Service Container Bindings, Event Listeners, Middleware κ.τ.λ.

```
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Post;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        view()->composer('view', function() {
            Post::all();
        })
    }

    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        App::bind('App\Billing\Stripe', function() {
            return new \App\Billing\Stripe('MySecretKey');
        });
    }
}
```

- Τα Facades προσφέρουν μια «στατική» διασύνδεση με κλάσεις που είναι διαθέσιμες στο Service Container της εφαρμογής.
- Διευκολύνουν το συντακτικό τόσο στην γραφή όσο και στην ανάγνωση

```
use vendor\Author\Package\App\Model\Post;  
[...]  
Post::show(1);
```



```
use Post;  
[...]  
Post::show(1);
```

Βασικά Στοιχεία

05



- Το artisan είναι ενα Command Line Interface που περιέχεται στην Laravel
- Απαρτίζεται από διάφορες εντολές που βοηθούν τον προγραμματιστή κατά την ανάπτυξη της εφαρμογής
- Υπάρχει η δυνατότητα δημιουργίας custom εντολών από τον προγραμματιστή

```
$- php artisan make:model Post
```

```
$- php artisan migrate
```

```
$- php artisan custom:command
```

- Στην Laravel υπάρχει εξ' ορισμού ένας δρομολογητής, δηλαδή ένα εργαλείο που δρομολογεί όλα τα Requests ενός χρήστη/εφαρμογής
- Οι διαδρομές καθορίζονται συνήθως στο αρχείο routes/web.php και στο routes/api.php
- Διαθέσιμες μέθοδοι: GET, POST, PUT, PATCH, DELETE, OPTIONS

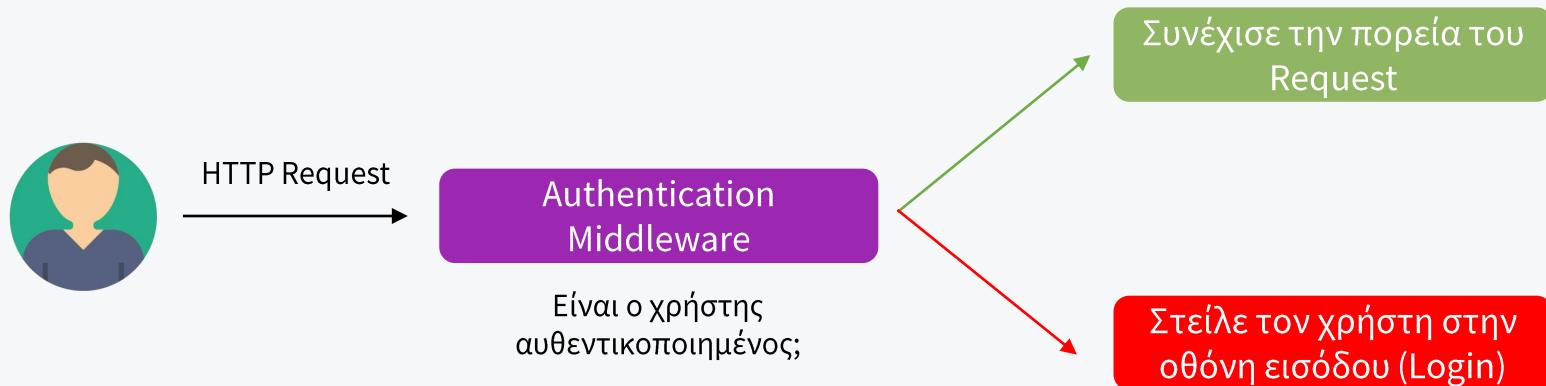
```
Route::get('foo', function () {  
    return 'Hello World';  
});
```

```
Route::get('/user', 'UserController@index');
```

```
Route::post('/user/{id}', 'UserController@edit');
```

```
Route::post('/user/{id}', 'UserController@edit')->name('edit.user');
```

- Το Middleware είναι ένας πρακτικός μηχανισμός της Laravel, ο οποίος «φιλτράρει» όλα τα HTTP Requests που γίνονται στην εφαρμογή
- Μπορεί να ενεργήσει τόσο πριν όσο και μετά το Request
- Η Laravel περιέχει ορισμένα Middleware (π.χ Authentication Middleware), αλλά είναι στην ευχαίρια του προγραμματιστή η συγγραφή Middleware που εξυπηρετούν τους σκοπούς του



- Η Laravel παρέχει ενσωματωμένο μηχανισμό προστασίας από επιθέσεις CSRF^[1]
- Παράγει αυτόματα ένα CSRF Token για κάθε ενεργό session χρήστη. Με αυτό το token πιστοποιεί ότι όντως ο χρήστης είναι αυτός που κάνει τα requests στην εφαρμογή

[1]: Η CSRF είναι επίθεση που αναγκάζει τον τελικό χρήστη να εκτελέσει ανεπιθύμητες ενέργειες σε μια εφαρμογή δικτύου στην οποία είναι πιστοποιημένος.

- Τα Models είναι οι οντότητες (κλάσεις - αντικείμενα) της εφαρμογής
- Είναι ουσιαστικά η προγραμματιστική αναπαράσταση ενός πίνακα της Βάσης Δεδομένων
- Επιτρέπουν την αλληλεπίδραση του χρήστη με την Βάση Δεδομένων (**CRUD Operations**)^[1]
- Εξ' ορισμού τοποθετούνται στο /app αλλα ο προγραμματιστής μπορεί να τοποθετήσει τα Models όπου επιθυμεί
- Ένας «έξυπνος» μηχανισμός αντιστοιχίζει τον όνομα του Model στον ενικό αριθμό με το αντίστοιχο όνομα του πίνακα στον πληθυντικό (Post - posts)

[1]: Create, Read, Update, Delete

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    public function user()
    {
        return $this->belongsTo('App\User');
    }
}
```

- Τα Views είναι τα αρχεία που αναλαμβάνουν την τελική παρουσίαση στον χρήστη.
- Είναι γραμμένα σε HTML και τις περισσότερες φορές χρησιμοποιούν το Blade Syntax
- Βρίσκονται στον κατάλογο resources/views
- Περιέχουν ελάχιστη εώς καθόλου προγραμματιστική λογική, λειτουργούν περισσότερο ώς προς το σχεδιαστικό κομμάτι της εφαρμογής

```
@extends('layouts.master')

@section('content')

    @foreach ($posts as $post)
        <h1>{{ $post->title }}</h1>
        <p>{{ $post->body }}</p>
    @endforeach

@endsection
```

- Οι Controllers περιέχουν όλη την λογική χειρισμού των requests της εφαρμογής
- Βρίσκονται στον κατάλογο app/Http/Controllers
- Αποτελούν τον συνδετικό κρίκο ανάμεσα στο Model και το View
- Όλοι οι Controllers είναι επέκταση του βασικού Controller που περιέχει το Framework (app/Http/Controller.php)

```
namespace App\Http;

use App\Post;

class PostController extends Controller
{

    public function show($id)
    {
        $post = Post::where('id', '=', $id)->first();

        return view('show')->with('post', $post);
    }
}
```

- Οι Resource Controllers αποτελούν έναν ευφυή τρόπο της Laravel για την διαχείριση όλων των CRUD Operations σε ένα Model
- Δημιουργούνται σε μία μόνο γραμμή:
 - `php artisan make:controller PostController --resource`
- Δηλώνονται στον Router με μία μόνο γραμμή:
 - `Route::resource('photos', 'PostController');`

METHOD	URI	Action	Route Name
GET	/posts	index	posts.index
GET	/posts/create	create	posts.create
POST	/posts	store	posts.store
GET	/posts/{post}	show	posts.show
GET	/posts/{post}/edit	edit	posts.edit
PUT/PATCH	/posts/{post}	update	posts.update
DELETE	/posts/{post}	destroy	posts.destroy

- Η Laravel παρέχει έναν μηχανισμό για την δυναμική δημιουργία των URLs της εφαρμογής.

```
$post = App\Post::find(1);  
  
echo url("/posts/{$post->id}");  
  
// http://example.com/posts/1
```

```
echo route('post.show', ['post' => 1]);  
  
// http://example.com/post/1
```

```
$url = action('HomeController@index');
```

- Η διαχείριση των Sessions γίνεται με διάφορους τρόπους από τους οποίους ο προγραμματιστής επιλέγει εκείνον που εξυπηρετεί περισσότερο τις ανάγκες και τις απαιτήσεις της εφαρμογής του.
- Στην Laravel περιέχονται out-of-the-box οι εξής επιλογές:
 - **file** – Τα Sessions αποθηκεύονται ως αρχεία στο storage/framework/sessions
 - **cookie** – Τα Sessions αποθηκεύονται σε ασφαλή, κρυπτογραφημένα cookies
 - **database** – Τα Sessions αποθηκεύονται σε μια Σχεσιακή Βάση Δεδομένων
 - **memcached / redis** – Τα sessions αποθηκεύονται σε μία από τις προηγμένες γρήγορες βιβλιοθήκες
 - **array** – Τα sessions αποθηκεύονται σε ένα PHP array και δεν είναι διαρκή

- Μέσω της βιβλιοθήκης `monolog` παρέχονται δυνατές υπηρεσίες καταγραφής
- Μέσω των `logs`, ο προγραμματιστής μπορεί να εντοπίσει πιθανά προβλήματα στην εφαρμογή και στην χρήση αυτής
- Επίσης μέσω των `logs` υλοποιείται το `Audit Trail`, δηλαδή ο εντοπισμός κακόβουλων ενεργειών των χρηστών της εφαρμογής
- Τα `logs` αποθηκεύονται σε αρχείο-a και είναι απόλυτα κατανοητά από τον προγραμματιστή
- Η βιβλιοθήκη `monolog` επίσης παρέχει την δυνατότητα αυτόματης ειδοποίησης των μελών μιας ομάδας στο `Slack` σε ορισμένες περιπτώσεις (π.χ εμφάνιση `Critical Error`)

Frontend

06



01

Blade Templates

- Το Blade, είναι ένας απλός αλλα ισχυρός μηχανισμός Templating της Laravel
- Επιτρέπει την χρησιμοποίηση PHP κώδικα στα Views.
- Η ονομασία των αρχείων είναι της μορφής: `myview.blade.php`
- Τα δύο μεγάλα πλεονεκτήματα του Blade, είναι η κληρονομικότητα των templates και τα sections
- **Κληρονομικότητα** – Τα templates μπορούν για είναι χωρισμένα σε πολλά κομμάτια τα οποία καλούνται σε άλλα μεγαλύτερα ιεραρχικά templates
- **Sections** – Μέσα στα templates καθορίζονται περιοχές οι οποίες μπορούν να είναι διαφορετικές σε κάθε view

```
<!-- resources/views/layouts/app.blade.php -->

<html>
  <head>
    <title>App Name</title>
  </head>
  <body>
    @yield('content')
  </body>
</html>

<!-- resources/views/test.blade.php -->

@extends('layouts.app')

@section('content')
  <p>This is my content!</p>
@endsection
```

- Υπάρχουν δύο τρόποι για την δημιουργία τοπικοποίησης της εφαρμογής
- Η τοπικοποίηση γίνεται μέσω των Translation Strings που βρίσκονται στα αρχεία /resources/lang/{lang_code}

Short Keys

```
//resources/lang/en/messages.php  
  
return [  
    'welcome' => 'Welcome to our application'  
];
```

```
@lang('messages.welcome')
```

Translation Strings As Keys

```
//resources/lang/es.json  
  
return [  
    "I love programming.": "Me encanta programar."  
];
```

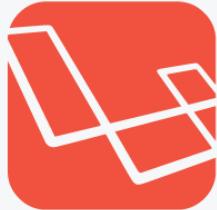
```
@lang('I love programming.')
```

- To Laravel Mix είναι ένας μηχανισμός μέσω του οποίου ορίζονται βήματα για το compiling αρχείων Javascript και CSS
- Χρησιμοποιεί ορισμένους Javascript & CSS Pre-processors που περιέχονται στο Webpack
- CSS: SASS, LESS, Stylus, PostCSS, Plain CSS
- JS: Vue.js, React, Vanilla JS

```
mix.js('resources/assets/js/app.js', 'public/js')
      .sass('resources/assets/sass/app.scss', 'public/css');
```

Βάσεις Δεδομένων

07



- Ένα χαρακτηριστικό γνώρισμα της Laravel είναι η ευκολία με την οποία γίνεται η επικοινωνία με μια πληθώρα τύπων Βάσεων Δεδομένων
- Οι τύποι που υποστηρίζονται είναι:
 - MySQL
 - PostgreSQL
 - SQLite
 - SQL Server
- Υπάρχουν βέβαια ορισμένα πακέτα που επιτρέπουν την διασύνδεση με άλλους τύπους Βάσεων Δεδομένων π.χ. (laravel-mongodb)
- Η αλληλεπίδραση με την Βάση Δεδομένων γίνεται με τρείς τρόπους:
 - Raw SQL
 - Query Builder
 - Eloquent ORM

- Τα Migrations είναι κάτι σαν Version Control για την Βάση Δεδομένων. Επιτρέπει τον εύκολο διαμοιρασμό του Database Schema με τα μέλη της ομάδας καθώς και διακρίνει τις αλλαγές που γίνονται στην πορεία του χρόνου (π.χ. προσθήκη μίας στήλης στον πίνακα)
- Η συνηθέστερη πρακτική είναι η δημιουργία ενός migration για κάθε αλλαγή που γίνεται σε κάποιο πίνακα (έννοώντας φυσικά και την δημιουργία του)
- Η δημιουργία του migration γίνεται μέσω της εντολής
php artisan make:migration migration_name
- Και εκτελούνται με την εντολή
php artisan migrate

```
class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
}
```

03

- To Seeding είναι μία μέθοδος στην οποία οι πίνακες αρχικοποιούνται με εγγραφές
- Είναι ιδιαίτερα πρακτικό κατά την διάρκεια της ανάπτυξης αλλά και για την αρχικοποίηση της εφαρμογής στο production περιβάλλον
- Ένας Seeder μπορεί να δημιουργεί εγγραφές τόσο χειροκίνητα όσο και αυτόματα με την χρησιμοποίηση κατάλληλων πακέτων (π.χ Faker)

```
class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }

    /**
     * Or with model Factories
     *
     * @return void
     */
    public function run()
    {
        factory(App\User::class, 50)->create()->each(function ($u) {
            $u->posts()->save(factory(App\Post::class)->make());
        });
    }
}
```

- Ο πρώτος τρόπος επικοινωνίας με την Βάση Δεδομένων είναι μέσω Raw Queries, δηλαδή με SQL Queries
- Υλοποιείται μέσω του Facade **DB** που υποστηρίζει τις μεθόδους **select**, **update**, **insert**, **delete**, **statement**
- Πρακτικά η μέθοδος **statement** δέχεται όλα τα SQL Queries που δεν ικανοποιούνται από τις προηγούμενες μεθόδους
- Είναι η τελευταία επιλογή καθώς αυξάνει την πολυπλοκότητα του κώδικα

```
$users = DB::select('select * from users where active = ?', [1]);
```

```
DB::statement('drop table users');
```

- Το Laravel Query Builder είναι ένα εύχρηστο interface το οποίο επιτρέπει την αλληλεπίδραση με την Βάση Δεδομένων μειώνοντας την πολυπλοκότητα του κώδικα
- Είναι μια λύση ανάμεσα στα Raw Queries και στο Eloquent ORM
- Αποδεικνύεται ιδιαίτερα βολικό όταν θέλουμε να εκτελέσουμε ενέργειες σε έναν πίνακα που ενδεχομένως δεν καλύπτονται από το Eloquent ORM

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

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

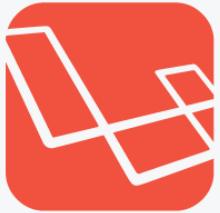
```
$users = DB::table('users')->select('name', 'email as user_email')->get();
```

- Το Eloquent ORM προσφέρει μια απλή και όμορφη εφαρμογή του **Active Record**^[1] για την αλληλεπίδραση με την Βάση Δεδομένων.
- Κάθε Model της Laravel είναι ουσιαστικά ένα Eloquent Model
- Μέσω το Eloquent επιτυγχάνεται καθαρότητα του κώδικα, γρηγορότερη ανάπτυξη και ταχύτερη αλληλεπίδραση

[1]: Κάθε πίνακας έχει ένα αντίστοιχο Model το οποίο χρησιμοποιείται για την αλληλεπίδραση με τον πίνακα

Eloquent ORM

08



- Κάθε πίνακας της Βάσης Δεδομένων αναπαριστάται από το αντίστοιχο Eloquent Model το οποίο και δέχεται εξ' ορισμού CRUD Operations
- Επίσης εξ' ορισμού το Eloquent ORM κάνει τις ακόλουθες παραδοχές:
 - Το όνομα του Model είναι το όνομα του πίνακα στον ενοικό (Post – posts)
 - Το κύριο κλειδί κάθε πίνακα είναι η στήλη id (INCREMENT INT)
 - Κάθε πίνακας περιέχει timestamps (created_at και updated_at)
- Φυσικά όλες οι παραπάνω παραδοχές μπορούν να παρακαμφθούν από τον προγραμματιστή με custom τιμές

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

$flights = Flight::where('active', 1)
    ->orderBy('name', 'desc')
    ->take(10)
    ->get();

foreach (Flight::where('foo', 'bar')->cursor() as $flight) {
    //
}
```

03

CRUD Operations / Create

```
class FlightController extends Controller
{
    /**
     * Create a new flight instance.
     *
     * @param Request $request
     * @return Response
     */
    public function store(Request $request)
    {
        //Create the model instance
        $flight = new Flight;

        //Set the attributes
        $flight->name = $request->name;

        //Save the record
        $flight->save();
    }
}
```

```
class FlightController extends Controller
{
    /**
     * Create a new flight instance.
     *
     * @param Request $request
     * @return Response
     */
    public function update(Request $request)
    {
        //Find the flight with primary key "1"
        $flight = App\Flight::find(1);

        //Set the updated attribute
        $flight->name = 'New Flight Name';

        //Save the record
        $flight->save();
    }
}
```

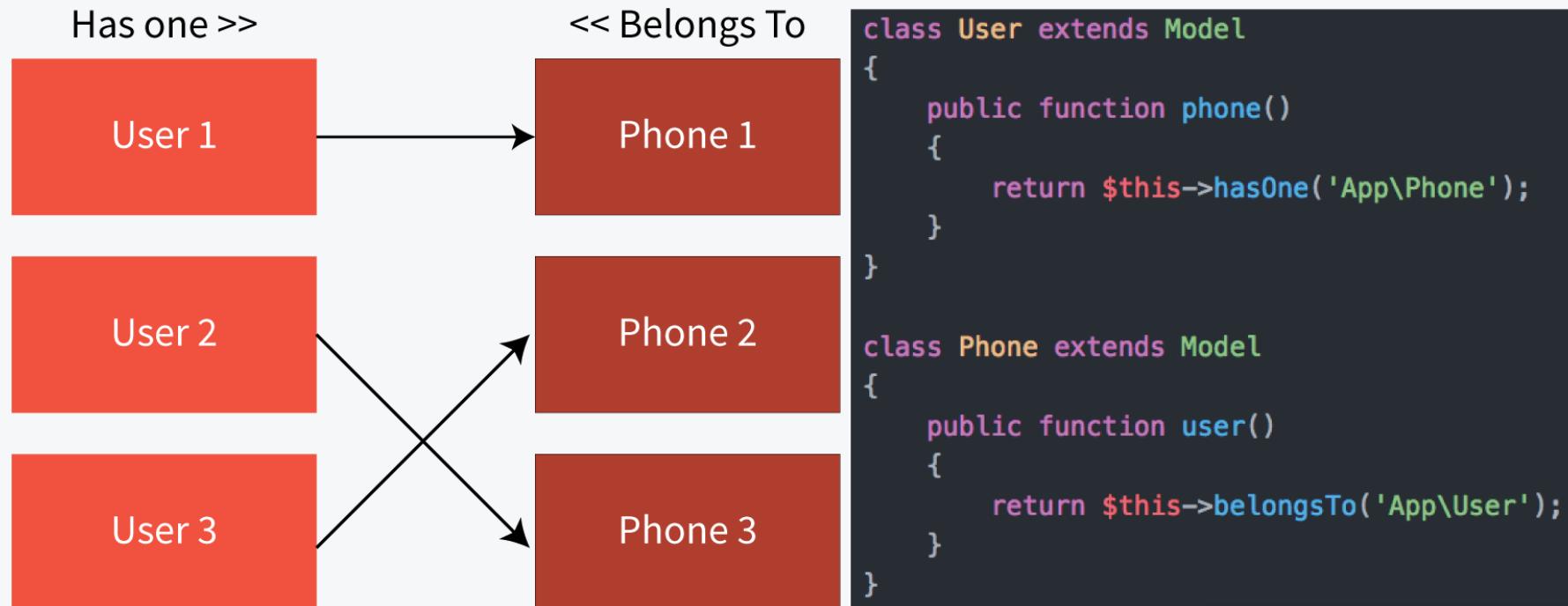
```
class FlightController extends Controller
{
    /**
     * Create a new flight instance.
     *
     * @param Request $request
     * @return Response
     */
    public function update(Request $request)
    {
        //Find the flight with primary key "1"
        $flight = App\Flight::find(1);

        //Delete the record
        $flight->delete();

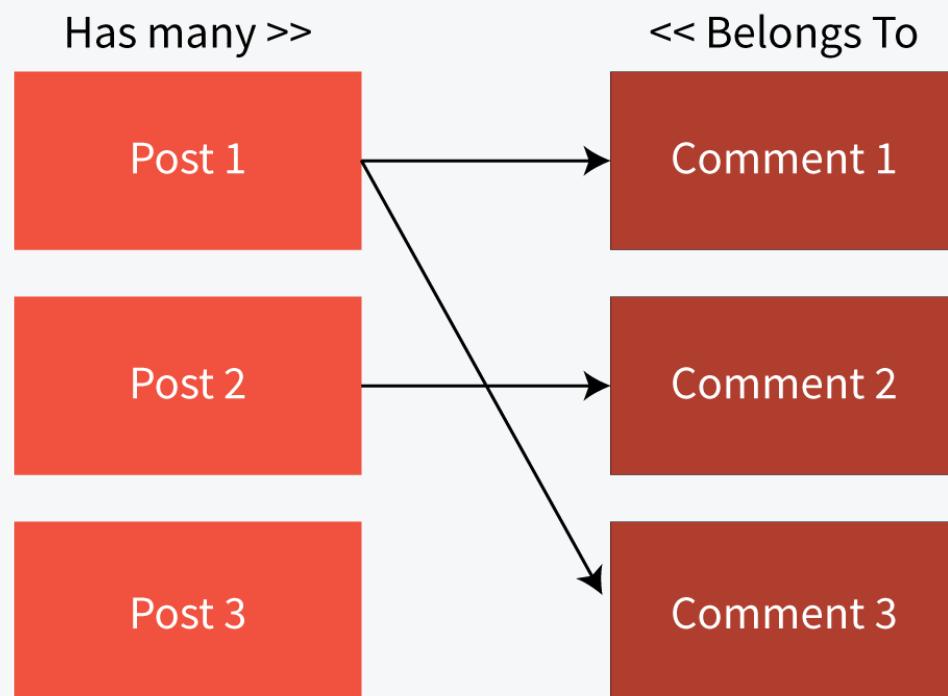
        //Delete the flights with primary keys "2", "3", "4"
        //This approach does not require the retrieval of the record
        App\Flight::destroy([2, 3, 4]);
    }
}
```

- Συχνά υπάρχουν αναφορές ανάμεσα στους πίνακες των Σχεσιακών Βάσεων Δεδομένων. Μέσω του Eloquent η διαχείριση αυτών των σχέσεων γίνεται εξαιρετικά απλή και εύκολη.
- Τα είδη των σχέσεων που υποστηρίζονται από το Eloquent είναι:
 - One To One ('Ενα προς 'Ένα)
 - One To Many ('Ενα προς Πολλά)
 - Many To Many (Πολλά προς Πολλά)
 - Has Many Through (Πολλά μέσω)
 - Polymorphic (Πολυμορφικές)
 - Many To Many Polymorphic (Πολλά προς Πολλά Πολυμορφικές)
- Τα Eloquent Relationships δηλώνονται ως μέθοδοι μέσα στο Model

- Η One To One είναι η πιο βασική σχέση. Για παράδειγμα **ένα** Model User σχετίζεται με **ένα** Phone



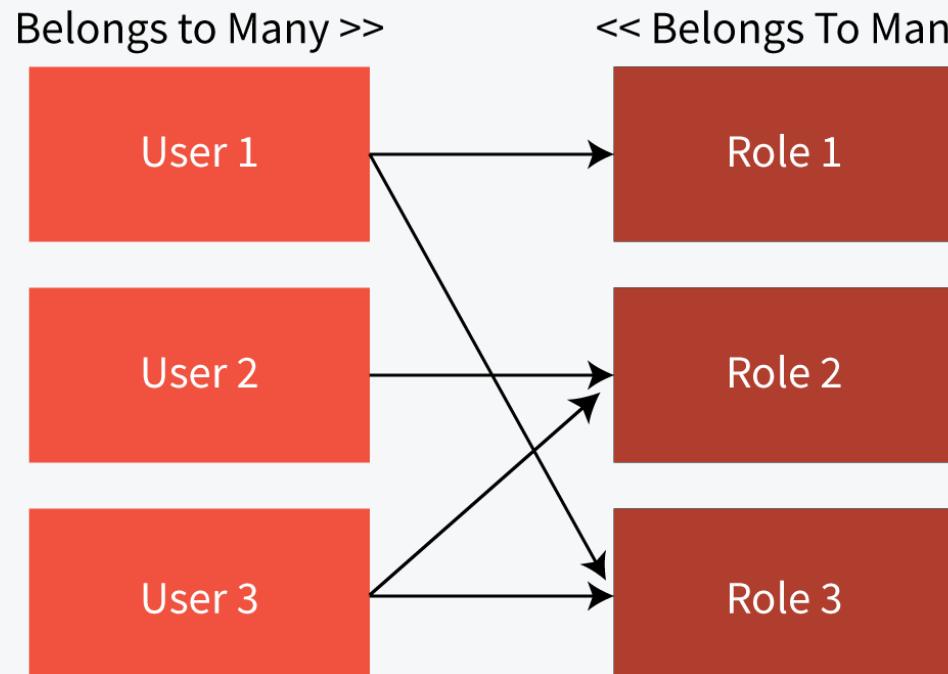
- Στην One To Many ένα Model σχετίζεται με πολλά άλλα Models. Για παράδειγμα **ένα Post** έχει **έναν άπειρο αριθμό** από **Comments**



```
class Post extends Model
{
    public function comments()
    {
        return $this->hasMany('App\Comment');
    }
}

class Comment extends Model
{
    public function post()
    {
        return $this->belongsTo('App\Post');
    }
}
```

- Η Many To Many είναι μια πιο πολύπλοκη σχέση από τις προηγούμενες. Για παράδειγμα είναι ένας χρήστης με πολλούς ρόλους, αλλά αυτοί οι ρόλοι μοιράζονται και από άλλους χρήστες. Για αυτή την υλοποίηση χρειαζόμαστε 3 πίνακες: **users**, **roles**, **role_user** (περιέχει user_id, role_id)



```
class User extends Model
{
    public function roles()
    {
        return $this->belongsToMany('App\Role');
    }
}

class Role extends Model
{
    public function users()
    {
        return $this->belongsToMany('App\User');
    }
}
```

- Η Has Many Through είναι μια βολική παράκαμψη για την πρόσβαση σε απομακρυσμένες σχέσεις μέσω μιας ενδιάμεσης σχέσης. Για παράδειγμα ένα **Country** μπορεί να έχει πολλά **Post** μέσω ενός ενδιάμεσου **User**

```
countries
    id - integer
    name - string

users
    id - integer
    country_id - integer
    name - string

posts
    id - integer
    user_id - integer
    title - string

class Country extends Model
{
    public function posts()
    {
        return $this->hasManyThrough('App\Post', 'App\User');
    }
}
```

- Οι πολυμορφικές σχέσεις επιτρέπουν σε ένα Model να ανήκει σε περισσότερα από ένα Model σε μία συσχετιση. Για παράδειγμα οι χρήστες μπορούν να σχολιάζουν τόσο σε άρθρα όσο και σε βίντεο. Με τις πολυμορφικές σχέσεις μπορούμε να χρησιμοποιούμε μόνο έναν πίνακα **comments** και για τις δύο περιπτώσεις.

```
class Comment extends Model
{
    public function commentable()
    {
        return $this->morphTo();
    }
}

class Post extends Model
{
    public function comments()
    {
        return $this->morphMany('App\Comment', 'commentable');
    }
}

class Video extends Model
{
    public function comments()
    {
        return $this->morphMany('App\Comment', 'commentable');
    }
}
```

posts
id - integer
title - string
body - text

videos
id - integer
title - string
url - string

comments
id - integer
body - text
commentable_id - integer
commentable_type - string

- Μπορούν να υπάρχουν και Many To Many πολυμορφικές σχέσεις. Για παράδειγμα ένα Model **Post** και ένα Model **Video** μπορούν να μοιράζονται την ίδια πολυμορφική σχέση με ένα Model **Tag**. Χρησιμοποιώντας αυτόν τον τύπο σχέσης μπορούμε να έχουμε πρόσβαση σε μια λίστα μοναδικών tags τα οποία υπάρχουν και στα posts και στα video

```
class Post extends Model
{
    public function tags()
    {
        return $this->morphToMany('App\Tag', 'taggable');
    }
}

class Tag extends Model
{
    public function posts()
    {
        return $this->morphedByMany('App\Post', 'taggable');
    }

    public function videos()
    {
        return $this->morphedByMany('App\Video', 'taggable');
    }
}
```

posts
id - integer
name - string

videos
id - integer
name - string

tags
id - integer
name - string

taggables
tag_id - integer
taggable_id - integer
taggable_type - string

- Κατά την πρόσβαση Eloquent σχέσεων σαν ιδιότητες, τα δεδομένα της σχέσης φορτώνονται με την μέθοδο του “lazy load”, δηλαδή τα δεδομένα αυτά δεν φορτώνονται στην πραγματικότητα μέχρι να ζητηθούν από την εφαρμογή.
- To Eloquent μπορεί να φορτώσει με τις σχέσεις με την μέθοδο του “eager loading”, δηλαδή την στιγμή που γίνεται το query στο model-γονέα.
- To eager loading ελαττώνει το πρόβλημα των N+1 queries.

```
$books = App\Book::with('author')->get();

foreach ($books as $book) {
    echo $book->author->name;
}
```

- Όταν επιστρέφεται ένα σετ αποτελεσμάτων από το Eloquent αυτό αποτελεί ένα αντικέιμενο του τύπου `Illuminate\Database\Eloquent\Collection`.

- Το γεγονός αυτό βοηθάει στο ότι η Laravel παρέχει αρκετές μεθόδους που κάνουν ευκολότερη την διαχείριση των αποτελεσμάτων.

<code>all</code>	<code>intersect</code>	<code>reverse</code>
<code>average</code>	<code>isEmpty</code>	<code>search</code>
<code>avg</code>	<code>isNotEmpty</code>	<code>shift</code>
<code>chunk</code>	<code>keyBy</code>	<code>shuffle</code>
<code>collapse</code>	<code>keys</code>	<code>slice</code>
<code>combine</code>	<code>last</code>	<code>sort</code>
<code>concat</code>	<code>map</code>	<code>sortBy</code>
<code>contains</code>	<code>mapInto</code>	<code>sortByDesc</code>
<code>containsStrict</code>	<code>mapSpread</code>	<code>splice</code>
<code>count</code>	<code>mapToGroups</code>	<code>split</code>
<code>crossJoin</code>	<code>mapWithKeys</code>	<code>sum</code>
<code>dd</code>	<code>max</code>	<code>take</code>
<code>diff</code>	<code>median</code>	<code>tap</code>
<code>diffKeys</code>	<code>merge</code>	<code>toArray</code>
<code>dump</code>	<code>min</code>	<code>toJson</code>
<code>each</code>	<code>mode</code>	<code>transform</code>
<code>eachSpread</code>	<code>nth</code>	<code>union</code>
<code>every</code>	<code>only</code>	<code>unique</code>
<code>except</code>	<code>pad</code>	<code>uniqueStrict</code>
<code>filter</code>	<code>partition</code>	<code>unless</code>
<code>first</code>	<code>pipe</code>	<code>values</code>
<code>flatMap</code>	<code>pluck</code>	<code>when</code>
<code>flatten</code>	<code>pop</code>	<code>where</code>
<code>flip</code>	<code>prepend</code>	<code>whereStrict</code>
<code>forget</code>	<code>pull</code>	<code>whereIn</code>
<code>forPage</code>	<code>push</code>	<code>whereInStrict</code>
<code>get</code>	<code>put</code>	<code>whereNotIn</code>
<code>groupBy</code>	<code>random</code>	<code>whereNotInStrict</code>
<code>has</code>	<code>reduce</code>	<code>zip</code>
<code>implode</code>	<code>reject</code>	

- Οι Accessors και οι Mutators βοηθάνε στην διαμόρφωση των χαρακτηριστικών όταν επιστρέφονται ή οταν ορίζονται αντίστοιχα

```
class User extends Model
{
    /**
     * Get the user's first name.
     * This is an accessor.
     * @param string $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}

class User extends Model
{
    /**
     * Set the user's first name.
     * This is a mutator.
     * @param string $value
     * @return void
     */
    public function setFirstNameAttribute($value)
    {
        $this->attributes['first_name'] = strtolower($value);
    }
}
```



Digging Deeper

09



- Η Laravel κάνει την υλοποίηση ενός συστήματος αυθεντικοποίησης να μοιάζει πάρα πολύ απλή!
- Τρέχοντας την απλά την εντολή
php artisan make:auth
- Αυτόματα η Laravel δημιουργεί για εμάς όλα τα απαραίτητα Models, Views, Controllers, Middleware κ.τ.λ που χρειάζονται για ένα πλήρες και κυρίως ασφαλές σύστημα αυθεντικοποίησης

- Όπως είπαμε η υποστήριξη της λειτουργίας Cache στην εφαρμογή μας υποστηρίζεται και ενθαρύνεται από το Framework με διάφορες μεθόδους
- Η Cache λειτουργεί είτε με την αποθήκευση αρχείων, είτε με βάση δεδομένων είτε χρησιμοποιώντας διάφορα διάσημα Caching Backends όπως το Redis και το Memcached
- Η χρήση της Cache βελτιώνει αισθητά τους χρόνους απόκρισης και επεξεργασίας της εφαρμογής

- Τα events της Laravel παρέχουν μια απλή υλοποίηση παρατηρητή (observer), επιτρέποντας την εγγραφή και της ακρόαση διάφορων γεγονότων που συμβαίνουν κατά την εκτέλεση της εφαρμογής
- Οι κλάσεις των events αποθηκεύονται στον κατάλογο “`app/Events`” ενώ οι αντίστοιχοι listeners στον κατάλογο “`app/Listeners`”.
- Τα events αποτελούν έναν σπουδαίο τρόπο για τον διαχωρισμό διάφορων χαρακτηριστικών της εφαρμογής μιας και ένα μοναδικό event μπορεί να έχει πολλούς listeners οι οποίο δεν εξαρτώνται και δεν συνδέονται μεταξύ τους.

- Η χρήση του E-mail σε μια Laravel εφαρμογή είναι απλή υπόθεση. Και αυτό γιατί παριέχει ένα απλό API για την βιβλιοθήκη SwiftMail
- Οι drivers που υποστηρίζει είναι **SMTP, Mailgun, SparkPost, Amazon SES**, μέθοδος **mail()** της PHP
- Έτσι ο προγραμματιστής ασχολείται περισσότερο με την δημιουργία των emails παρά με την ρύθμιση τους
- Για κάθε τύπο email που αποστέλλεται από την εφαρμογή, αυτό αναπαρίσταται από μια “mailable” κλάση η οποία βρίσκεται στον κατάλογο “app/Mail”

- Εκτός από την αποστολή mail υποστηρίζεται και η αποστολή ειδοποιήσεων μέσω διαφόρων καναλιών επικοινωνίας όπως e-mail, SMS, Slack ή/και με αποθήκευση στην Βάση Δεδομένων και προβολή στο Web Interface

- Μέσω των Queues επιτρέπεται η αναβολή ορισμένων εργασιών που απαιτούν μεγάλη επεξεργασία και χρόνο (π.χ. αποστολή email) μέχρι κάποιο χρονικό διάστημα.
- Το γεγονός αυτό αυξάνει αισθητά την ταχύτητα απόκρισης της εφαρμογής όταν δέχεται Web Requests
- Οι υλοποιήσεις που υποστηρίζονται είναι:
 - Βάση Δεδομένων
 - Beanstalk
 - Amazon SQS
 - Redis

- Σε ένα Laravel Application όλες οι εργασίες που πρέπει να εκτελούνται με συγκεκριμένο πρόγραμμα καθορίζονται μέσα στην ίδια την εφαρμογή.
- Έτσι αντί να χρειάζεται και από ένα cron για κάθε εργασία μέσω αυτής της οδού πρέπει να καθοριστεί μόναχα ένα cron:

** * * * * php /path-to-your-project/artisan schedule:run >> /dev/null 2>&1*
το οποίο ουσιασικά «ακούει» κάθε ένα λεπτό το Laravel Scheduler

Τέλος Α' Μέρους

Ερωτήσεις;

LGthree

Πτυχιακή εργασία του Καρπούζη Κυριάκου

Αλεξάνδρειο Τ.Ε.Ι. Θεσσαλονίκης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

Θεσσαλονίκη 2018

Μέρος Β'



Էլօգայոն

01



01

Τι είναι το LGThree;

- Laravel Web Application παροχής πληροφοριών για τα αεροδρόμια του ελληνικού FIR
- Αναπτύχθηκε στα πλαίσια της πτυχιακής εργασίας για το τμήμα Μηχανικών Πληροφορικής Τ.Ε. του ΑΤΕΙΘ

- Παροχή χρήσιμων πληροφοριών σε πιλότους και ελεγκτές εναέριας κυκλοφορίας που χρησιμοποιούν τα ελληνικά αεροδρόμια
- Υπολογισμός του ενεργού διαδρόμου προσγείωσης-απογείωσης
- Real-Time ενημέρωση και ειδοποίηση στην περίπτωση αλλαγής του καιρού του αεροδρομίου
- Πλήρως λειτουργικό περιβάλλον διαχείρισης

- Laravel – PHP Framework
- mySQL – Σύστημα Διαχείρισης ΣΒΔ
- Vue.js – JavaScript Framework
- jQuery – JavaScript Library
- Sass – Style Sheet Language
- npm – JavaScript Package Manager

- Bootstrap – Frontend Framework
- Webpack – JavaScript Module Bundler
- Pusher – API για real-time ειδοποιήσεις
- Laravel Echo – JavaScript Library για την παρακολούθηση γεγονότων

Χρήσιμες ορολογίες

02



- **Meteorological Terminal Air Report (METAR)**
- Περιγράφει τις τρέχουσες καιρικές συνθήκες ένος σταθμού παρατήρησης (στην περίπτωση μας του αεροδρομίου)
- Ακολουθεί τα πρότυπα του ICAO^[1], συνεπώς είναι εύκολο να κατανοηθεί σε όλο τον πλανήτη
- Σε φυσιολογικές συνθήκες εκδίδεται στις :20 και :50 κάθε ώρας, εκτός και αν υπάρχει έκτακτη μεταβολή στις παρατηρήσεις

[1]: International Civil Aviation Organization

LGTS 181350Z 16013KT 6000 FEW018 BKN030 15/04 Q1011 NOSIG

- **LGTS**: ICAO Code του αεροδρομίου
- **18**: Ημέρα έκδοσης
- **1350Z**: Ώρα έκδοσης σε UTC
- **160**: Κατεύθυνση ανέμου σε μοίρες
- **13KT**: Ταχύτητα ανέμου σε κόμβους
- **6000**: Ορατότητα σε χλμ
- **FEW018**: Επίπεδο νεφών στα 1800 πόδια με κάλυψη ~1/8 ουρανού

- **BKN030**: Επίπεδο νεφών στα 3000 πόδια με κάλυψη ~7/8 ουρανού
- **15/04**: Θερμοκρασία και σημείο δρόσου σε βαθμούς Κελσίου
- **Q1011**: Η ατμοσφαιρική πίεση σε hPa
- **NOSIG**: Δεν αναμένεται κάποια σημαντική αλλαγή

- Terminal Aerodrome Forecast (TAF)
- Περιγράφει την πρόγνωση του καιρού για το εκάστοτε αεροδρόμιο
- Εκδίδεται συνήθως 4 φορές την ημέρα
- Περιέχει την πρόγνωση για μία περίοδο συνήθως 24 ή 30 ωρών για μία περιοχή 5sm (8km) πέριξ του αεροδρομίου

**TAF LGTS 040500Z 0406/0506 VRB03KT 9999 FEW030 BECMG
0412/0414 16010KT BECMG 0418/0420 VRB03KT**

[1]: International Civil Aviation Organization

■ NOTice To AirMen (NOTAM)

- Είναι μια ειδοποίηση που περιέχει πληροφορίες σχετικά με τις εγκαταστάσεις, τις συνθήκες και τις αλλαγές που αφορούν τον καθένα που συμμετέχει σε υπηρεσίες πτήσεων

A1234/06 NOTAMR A1212/06

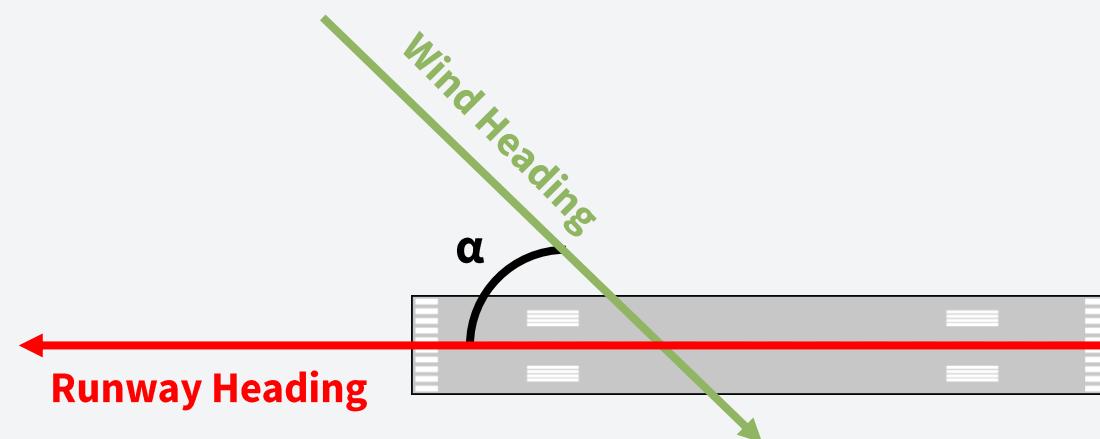
Q) EGTT/QMXLC/IV/NBO/A/000/999/5129N00028W005
A) EGLL
B) 0609050500
C) 0704300500
E) DUE WIP TWY B SOUTH CLSD BTN 'F' AND 'R'. TWY 'R' CLSD BTN 'A' AND 'B' AND DIVERTED VIA NEW GREEN CL AND BLUE EDGE LGT. CTN ADZ

- Για την επιλογή του ενεργού διαδρόμου προσγείωσης-απογείωσης λαμβάνονται υπ' όψην αρκετές παράμετροι με συνηθέστερες και καθοριστικότερες την κατεύθυνση και την ένταση του ανέμου
- Η απόφαση για τον χαρακτηρισμό ενός διαδρόμου ως ενεργό λαμβάνεται από τον ελεγκτή εναέριας κυκλοφορίας (συγκεκριμένα τον ελεγκτή στην θέση TWR) σε ελεγχόμενα αεροδρόμια. Σε περίπτωση απουσίας λειτουργικού πύργου ελέγχου υπεύθυνος είναι ο πιλότος.
- Για χάρη ευκολίας θα υποθέσουμε πώς η κατεύθυνση και η ταχύτητα του ανέμου είναι το μοναδικά κριτήρια για την επιλογή του ενεργού διαδρόμου

- Για κάθε διάδρομο υπάρχει το αντίστοιχο Wind Component που περιγράφεται από τον τύπο:

$$\text{WindComponent} = \text{WindSpeed} * \cos(\alpha)$$

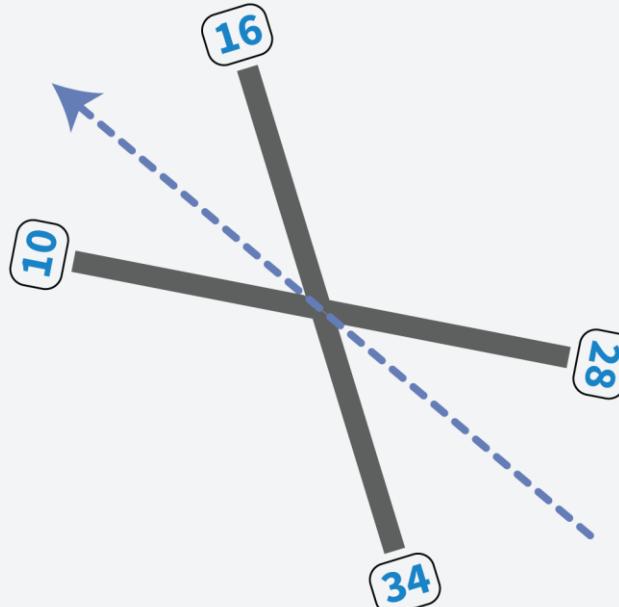
$$\text{όπου } \alpha = \text{RunwayHeading} - \text{WindHeading}$$



- Σε περίπτωση που το Wind Component είναι θετικό τότε λέμε ότι ο διάδρομος έχει Μετωπικό άνεμο (Headwind)
- Αν το αποτέλεσμα είναι αρνητικό τότε ο διάδρομος έχει Ούριο άνεμο (Tailwind)
- Ως ενεργός επιλέγεται ο διάδρομος με το μεγαλύτερο Headwind.

Διάδρομοι: **10**(101°), **16**(163°),
28(281°), **34**(343°)

Άνεμος: $140^\circ / 10\text{KT}$



10: 8	(+)	
16: 9	(+)	Ενεργός
28: -8	(-)	
34: -9	(-)	

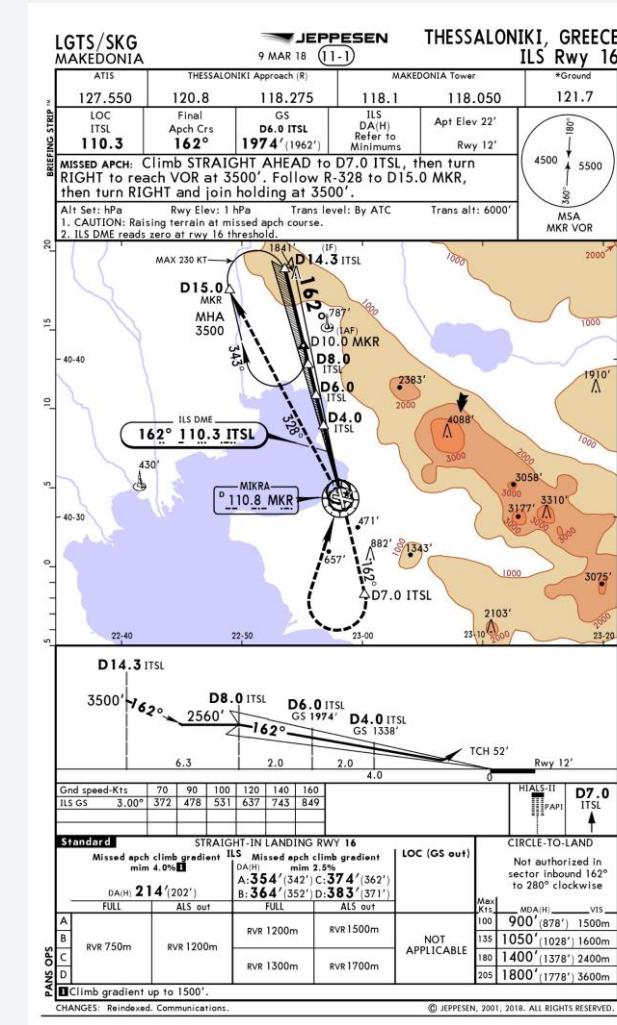
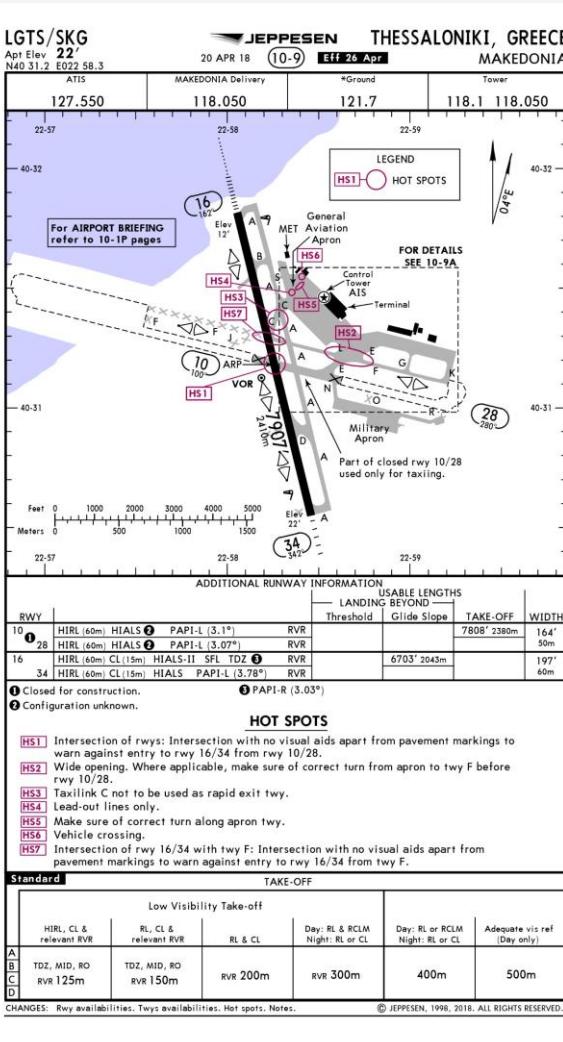
- SID – **S**tandard **I**nstrument **Departure, μια καθορισμένη διαδρομή που ακολουθεί το αεροσκάφος για να μεταβεί από την φάση της απογείωσης στην φάση της πορείας**
- STAR – **S**tandard **A**rrival **R**oute, μια καθορισμένη διαδρομή που ακολουθεί το αεροσκάφος για να μεταβεί από την φάση πορείας στο αρχικό σημείο προσέγγισης
- Χρησιμοποιούνται από τους ΕΕΚ για τον καλύτερο χειρισμό και ροή της κυκλοφορίας σε αεροδρόμια με μεγάλη κίνηση

06

Charts

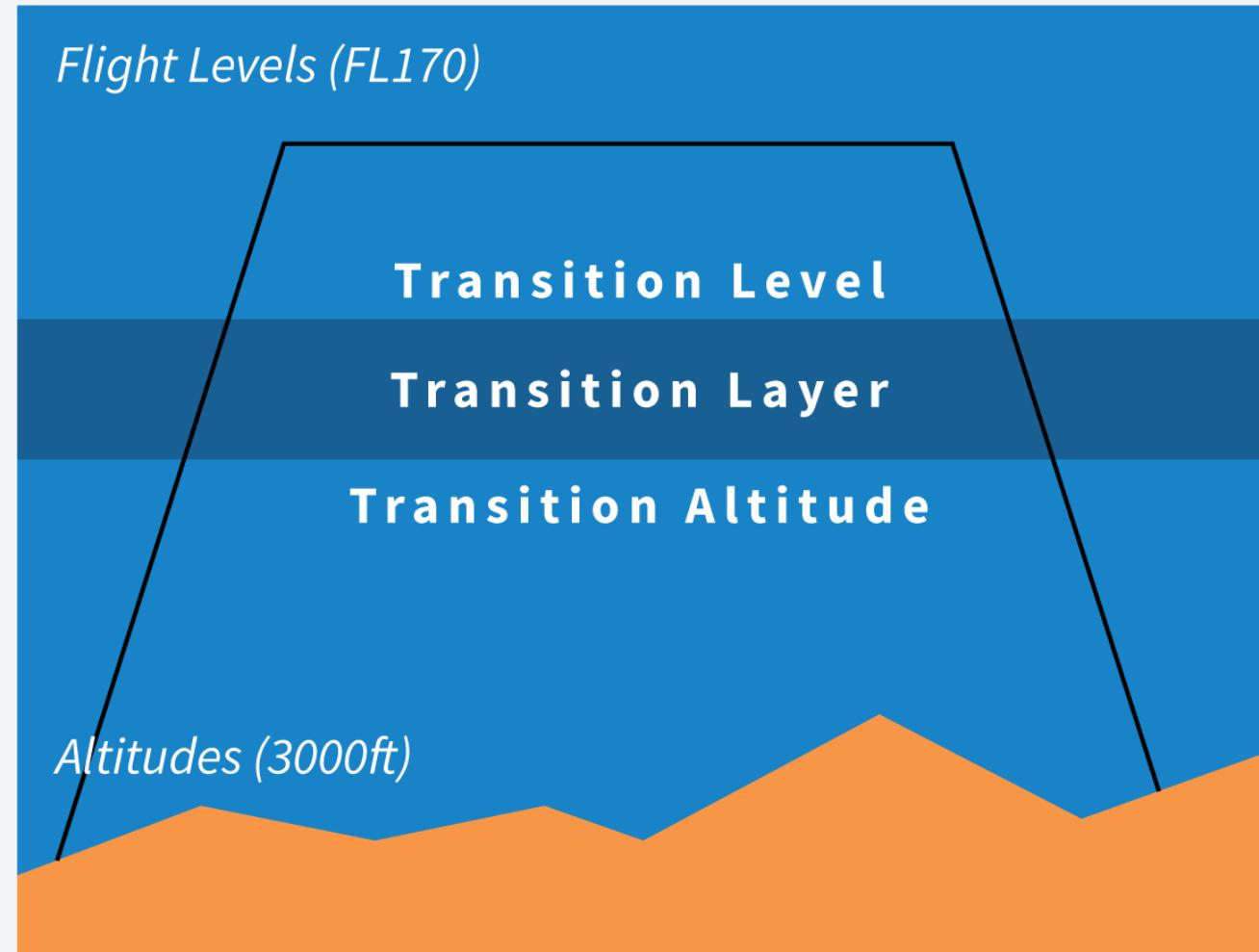
■ Τα Charts είναι τα έγγραφα που χρησιμοποιούνται από τους χειριστές των αεροσκαφών και περιγράφουν όλα τα χαρακτηριστικά των αεροδρομίων ή ενός FIR

- Διαχωρίζονται σε κατηγορίες όπως πληροφορίες αεροδρομίου, αναχώρησης, άφιξης, τροχοδρόμησης κ.τ.λ



- Το QNH προσδιορίζει την ατμοσφαιρική πίεση με βάση το επίπεδο της θάλασσας (MSL)
- Η μονάδα μέτρησης στην Ευρώπη είναι το hectopascal (hPa) ή millibar
- Στην ελληνική φρασεολογία προσδιορίζεται ως «τοπική βαρομετρική»
- Παίζει πολύ μεγάλο ρόλο στην ορθή αναφορά των υψομέτρων στα οποία βρίσκονται τα αεροσκάφη
- Η standard τιμή του QNH είναι τα 1013.2 hPa

- Transition Altitude – Το υψόμετρο από το οποίο από αυτό και κάτω, η κάθετη θέση ενός αεροσκάφους μετριέται με αναφορά στο υψόμετρο (πόδια)
- Transition Level – Το χαμηλότερο Flight Level που είναι διαθέσιμο πάνω από το Transition Altitude
- Transition Layer – Ο χώρος ανάμεσα στο TA και στο TRL
- Το TA στην Ευρώπη καθορίζεται ανάλογα το αεροδρόμιο ενώ το TRL με βάση το QNH



Χαρακτηριστικά & Τεχνολογίες

03

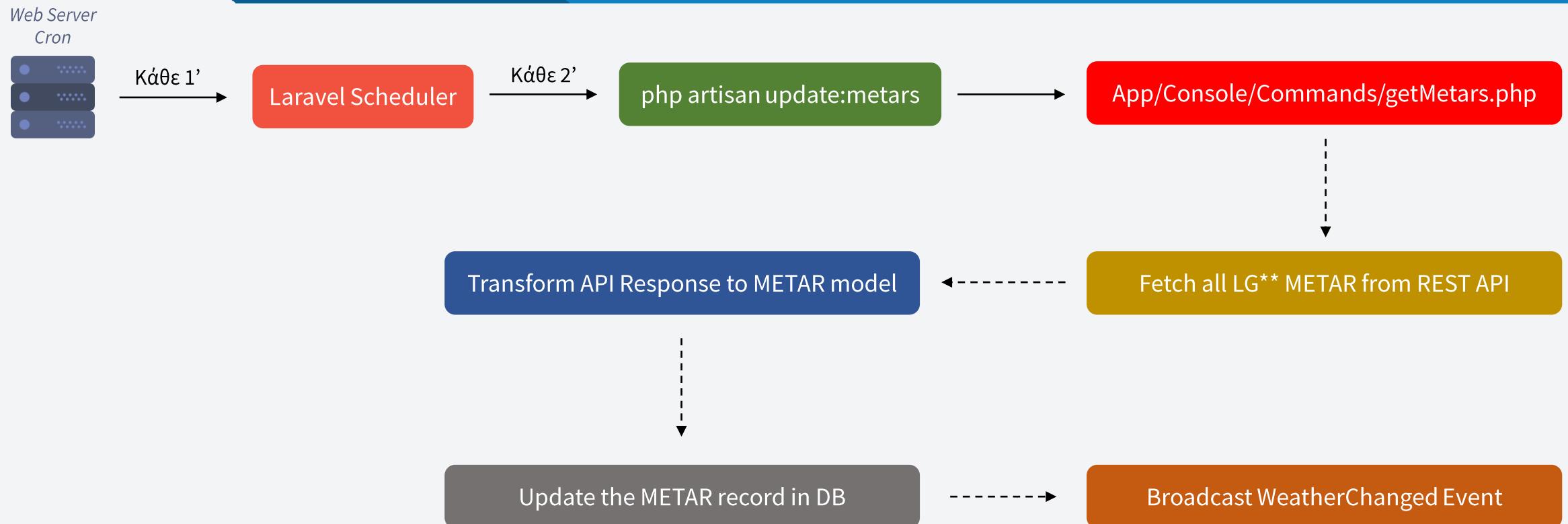


- Η εφαρμογή παρέχει πληροφορίες σχετικά με όλα τα αεροδρόμια που βρίσκονται εντός του ελληνικού FIR
- Οι πληροφορίες αυτές περιέχουν μεταξύ άλλων τα γενικά στοιχεία του αεροδρομίου (ICAO Code, όνομα, πόλη κ.α.), διαδρόμους, METAR, TAF, NOTAM, SID, STAR κ.α.
- Λειτουργικά κάθε φόρα που ο χρήστης δημιουργεί ένα HTTP Request για κάποιο αεροδρόμιο (/airports/{icao}) επιστρέφονται από την ΒΔ οι ανάλογες εγγραφές για το αεροδρόμιο αυτό.
- Κάποιες πληροφορίες παρουσιάζονται στατικά ενώ άλλες παρουσιάζονται και τροποποιούνται κατά το runtime της εφαρμογής

- Ορισμένες από τις πληροφορίες μεταβάλλονται διαρκώς στην διάρκεια της ημέρας (π.χ. METAR)
- Για αύτες τις πληροφορίες χρησιμοποιείται το Laravel Scheduler, μέσω του οποίου τρέχουν οι ανάλογες artisan εντολές οι οποίες ζητούν και διαχειρίζονται τις αντίστοιχες απαντήσεις
- Το scheduler περιέχει 3 εντολές:
 - php artisan update:metars – Ενημέρωση των METAR
 - php artisan update:tafs – Ενημέρωση των TAF
 - php artisan update:notams – Ενημέρωση των NOTAM
- Η λογική των 3 εντολών είναι παρόμοια:
Fetch – Transform - Update

03

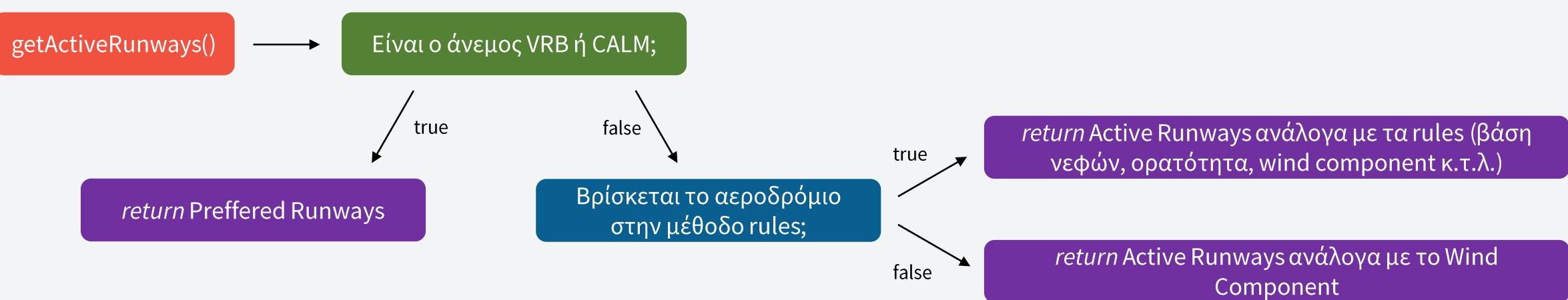
Παράδειγμα: Ενημέρωση METAR



- Η επιλογή του ενεργού διαδρόμου γίνεται μέσω της μεθόδου `Airport::getActiveRunways()`, η οποία περιγράφεται στο trait `ActiveRunwayLogic` και το οποίο χρησιμοποιείται από το model `Airport`
- Η κλήση της μεθόδου γίνεται σε δύο σημεία:
 - Κατά το Request για ένα αεροδρόμιο (`/airports/{icao}`)
 - Κατά την δημιουργία του event `WeatherChanged`
- To trait `ActiveRunwayLogic` περιέχει όλη την λογική για τον υπολογισμό και την επιστροφή του ενεργού διαδρόμου για κάθε αεροδρόμιο

05

Υπολογισμός Ενεργού Διαδρόμου



Προσοχή! Το παραπάνω διάγραμμα περιέχει την λογική του trait στην υπεραπλουστευμένη μορφή της!

App/Repositories/ActiveRunwayLogic.php

- Τα Vue Components είναι επαναχρησιμοποιούμενα κομμάτια που περιέχουν τόσο το UI (<template>) όσο και την λογική (<script>)
- Βοηθούν τόσο στον κατακερματισμό του κώδικα σε μικρότερα κομμάτια όσο και στην υλοποίηση και επεξεργασία των πληροφοριών που περιέχουν μέσω του Vue.js
- Η εφαρμογή χρησιμοποιεί Vue Components για όλες τις πληροφορίες που ενημερώνονται δυναμικά είτε λόγω ενημέρωσης είτε λόγω αλληλεπίδρασης του χρήστη

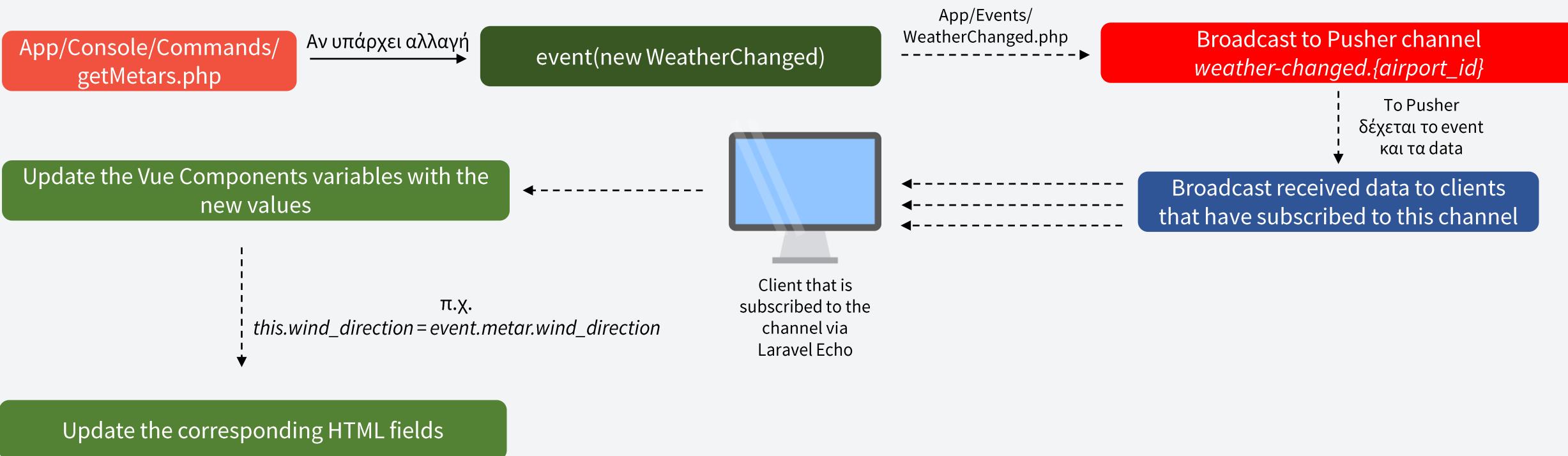
```
Vue.component('button-counter', {  
  data: function () {  
    return {  
      count: 0  
    }  
  },  
  template: '<button v-on:click="count++">You clicked me {{  
    count }} times.</button>'  
})
```

```
<div id="components-demo">  
  <button-counter></button-counter>  
</div>  
  
new Vue({ el: '#components-demo' })
```

- Όπως αναφέρθηκε η εφαρμογή ενημερώνεται αυτόματα κάθε φορά που υπάρχει κάποια αλλαγή στις καιρικές συνθήκες (δηλαδή στο METAR)
- Τα σημεία που ενημερώνονται είναι οι ενεργοί διάδρομοι, το METAR, το transition level
- Η ενημέρωση αυτή υλοποιείται μέσω ενός Laravel Event που περιέχει όλες τις αλλαγές και τους νέους υπολογισμούς
- Για την μετάδοση του Event από τον Server στους Client χρησιμοποιείται το Pusher
- Για την ακρόαση του Event από τους Client χρησιμοποιείται το Laravel Echo και για την διαχείριση του η Vue.js
- Η διαχείριση του event γίνεται στην πλευρά του client μέσω των Vue Components που ενημερώνονται δυναμικά κάθε φορά που γίνεται εκπομπή ενός γεγονότος

07

Real Time Updates (2/2)



- Η εφαρμογή διαθέτει ένα πλήρες περιβάλλον διαχείρισης για την προσθήκη, τροποποίηση και διαγραφή εγγραφών από την ΒΔ
- Επίσης περιέχει πρόσθετες δυνατότητες όπως διαχείριση χρηστών, Test καιρού κ.α.
- Το περιβάλλον διαχείρισης επιτρέπει τον πολύ εύκολο χειρισμό των δεδομένων της εφαρμογής αφού είναι εύχρηστο, απλό και κατανοητό
- Όπως και το Frontend, η υλοποίηση του έγινε χρησιμοποιώντας εργαλέα όπως Laravel, Vue.js, mySQL κ.τ.λ.
- Η πρόσβαση στο Admin Panel ελέγχεται μέσω του middleware “admin” το οποίο ουσιαστικά ελέγχει αν **Auth::user() -> isAdmin**

- Η εφαρμογή διαθέτει ένα πλήρες περιβάλλον διαχείρισης για την προσθήκη, τροποποίηση και διαγραφή εγγραφών από την ΒΔ
- Επίσης περιέχει πρόσθετες δυνατότητες όπως διαχείριση χρηστών, Test καιρού κ.α.
- Το περιβάλλον διαχείρισης επιτρέπει τον πολύ εύκολο χειρισμό των δεδομένων της εφαρμογής αφού είναι εύχρηστο, απλό και κατανοητό
- Όπως και το Frontend, η υλοποίηση του έγινε χρησιμοποιώντας εργαλέα όπως Laravel, Vue.js, mySQL κ.τ.λ.
- Η πρόσβαση στο Admin Panel ελέγχεται μέσω του middleware “admin” το οποίο ουσιαστικά ελέγχει αν **Auth::user() -> isAdmin**

Τέλος Β' Μέρους

Ερωτήσεις;