

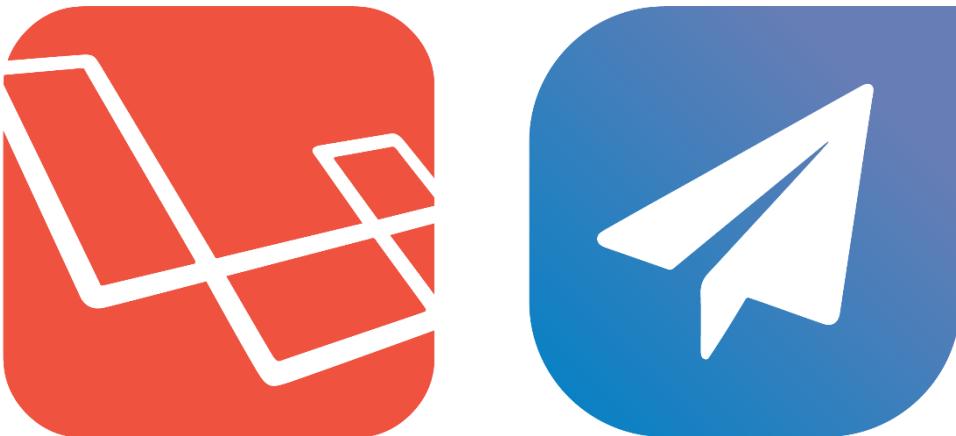


Τμήμα Μηχανικών  
Πληροφορικής ΑΤΕΙΘ

ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ  
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ  
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εισαγωγή στο PHP Framework “Laravel” και  
ανάπτυξη Web Application παροχής  
πληροφοριών για τα αεροδρόμια του ελληνικού  
FIR



Καρπούζης Κυριάκος – Α.Μ. 123886

Επιβλέπων καθηγητής: Σιδηρόπουλος Α.

Θεσσαλονίκη 2018

## Πρόλογος

Μέρα με την μέρα η επιστήμη της πληροφορικής εξελίσσεται με ραγδαίους ρυθμούς. Η ανακάλυψη νέων τεχνολογιών, η βελτίωση των ήδη υπαρχόντων αλλά και η τοποθέτηση τρεχόντων στο «χρονοντούλαπο» της ιστορίας έχουν ως φυσικό επακόλουθο την αύξηση των απαιτήσεων από τους επιστήμονες της πληροφορικής. Καθένας που έχει επιλέξει να αφιερώσει την επαγγελματική του σταδιοδρομία σε αυτό τον τομέα καλείται καθημερινώς να βρίσκει την βέλτιστη λύση για προβλήματα που παρουσιάζονται διαρκώς, πολλές φορές μέσα σε στενά χρονικά περιθώρια. Για την περάτωση όλων αυτών των προβλημάτων οι μηχανικοί πληροφορικής έχουν πολλές φορές στα χέρια τους εργαλεία που τους βοηθούν να είναι αποδοτικότεροι και δημιουργικότεροι σε μικρό χρονικό διάστημα. Τα εργαλεία αυτά μπορούν να είναι τόσο «μεγάλα» (π.χ. Frameworks, βιβλιοθήκες) όσο και «μικρά» (π.χ. scripts), αλλά όλα τους ικανοποιούν έναν από τους μεγαλύτερους περιορισμούς του προγραμματισμού: την επαναχρησιμοποίηση.

## **Περίληψη**

Η πτυχιακή εργασία αποτελείται από δύο μέρη: Εισαγωγή στο PHP Framework “Laravel” και ανάπτυξη μιας διαδικτυακής εφαρμογής για την παροχή πληροφοριών σχετικά με τα ελληνικά αεροδρόμια.

Στο πρώτο μέρος γίνεται μια γενική επισκόπηση των χαρακτηριστικών και των δυνατοτήτων που παρέχει η Laravel καθώς και μια σύντομη εμβάθυνση σε κάποια ειδικότερα χαρακτηριστικά. Η εισαγωγή είναι δημιουργημένη έτσι ώστε να είναι εύκολα κατανοητή τόσο από έμπειρους προγραμματιστές όσο και από κάποιον που πιθανώς δεν έχει χρησιμοποιήσει κάποιο framework στο παρελθόν ή έχει χρησιμοποιήσει κάτι που υλοποιείται μέσω κάποιας άλλης λογικής ή αρχιτεκτονικής.

Στο δεύτερο και πρακτικό μέρος γίνεται η παρουσίαση της διαδικτυακής εφαρμογής LGthree η οποία εν μέρει έχει υλοποιηθεί με την χρήση τεχνολογιών και δυνατοτήτων που προσφέρει η Laravel. Η εφαρμογή αποτελεί ένα παράδειγμα για το πως όλα αυτά που παρουσιάστηκαν σε θεωρητικό επίπεδο στο πρώτο μέρος μπορούν να εφαρμοστούν και στην πράξη βοηθώντας τον προγραμματιστή να επιλύσει τα ζητήματα με τα οποία έρχεται αντιμέτωπος με όλα τα πλεονεκτήματα και τα μειονεκτήματα που μπορεί να έχει μία τέτοια υλοποίηση.

## **Abstract**

The bachelor thesis consists of two parts: An introduction to the PHP Framework “Laravel” as well as the development of a web application that provides information regarding the airports located in the Greek FIR.

The first part is a general overview of the characteristics and capabilities of Laravel as well as a brief deepening in some more advanced features. The goal of the introduction is for it to be understood by developers with a variety of programming background, such as developers that have never used a framework before or developers that have used a framework that implements different logic of mechanisms.

In the second and practical part, a presentation of the web application LGthree takes place. The application has been partly developed using technologies and features that provided by Laravel and it serves as an example of how all the previously presented points, attributes and characteristics, can be implemented in the real world in order to help the developer solve the problems he might face with all the pros and cons of such a choice.

## Περιεχόμενα

Πρόλογος .....	2
Περίληψη .....	3
Abstract.....	4
Περιεχόμενα.....	5
Ευρετήριο Σχημάτων.....	9
Ευρετήριο Πινάκων .....	10
1. Εισαγωγή .....	11
1.1. Τι είναι η Laravel; .....	11
1.2. Γιατί Laravel; .....	11
1.3. Χαρακτηριστικά .....	12
1.4 Ιστορικό Εκδόσεων.....	13
2. Εγκατάσταση .....	14
2.1. Development Environments.....	14
2.2 Composer .....	14
2.3 Εγκατάσταση .....	15
3. Δομή Καταλόγων .....	16
3.1 Root Directory (Κατάλογος Ρίζα) .....	16
3.2. App Directory (Κατάλογος App) .....	17
3.3 Αρχείο .env .....	19
4. Αρχιτεκτονική .....	21
4.1 Αρχιτεκτονική Model-View-Controller (MVC) .....	21
4.2 Request Lifecycle .....	21
4.3 Service Container.....	22
4.4 Service Providers .....	23
4.5 Facades.....	23
5. Βασικά Στοιχεία .....	24
5.1 Artisan.....	24
5.2 Routing (Δρομολόγηση) .....	24
5.3 Middleware.....	24

5.4 Προστασία CSRF .....	25
5.5 Models .....	25
5.6 Views.....	25
5.7 Controllers.....	26
5.8 URL Generation.....	27
5.9 Sessions.....	27
5.10 Logging.....	27
6. Frontend.....	28
6.1 Blade Templates .....	28
6.2 Τοπικοποίηση (Localization).....	29
6.3 Compiling Assets (Laravel Mix).....	30
7. Βάσεις Δεδομένων .....	31
7.1 Εισαγωγή.....	31
7.2 Migrations.....	31
7.3 Seeding.....	32
7.4 Raw Queries.....	33
7.5 Query Builder .....	34
7.6 Eloquent ORM.....	35
8. Eloquent ORM.....	36
8.1 Εισαγωγή.....	36
8.2 CRUD Operations – Read .....	36
8.3 CRUD Operations – Create.....	37
8.4 CRUD Operations - Update.....	38
8.5 CRUD Operations – Delete.....	39
8.6 Relationships (Σχέσεις).....	39
8.7 Eager Loading.....	43
8.8 Accessors & Mutators.....	43
9. Εμβάθυνση (Digging Deeper) .....	45
9.1 Αυθεντικοποίηση .....	45
9.2 Caching.....	45

9.3 Events .....	45
9.4 Mail .....	45
9.5 Notifications.....	46
9.6 Queues.....	46
9.7 Scheduling.....	46
10. Παράδειγμα: Υλοποίηση Εφαρμογής «Λίστα Καθηκόντων».....	48
10.1 Εισαγωγή .....	48
10.2 Εγκατάσταση .....	48
10.3 Σύνδεση με την Βάση Δεδομένων και υλοποίηση συστήματος αυθεντικοποίησης .....	48
10.4 Δημιουργία Migration και Model.....	49
10.5 Routing.....	51
10.6 Views.....	52
10.7 Δημιουργία ενός Task.....	55
10.8 Μετάβαση στον Controller και προβολή των Tasks .....	56
10.9 Διαγραφή των Tasks.....	59
10.10 Επίλογος.....	60
11. LGthree – Εισαγωγή .....	62
11.1 Τι είναι το LGthree;.....	62
11.2 Επισκόπηση Χαρακτηριστικών .....	62
11.3 Τεχνολογίες.....	62
12. Χρήσιμες Ορολογίες.....	64
12.1 METAR.....	64
12.2 TAF.....	65
12.3 NOTAM.....	66
12.4 Ενεργός Διάδρομος Προσγείωσης-Απογείωσης .....	66
12.5 SID – STAR .....	70
12.6 Charts (Χάρτες) .....	70
12.7 QNH.....	71
12.8 Transition Altitude – Transition Layer - Transition Level .....	72

13. Χαρακτηριστικά και Τεχνολογίες.....	74
13.1 Αυθεντικοποίηση .....	74
13.2 Παροχή Πληροφοριών .....	74
13.3 Προγραμματισμένες Ενημερώσεις .....	76
13.4 Επιλογή Ενεργού Διαδρόμου .....	77
13.5 Ενημερώσεις Πραγματικού Χρόνου .....	78
13.6 Περιβάλλον Διαχείρισης.....	82
Συμπεράσματα .....	84
Βιβλιογραφία.....	85

## **Ευρετήριο Σχημάτων**

Σχήμα 1 "Δημοφιλία Frameworks στις αναζητήσεις Google" .....	12
Σχήμα 2 "Η δομή του Root Directory" .....	16
Σχήμα 3 "Η δομή του App Directory" .....	17
Σχήμα 4 "Παράδειγμα αρχείου .env" .....	20
Σχήμα 5 "Απλουστευμένο παράδειγμα μοντέλου MVC" .....	21
Σχήμα 6 "Σχηματική απεικόνιση του Request Lifecycle" .....	22
Σχήμα 7 "Παραδείγματα δήλωσης routes" .....	24
Σχήμα 8 "Παράδειγμα Laravel Blade" .....	28
Σχήμα 9 "Παράδειγμα υλοποίησης τοπικοποίησης με χρήση Short Keys" .....	29
Σχήμα 10 "Παράδειγμα υλοποίησης τοπικοποίησης με χρήση Translation Strings As Keys" .....	30
Σχήμα 11 "Παράδειγμα κλάσης Migration" .....	32
Σχήμα 12 "Παράδειγμα κλάσης Seeder" .....	33
Σχήμα 13 "Παράδειγμα χρήσης Raw Queries" .....	34
Σχήμα 14 "Παράδειγμα χρήσης Query Builder" .....	34
Σχήμα 15 "Παράδειγμα CRUD - Read" .....	37
Σχήμα 16 "Παράδειγμα CRUD - Create" .....	38
Σχήμα 17 "Παράδειγμα CRUD - Update" .....	38
Σχήμα 18 "Παράδειγμα CRUD - Delete" .....	39
Σχήμα 19 "Η σχέση One to One" .....	40
Σχήμα 20 "Η σχέση One to Many" .....	40
Σχήμα 21 "Η Σχέση Many to Many" .....	41
Σχήμα 22 "Η Σχέση Has Many Through" .....	41
Σχήμα 23 "Η σχέση Polymorphic" .....	42
Σχήμα 24 "Η σχέση Many To Many Polymorphic" .....	43
Σχήμα 25 "Η χρήση του Eager Loading" .....	43
Σχήμα 26 "Παράδειγμα accessor & mutator" .....	44
Σχήμα 27 "Παράδειγμα πρόγνωσης TAF" .....	66
Σχήμα 28 "Σχηματική απεικόνιση του wind component" .....	68
Σχήμα 29 "Σχηματική απεικόνιση διαδρόμων LGTS και ανέμου 140 μοιρών" .....	69
Σχήμα 30 "Παραδείγματα χαρτών για το αεροδρόμιο LGTS" .....	71
Σχήμα 31 "Σχηματική απεικόνιση transition altitude, layer, level" .....	72
Σχήμα 32 "Η σελίδα σύνδεσης στην εφαρμογή" .....	74
Σχήμα 33 "Η αρχική σελίδα του LGthree" .....	75
Σχήμα 34 "Στιγμιότυπο από την σελίδα για το αεροδρόμιο LGTS" .....	76
Σχήμα 35 "Απλουστευμένη σχηματική απεικόνιση της εντολής update:metars" .....	77
Σχήμα 36 "Σχηματική απεικόνιση του trait ActiveRunwayLogic" .....	77

Σχήμα 37 "Στιγμιότυπο προβολής ενεργού διαδρόμου" .....	78
Σχήμα 38 "Ο κώδικας δημιουργίας του event WeatherChanged".....	79
Σχήμα 39 "Η εισαγωγή και ρύθμιση του Laravel Echo" .....	79
Σχήμα 40 "Παράδειγμα Vue Component" .....	80
Σχήμα 41 "Η κλήση του component Metar.vue" .....	81
Σχήμα 42 "Το Vue Component Metar.vue" .....	82
Σχήμα 43 "Απόσπασμα από το περιβάλλον διαχείρισης" .....	83

## **Ευρετήριο Πινάκων**

Πίνακας 1 "Εκδόσεις της Laravel και ημερομηνίες" .....	13
Πίνακας 2 "Διαθέσιμες ενέργειες Resource Controller" .....	26
Πίνακας 3 "Όνομασίες κάλυψης ουρανού" .....	65
Πίνακας 4 "Διάδρομοι προσγείωσης-απογείωσης LGTS" .....	69

## **1. Εισαγωγή**

### **1.1. Τι είναι η Laravel;**

Η Laravel είναι ένα δωρεάν PHP Framework ανοιχτού κώδικα, το οποίο έχει δημιουργηθεί από τον Taylor Otwell με σκοπό την δημιουργία διαδικτυακών εφαρμογών που ακολουθούν την αρχιτεκτονική Model-View-Controller (MVC). Τα σημαντικότερα χαρακτηριστικά της Laravel είναι η εκφραστική και κομψή σύνταξη, η παροχή προ-υλοποιημένων χαρακτηριστικών (π.χ. αυθεντικοποίηση), οι ποικίλοι τρόποι προσπέλασης βάσεων δεδομένων κ.α.

Όπως χαρακτηριστικά αναφέρεται στην επίσημη ιστοσελίδα, η Laravel προσπαθεί να διευκολύνει την ανάπτυξη εφαρμογών με την υλοποίηση χαρακτηριστικών που είναι κοινά στις περισσότερες εφαρμογές διαδικτύου όπως authentication, routing, sessions, caching κ.τ.λ. Στόχος του framework είναι να κάνει ευχάριστη ολόκληρη την εμπειρία της ανάπτυξης χωρίς όμως να στερεί την εφαρμογή από δυνατότητες.

Έτσι λοιπόν, η Laravel συνδυάζει κάποια από τα καλύτερα χαρακτηριστικά δημοφιλών frameworks άλλων γλωσσών όπως Ruby on Rails, ASP.NET MVC και Sinatra.

Ο πηγαίος κώδικας φιλοξενείται στο GitHub και διέπεται από την áδεια MIT, δίνοντας έτσι την δυνατότητα παρέμβασης και προσθήκης χαρακτηριστικών σε όποιον το επιθυμεί.

### **1.2. Γιατί Laravel;**

Οι λόγοι και τα πλεονεκτήματα για να επιλέξει κάποιος να χρησιμοποιήσει την Laravel για την δημιουργία της διαδικτυακής του εφαρμογής είναι πολλά. Ενδεικτικά αναφέρονται:

- 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)

Το παρακάτω στιγμιότυπο από τα Google Trends αναφέρεται στην δημοφιλία πέντε PHP frameworks από την 1<sup>η</sup> Μαΐου 2012 έως και 1<sup>η</sup> Μαΐου 2018. Είναι χαρακτηριστική η άνοδος των αναζητήσεων σχετικά με την Laravel έναντι των άλλων -εξαιρετικά δημοφιλών στην αγορά εργασίας- frameworks.



Σχήμα 1 "Δημοφιλία Frameworks στις αναζητήσεις Google"

### 1.3. Χαρακτηριστικά

Παρακάτω παρουσιάζονται κάποια από τα καλύτερα και πιο δημοφιλή χαρακτηριστικά της Laravel που την καθιστούν το νούμερο 1 σε προτίμηση framework.

- 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

Στα επόμενα κεφάλαια θα αναλύσουμε τα παραπάνω χαρακτηριστικά για να δούμε πως βιοθούν στην ανάπτυξη των εφαρμογών.

#### 1.4 Ιστορικό Εκδόσεων

Η Laravel δημιουργήθηκε από τον Taylor Otwell σαν μια εναλλακτική μιας εξελιγμένης μορφής του CodeIgniter που δεν παρείχε δυνατότητες όπως σύστημα αυθεντικοποίησης. Η πρώτη beta έκδοση κυκλοφόρησε στις 9 Ιουνίου 2011. Έναν μήνα αργότερα έγινε η κυκλοφορία της πρώτης έκδοσης της Laravel που ενώ παρείχε χαρακτηριστικά όπως built-in authentication system, models, localization κ.τ.λ., δεν χρησιμοποιούσε controllers, κάτι που την καθιστούσε ένα frameworks που δεν υλοποιούσε την αρχιτεκτονική MVC. Φυσικά σήμερα κάτι τέτοιο δεν ισχύει αλλά αποτελεί ένα άριστο παράδειγμα χρήσης της αρχιτεκτονικής αυτής. Με την πάροδο των χρόνων προστέθηκαν νέες δυνατότητες με αποτέλεσμα να φτάσει σήμερα να θεωρείται ως ένα εξαιρετικό framework όχι μόνο στην γλώσσα της php αλλά γενικώς στον τομέα του προγραμματισμού.

Πίνακας 1 "Εκδόσεις της Laravel και ημερομηνίες"

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

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

## 2. Εγκατάσταση

### 2.1. Development Environments

Για την δημιουργία και την ανάπτυξη Laravel Applications είναι απαραίτητη η ρύθμιση του περιβάλλοντος ανάπτυξης. Ο προγραμματιστής μπορεί να δημιουργήσει εκείνος το περιβάλλον που επιθυμεί ή να εξομαλύνει αυτή την διαδικασία επιλέγοντας μία από τις ήδη «έτοιμες» λύσεις που προσφέρει η κοινότητα της Laravel. Παρακάτω θα γίνει αναφορά σε τρία δημοφιλή dev environments.

**Homestead:** Το Laravel Homestead είναι ένα επίσημο, pre-packaged Vagrant box το οποίο δεν απαιτεί από τον προγραμματιστή να εγκαταστήσει οτιδήποτε (π.χ. PHP, Web Server). Το μεγαλύτερο πλεονέκτημα είναι πως σε περίπτωση λάθος ρύθμισης το «Vagrant Box» μπορεί να καταστραφεί και να δημιουργηθεί σε λίγα μόνο λεπτά. Το Homestead λειτουργεί σε συστήματα Windows, Mac και Linux και περιέχει Nginx web server, PHP 7.2, PHP 7.1, PHP 5.6, MySQL, PostgreSQL, Redis, Memcached, Node και ότι άλλο είναι απαραίτητο για την δημιουργία μιας Laravel εφαρμογής.

**Valet:** Το Valet είναι ένα περιβάλλον ανάπτυξης το οποίο είναι συμβατό με συστήματα Mac. Λειτουργεί μέσω του Homebrew και η εγκατάσταση-ρύθμιση του είναι κυριολεκτικά μικρότερη από δύο λεπτά, επιτρέποντας τον προγραμματιστή να αφοσιωθεί στην παραγωγή κώδικα.

**Docker:** Το Docker είναι μια πλατφόρμα ανοιχτού λογισμικού που υλοποιεί εικονοποίηση και χρησιμοποιεί Containers, με τα οποία αποφεύγεται η χρήση πόρων που θα απαιτούσε μια εικονική μηχανή. Είναι η λιγότερο δημοφιλής επιλογή ως περιβάλλον ανάπτυξης σε αντίθεση όμως με την χρήση του σε περιβάλλοντα παραγωγής που ο ρυθμός χρήσης του αυξάνεται κατακόρυφα.

### 2.2 Composer

Το Composer είναι ένα package manager επιπέδου εφαρμογής για την PHP, το οποίο διαχειρίζεται τις εξαρτήσεις (dependencies) και τις βιβλιοθήκες των εφαρμογών που το χρησιμοποιούν. Είναι βασισμένο στην idéa του npm (package manager για την node.js). Λειτουργεί μέσω Command Lines και αντλεί τα διαθέσιμα πακέτα από το Packagist (<https://packagist.org>), χωρίς όμως να περιορίζεται στην χρήση μόνο αυτών των πακέτων. Λειτουργεί μέσω Command Line και προσφέρει δυνατότητες autoload για βιβλιοθήκες που περιέχουν ανάλογη τεκμηρίωση κάνοντας έτσι την χρήση third-party κώδικα ακόμα πιο εύκολη.

## 2.3 Εγκατάσταση

Συγκεκριμένα για την Laravel, παρατηρούμε μια αναδρομή. Η ίδια η Laravel αποτελεί ένα package που εγκαθίσταται μέσω του composer, αλλά ταυτόχρονα χρησιμοποιεί άλλα packages που αποτελούν και αυτά μέρος του composer. Αυτό είναι ένα τεράστιο πλεονέκτημα καθώς επιτρέπει την επαναχρησιμοποίηση κώδικα που έχει υλοποιηθεί από κάποιον προγραμματιστή χωρίς να χρειάζεται να «ανακαλυφθεί ξανά ο τροχός».

Η εγκατάσταση της Laravel σε ένα μηχάνημα γίνεται με δύο τρόπους:

### Μέσω Laravel Installer:

Πρώτα χρειάζεται να γίνει εγκατάσταση του Laravel Installer μέσω του Composer:

```
composer global require "laravel/installer"
```

Υστερα το composer θα πρέπει να τοποθετηθεί στην μεταβλητή \$PATH του συστήματος ούτως ώστε να είναι εμφανής στο σύστημα. Αφού γίνει η ανάλογη τοποθέτηση η δημιουργία μιας εφαρμογής Laravel γίνεται με μία απλή εντολή:

```
laravel new blog
```

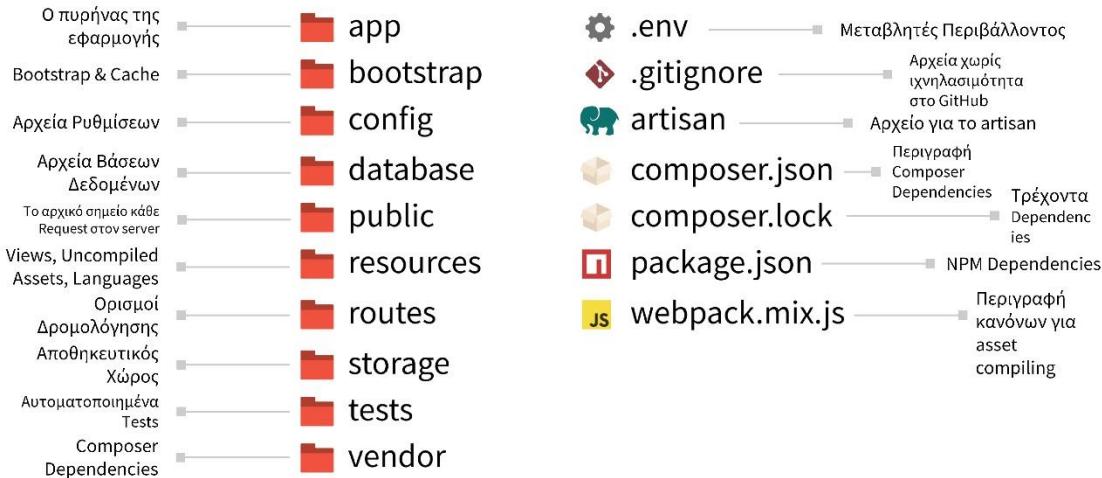
### Μέσω του Composer Create-Project

Εναλλακτικά η Laravel μπορεί να εγκατασταθεί μέσω της εντολής *create-project*:

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

### 3. Δομή Καταλόγων

#### 3.1 Root Directory (Κατάλογος Ρίζα)



Σχήμα 2 "Η δομή του Root Directory"

#### App Directory:

Περιέχει τον πυρήνα της εφαρμογής. Εδώ υπάρχουν οι περισσότερες κλάσεις της εφαρμογής. Θα γίνει εκτενέστερη αναφορά στην συνέχεια.

#### Bootstrap Directory:

Περιέχει το αρχείο app.php το οποίο υλοποιεί το bootstrap του framework. Επίσης σε αυτόν τον κατάλογο υπάρχει ο κατάλογος cache ο οποίος περιέχει αρχεία δημιουργημένα από το framework για την βελτιστοποίηση της απόδοσης της εφαρμογής.

#### Config Directory:

Όπως γίνεται κατανοητό και από το όνομα ο κατάλογος αυτός περιέχει όλα τα αρχεία τα οποία αποτελούν τις ρυθμίσεις της εφαρμογής και των πακέτων της.

#### Database Directory:

Εδώ περιέχονται όλα τα migrations, model factories και seeds της βάσης δεδομένων (βλ. κεφ. 7).

#### Public Directory:

Ο κατάλογος public περιέχει το αρχείο index.php, το οποίο είναι το σημείο εισόδου για όλα τα requests που καταφθάνουν στην εφαρμογή καθώς επίσης και διαχειρίζεται το autoloading. Επιπλέον στον κατάλογο αυτό υπάρχουν όλα τα δημόσια στοιχεία της εφαρμογής όπως εικόνες, JavaScript και CSS.

### **Resources Directory:**

Στον φάκελο resources υπάρχουν όλα τα views της εφαρμογής καθώς και όλα τα raw, un-compiled assets όπως LESS, SASS, JavaScript κ.τ.λ., όπως επίσης και όλα τα αρχεία γλωσσών της εφαρμογής.

### **Routes Directory:**

Περιέχει όλους τους ορισμούς δρομολόγησης της εφαρμογής. Εξ' ορισμού οι διαδρομές περιγράφονται στα αρχεία web.php, api.php, console.php και channels.php. Καθ' ένα από αυτά υλοποιεί την δρομολόγηση με διαφορετικό τρόπο και για διαφορετικό σκοπό.

### **Storage Directory:**

Στον κατάλογο Storage περιέχονται τα compiled Blade Templates, file based sessions, file caches και άλλα αρχεία που παράγονται από το framework.

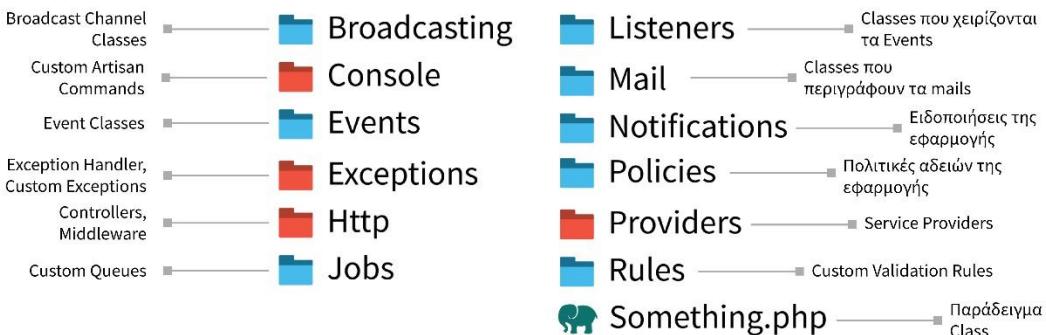
### **Tests Directory:**

Όλα τα αυτοματοποιημένα test υπάρχουν μέσα στον ομώνυμο κατάλογο.

### **Vendor Directory:**

Ο φάκελος vendor περιέχει όλα τα composer dependencies.

## **3.2. App Directory (Κατάλογος App)**



Σχήμα 3 "Η δομή του App Directory"

Η πλειονότητα της εφαρμογής υπάρχει στον κατάλογο app. Εξ' ορισμού, αυτός ο κατάλογος είναι namespaced στο App και γίνεται autoload μέσω του Composer χρησιμοποιώντας το PSR-4 autoloading standard.

Οι 3 βασικοί υποκατάλογοι που περιέχονται στον κατάλογο app είναι οι Console, Http και Providers. Παρ' όλα αυτά μέσω του artisan command “make” μπορούν να δημιουργηθούν νέοι κατάλογοι ανάλογα με την φύση της εντολής και την λογική που

υλοποιεί η αντίστοιχη κλάση. Παρακάτω θα δούμε αναλυτικότερα την περιγραφή όλων των υποκαταλόγων.

### **Broadcasting Directory:**

Ο κατάλογος Broadcasting περιέχει όλα τις broadcast channel classes της εφαρμογής. Αυτές οι κλάσεις δημιουργούνται χρησιμοποιώντας την εντολή “make:channel”. Αυτός ο κατάλογος δεν υπάρχει εξ’ ορισμού, αλλά δημιουργείται με την δημιουργία του πρώτου channel.

### **Console Directory:**

Ο κατάλογος Console περιέχει όλες τις custom artisan εντολές. Αυτές οι εντολές δημιουργούνται μέσω της εντολής “make:command”. Επίσης αυτός ο κατάλογος περιέχει το console kernel, στο οποίο καταχωρούνται οι custom artisan εντολές καθώς και όλα τα scheduled tasks.

### **Events Directory:**

Αυτός ο κατάλογος δεν υπάρχει εξ’ ορισμού αλλά δημιουργείται μέσω στον εντολών “event:generate” και “make:event”. Εδώ περιέχονται όλες οι event classes, οι οποίες μπορούν να χρησιμοποιηθούν για την ειδοποίηση άλλων σημείων της εφαρμογής μετά από μια ενέργεια.

### **Exceptions Directory:**

Εδώ περιέχεται ο exception handler της εφαρμογής καθώς και είναι το κατάλληλο μέρος για την τοποθέτηση όλων των exceptions που συμβαίνουν στην εφαρμογή.

### **Http Directory:**

Ίσως ο σημαντικότερος κατάλογος καθώς περιέχει όλους τους controllers, middleware και form requests. Σχεδόν όλη η λογική για τον χειρισμό των requests που γίνονται στην εφαρμογή λαμβάνει χώρα σε αυτόν τον κατάλογο.

### **Jobs Directory:**

Ακόμη ένας κατάλογος που δεν υπάρχει εξ’ ορισμού αλλά δημιουργείται μέσω της εντολής “make:job”. Εδώ περιέχονται όλα τα queueable jobs της εφαρμογής, δηλαδή κλάσεις η οποίες μπαίνουν στην «ουρά» και εκτελούνται ταυτόχρονα με την εφαρμογή.

### **Listeners Directory:**

Ο κατάλογος listeners δεν δημιουργείται με την αρχικοποίηση της εφαρμογής αλλά μόνο μετά την εκτέλεση της εντολής “event:generate” ή της εντολής “make:listener”.

Στον κατάλογο αυτό υπάρχουν οι κλάσεις οι οποίες χειρίζονται τα events. Κάθε listener λαμβάνει ένα instance ενός event και υλοποιεί την λογική ως απάντηση στο event.

### **Mail Directory:**

Με την εκτέλεση της εντολής “make:mail” δημιουργείται και ο κατάλογος mail. Εδώ περιέχονται όλες οι κλάσεις οι οποίες αντιπροσωπεύουν όλα τα emails που στέλνονται από την εφαρμογή. Τα αντικείμενα Mail επιτρέπουν την ενθυλάκωση ολόκληρης της λογικής της δημιουργίας ενός email, σε μια απλή κλάση η οποία μπορεί να σταλεί μέσω της μεθόδου “Mail::send”.

### **Notifications Directory:**

Εξ’ ορισμού ο κατάλογος αυτός δεν υπάρχει στην εφαρμογή αλλά δημιουργείται μετά την εκτέλεση της εντολής “make:notification”. Περιέχει όλες τις ειδοποιήσεις που στέλνονται από την εφαρμογή, όπως για παράδειγμα μια απλή ειδοποίηση μετά από ένα γεγονός που συνέβη στην εφαρμογή. Η Laravel περιέχει έτοιμες υλοποιήσεις για μια πληθώρα ειδοποιήσεων (Email, Slack, SMS, Database)

### **Policies Directory:**

Ο κατάλογος policies δεν υπάρχει εξ’ ορισμού. Δημιουργείται με την εντολή “make:policy”. Περιέχει όλες τις policy classes, οι οποίες καθορίζουν εάν ένας χρήστης μπορεί να εκτελέσει μία συγκεκριμένη ενέργεια.

### **Providers Directory:**

Σε αυτόν τον κατάλογο περιέχονται όλοι οι service providers της εφαρμογής. Οι service providers είναι υπεύθυνοι για το bootstrap της εφαρμογής κατά το οποίο συνδέουν τα services στο service container, καταχωρούν γεγονότα ή εκτελούν καθήκοντα για την προετοιμασία της εφαρμογής για επερχόμενα αιτήματα.

### **Rules Directory:**

Με την εντολή “make:rule” δημιουργείται και ο κατάλογος rules. Περιέχει όλες τις custom validation rules κλάσεις για την εφαρμογή. Τα rules χρησιμοποιούνται για την ενθυλάκωση μιας περίπλοκης λογικής επικύρωσης σε ένα απλό αντικείμενο.

## **3.3 Αρχείο .env**

Τις περισσότερες φορές κάθε περιβάλλον στο οποίο βρίσκεται η εφαρμογή απαιτεί διαφορετικές ρυθμίσεις. Για παράδειγμα, μπορεί η εφαρμογή να χρησιμοποιεί έναν διαφορετικό cache driver στο τοπικό περιβάλλον απ’ ότι στον production server.

Για την υλοποίηση αυτών των ρυθμίσεων η Laravel χρησιμοποιεί την βιβλιοθήκη DotEnv η οποία έχει αναπτυχθεί από τον Vance Lucas. Σε ένα fresh Laravel installation, ο κατάλογος ρίζα περιέχει ένα αρχείο .env.example. Σε περίπτωση που η Laravel εγκατασταθεί μέσω του composer, αυτό το αρχείο αυτόματα μετονομάζεται σε .env και δεν απαιτεί την χειροκίνητη μετονομασία του.

Το αρχείο .env δεν πρέπει να καταχωρείται στο source control της εφαρμογής, αφού κάθε developer/server απαιτεί και διαφορετικές ρυθμίσεις για το περιβάλλον του. Επί προσθέτως αυτό θα αποτελούσε και ένα ρίσκο ασφαλείας μιας και ένας εισβολέας θα μπορούσε να αποκτήσει πρόσβαση στο source control και συνεπώς σε ευαίσθητα διαπιστευτήρια της εφαρμογής.

Σε περίπτωση που η εφαρμογή αναπτύσσεται μέσω μιας ομάδας ατόμων, συστήνεται η παροχή ενός αρχείου .env.example, το οποίο λειτουργεί ως ένα προσχέδιο του αρχείου .env που ο κάθε προγραμματιστής πρέπει να ρυθμίσει για να καλύπτει τις απαιτήσεις του περιβάλλοντος που εργάζεται.

```
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

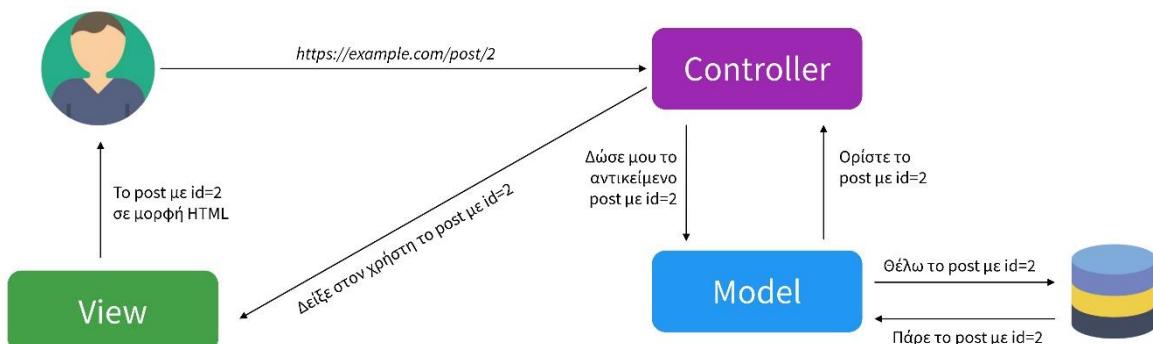
MY_CUSTOM_VARIABLE=abc
```

Σχήμα 4 "Παράδειγμα αρχείου .env"

## 4. Αρχιτεκτονική

### 4.1 Αρχιτεκτονική Model-View-Controller (MVC)

Το MVC είναι ένα μοντέλο αρχιτεκτονικής λογισμικού το οποίο χρησιμοποιείται εκτενώς τα τελευταία χρόνια στα περιβάλλοντα αλληλεπίδρασής χρήστη. Η λογική του MVC είναι πως η εφαρμογή διαιρείται σε τρία μέρη τα οποία συνδέονται μεταξύ τους, διαχωρίζοντας έτσι την παρουσίαση της πληροφορίας στον χρήστη από την μορφή που υπάρχει αποθηκευμένη στο σύστημα. Το Model είναι αυτό που διαχειρίζεται την ανάκτηση και την αποθήκευση των δεδομένων στο σύστημα. Το View είναι υπεύθυνο για την παρουσίαση της πληροφορίας στον χρήστη (π.χ. μέσω ενός GUI). Τέλος ο Controller είναι εκείνος που συνδέει το Model με το View και ουσιαστικά περιέχει όλη την λειτουργική λογική της εφαρμογής.



Σχήμα 5 "Απλουστευμένο παράδειγμα μοντέλου MVC"

### 4.2 Request Lifecycle

Το σημείο εισόδου όλων των αιτημάτων σε μια εφαρμογή Laravel είναι το αρχείο `public/index.php`. Όλα τα αιτήματα μεταβιβάζονται σε αυτό το αρχείο από τον web server (Apache/Nginx). Το αρχείο `index.php` δεν περιέχει πολύ κώδικα, αντίθετα είναι ένα αρχικό σημείο για την φόρτωση του υπόλοιπου framework.

Το αρχείο `index.php` φορτώνει το autoloader definition που έχει δημιουργηθεί από το Composer και ύστερα λαμβάνει ένα instance του Laravel Application από το `bootstrap/app.php`. Η πρώτη ενέργεια που γίνεται από την Laravel είναι η δημιουργία ενός Instance της εφαρμογής/Service Container.

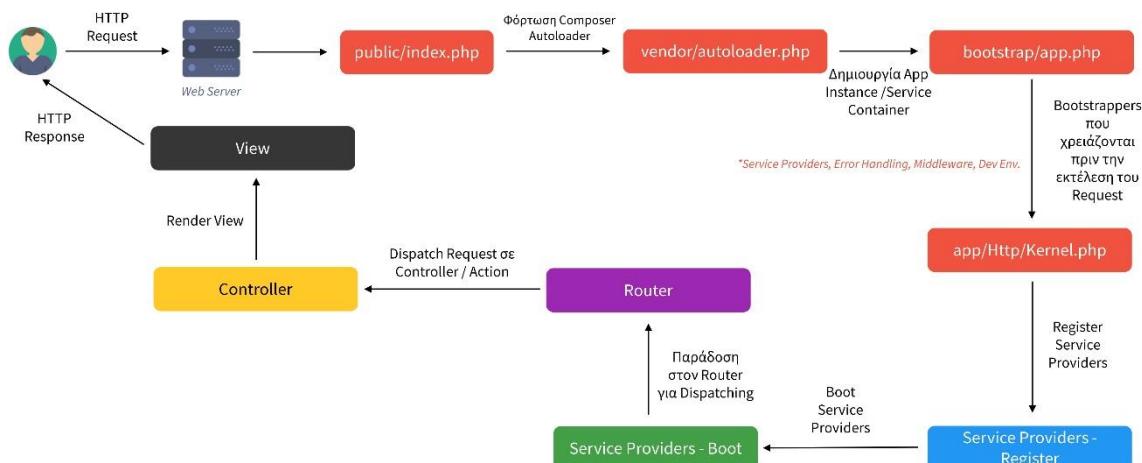
Στην συνέχεια το αίτημα στέλνεται είτε στο HTTP kernel ή στο Console kernel, ανάλογα με τον τύπο του αιτήματος. Αυτά τα δύο kernels λειτουργούν ως το κεντρικό μέρος από το οποίο διέρχονται όλα τα αιτήματα. Για την ώρα θα πάρουμε ως παράδειγμα το HTTP kernel το οποίο βρίσκεται στο `app/Http/Kernel.php`

Το HTTP kernel επεκτείνει την κλάση Illuminate\Foundation\Http\Kernel, η οποία ορίζει έναν πίνακα από bootstrappers οι οποίοι ενεργούν πριν την εκτέλεση ενός αιτήματος. Αυτοί οι bootstrappers ρυθμίζουν την διαχείριση των σφαλμάτων, εντοπίζουν το περιβάλλον της εφαρμογής και γενικώς εκτελούν όλα τα καθήκοντα που απαιτούνται πριν την εκτέλεση ενός request.

Επίσης το HTTP kernel ορίζει μία λίστα από κάποια HTTP middleware από τα οποία πρέπει να περάσουν όλα τα αιτήματα πριν τον χειρισμό τους από την εφαρμογή.

Μια από σημαντικότερες bootstrapping ενέργειες που φορτώνονται από το kernel είναι οι Service Providers. Οι Service Providers είναι υπεύθυνοι για το bootstrap όλων των διαφόρων συστατικών της εφαρμογής όπως η βάση δεδομένων, η ουρά, η επικύρωση κ.τ.λ. Στην επόμενη ενότητα θα ακολουθήσει μια πιο λεπτομερής περιγραφή των Service Providers.

Μετά το bootstrapping της εφαρμογής το request θα παραδοθεί στον router (δρομολογητή) για αποστολή. Ο router θα αποστέλει το request σε ένα route ή έναν controller καθώς και θα εκτελέσει κάποια συγκεκριμένα middleware της διαδρομής.



#### 4.3 Service Container

Το Service Container της Laravel είναι ένα ισχυρό εργαλείο για την διαχείριση των class dependencies και για την εκτέλεση του dependency injection. Ουσιαστικά με το dependency injection εννοείται πως όλα τα dependencies των κλάσεων εισέρχονται στην κλάση μέσω ενός δομητή ή σπανιότερα μέσω κάποιων "setter" μεθόδων.

Πρακτικά, το Service Container αποτελεί το instance της εφαρμογής.

## 4.4 Service Providers

Οι Service Providers είναι το κεντρικό μέρος στο οποίο συμβαίνει εξ' ολοκλήρου το απαραίτητο bootstrapping της εφαρμογής. Η γενική επεξήγηση του όρου bootstrapping είναι το register όλων των απαραίτητων πραγμάτων της εφαρμογής όπως Service Container Bindings, Event Listeners, Middleware κ.τ.λ. Οι περισσότεροι Service Providers αποτελούνται από τις μεθόδους “register” και “boot”.

Στην μέθοδο register γίνεται το bind όλων των απαραίτητων πραγμάτων στο service container. Δεν θα πρέπει ποτέ να γίνεται η καταχώρηση κλάσεων όπως event listeners, routes και γενικώς οποιασδήποτε λειτουργικότητας η οποία χρησιμοποιεί ένα service που παρέχεται από κάποιον service provider. Αυτό συμβαίνει διότι ο service provider στον οποίο γίνεται η κλήση μπορεί να μην έχει φορτώσει την στιγμή της κλήσης του.

Στην μέθοδο boot δίνεται το έναυσμα για την εκκίνηση των υπηρεσιών. Καλείται αφού έχουν γίνει register όλοι οι service providers το οποίο επιτρέπει στην εφαρμογή να έχει πρόσβαση σε όλα τα dependencies.

Ο προγραμματιστής είναι σε θέση να δημιουργεί δικούς του Service Providers μέσω της εντολής “`php artisan make:provider MyServiceProvider`”. Μετά την δημιουργία του provider η καταχώρηση του γίνεται στο αρχείο `config/app.php` και συγκεκριμένα στον πίνακα “providers” του αρχείου.

## 4.5 Facades

Τα Facades προσφέρουν μια «στατική» διασύνδεση με κλάσεις που είναι διαθέσιμες στο Service Container της εφαρμογής. Η Laravel έρχεται με αρκετά facades που δίνουν πρόσβαση στα περισσότερα από τα χαρακτηριστικά του framework.

Η χρήση των facades προσφέρει αρκετά πλεονεκτήματα. Παρέχουν ένα λακωνικό, αξιομνημόνευτο συντακτικό σε σχέση με τα μακριά ονόματα ορισμένων κλάσεων. Παρ' όλα αυτά συνιστάται προσοχή καθώς η χαρακτηριστική ευκολία τους μπορεί να οδηγήσει στην αλόγιστη χρήση τους μέσα σε μια μοναδική κλάση.

## 5. Βασικά Στοιχεία

### 5.1 Artisan

To artisan είναι ένα command-line interface που συμπεριλαμβάνεται στην Laravel. Παρέχει έναν αριθμό από χρήσιμες εντολές οι οποίες βοηθούν στην ανάπτυξη της εφαρμογής. Η Laravel έχει εξ' ορισμού κάποιες εντολές παρ' όλα αυτά ο προγραμματιστής είναι σε θέση να δημιουργήσει τις δικές του εντολές που θα ικανοποιούν τις απαιτήσεις του.

Μέσω του artisan γίνεται πάρα πολύ εύκολη η δημιουργία models, controllers, migrations και γενικώς όλα τα στοιχεία που χρησιμοποιούνται πολύ συχνά μέσα στην εφαρμογή. Οι εντολές που είναι διαθέσιμες φαίνονται με την εντολή “`php artisan list`”.

Επίσης, όλες οι εφαρμογές Laravel συμπεριλαμβάνουν το Tinker, ένα REPL βασισμένο στο PsySH package. Το Tinker καθιστά δυνατή την αλληλεπίδραση με ολόκληρη την εφαρμογή μέσω της γραμμής εντολών. Για την εκκίνηση του tinker, χρειάζεται η εκτέλεση της εντολής “`php artisan tinker`”.

### 5.2 Routing (Δρομολόγηση)

Στην Laravel υπάρχει εξ' ορισμού ένας μηχανισμός δρομολόγησης ο οποίος δρομολογεί όλα τα αιτήματα που καταφθάνουν στην εφαρμογή. Οι διαδρομές αυτές καθορίζονται στον κατάλογο routes. Τα συνηθέστερα αρχεία είναι τα `web.php` και `api.php` τα οποία υλοποιούν τα `web` και `api` middleware αντίστοιχα. Οι διαθέσιμες μέθοδοι με τις οποίες δηλώνονται τα routes είναι: `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');
```

Σχήμα 7 "Παραδείγματα δήλωσης routes"

### 5.3 Middleware

Το Middleware είναι ένας πρακτικός μηχανισμός της Laravel για το «φιλτράρισμα» των HTTP Requests που φτάνουν στην εφαρμογή. Για παράδειγμα, η Laravel συμπεριλαμβάνει εξ' ορισμού ένα middleware το οποίο εξακριβώνει αν ο χρήστης είναι αυθεντικοποιημένος. Σε περίπτωση που δεν είναι θα στείλει τον χρήστη στην σελίδα του Login, αν όμως είναι θα επιτρέψει στο request να προχωρήσει παρακάτω στην εφαρμογή.

Φυσικά είναι στην ευχέρεια του προγραμματιστή η συγγραφή middleware που εξυπηρετεί τους σκοπούς του. Όλα τα middleware βρίσκονται στον app\Http\Middleware. Επίσης τα middleware μπορούν να ενεργήσουν τόσο πριν όσο και μετά από ένα request.

#### 5.4 Προστασία CSRF

Στο framework υπάρχει ενσωματωμένος ένας μηχανισμός προστασίας από επιθέσεις CSRF<sup>1</sup>. Για κάθε ενεργό user session δημιουργείται αυτόματα από την Laravel ένα “CSRF Token”, το οποίο επαληθεύει πως ο αυθεντικοποιημένος χρήστης είναι εκείνος που δημιουργεί τα αιτήματα προς την εφαρμογή.

#### 5.5 Models

Τα models είναι οι οντότητες (κλάσεις-αντικείμενα) της εφαρμογής. Επειδή η Laravel υλοποιεί το μοτίβο “Active Record”, κάθε model αποτελεί ουσιαστικά την προγραμματιστική αναπαράσταση ενός πίνακα της βάσης δεδομένων. Για αυτό άλλωστε εξ' ορισμού ένας ευφυής μηχανισμός αντιστοιχίζει το όνομα του model στον ενικό αριθμό με το αντίστοιχο όνομα του πίνακα στον πληθυντικό χωρίς φυσικά αυτό να είναι δεσμευτικό, για παράδειγμα το model Post αντιστοιχίζεται αυτόματα με τον πίνακα posts.

Τα models επιτρέπουν την αλληλεπίδρασή του χρήστη με την βάση δεδομένων μέσω CRUD Operations. Έτσι η εφαρμογή και συνεπώς ο χρήστης μπορεί να δημιουργήσει, να διαβάσει, να τροποποιήσει και να διαγράψει εγγραφές στον πίνακα της βάσης δεδομένων.

Σε αυτό το σημείο θα ήταν καλό να επισημανθεί πως ενώ η οργάνωση του καταλόγου app είναι καλά δομημένη, παρατηρείται η απουσία ενός καταλόγου models που προφανώς θα περιείχε όλα τα models της εφαρμογής. Η απουσία αυτή είναι εκούσια. Σύμφωνα με τους δημιουργούς της Laravel, η έννοια της λέξης “models” είναι διφορούμενη. Έτσι επέλεξαν τα models να τοποθετούνται εξ' ορισμού στον κατάλογο app και να επιτρέπεται η μεταφορά σε άλλον κατάλογο αργότερα από τον προγραμματιστή της εφαρμογής.

#### 5.6 Views

Τα views περιέχουν κώδικα HTML και αναλαμβάνουν την τελική παρουσίαση της πληροφορίας στον χρήστη, καθώς επίσης διαχωρίζουν την λογική της εφαρμογής από την λογική της παρουσίασης. Βρίσκονται στον κατάλογο resources/views. Περιέχουν

---

<sup>1</sup> Η CSRF είναι επίθεση που αναγκάζει τον τελικό χρήστη να εκτελέσει ανεπιθύμητες ενέργειες σε μια εφαρμογή δικτύου στην οποία είναι πιστοποιημένος.

ελάχιστη έως καθόλου προγραμματιστική λογική εστιάζοντας περισσότερο στην μορφοποίηση και την παρουσίαση της πληροφορίας.

## 5.7 Controllers

Οι Controllers είναι αρχεία στα οποία περιγράφεται όλη η λογική χειρισμού των αιτημάτων που γίνονται στην εφαρμογή. Στην ουσία μπορούν να ομαδοποιήσουν συσχετιζόμενες μεθόδους χειρισμού των αιτημάτων σε μία κλάση. Βρίσκονται στον κατάλογο app\Http\Controllers.

Όπως αναφέρθηκε σε προηγούμενη ενότητα, οι controllers αποτελούν τον συνδετικό κρίκο ανάμεσα στα models και τα views. Όλοι οι controllers αποτελούν επέκταση του βασικού controller που περιέχει το framework (αρχείο app\Http\Controller.php).

Άξιος αναφοράς είναι ο τρόπος με τον οποίο εξ' ορισμού η Laravel διαχειρίζεται τυπικές CRUD operations. Μέσω του resource routing ανατίθενται όλες οι συνηθισμένες CRUD διαδρομές σε έναν μόνο controller και μόνο με μια γραμμή κώδικα. Έτσι μπορεί να χρησιμοποιηθεί ένας controller που ελέγχει όλες τις ενέργειες για την δημιουργία, ανάγνωση, τροποποίηση και διαγραφή μιας εγγραφής ενός model.

Η δημιουργία ενός Resource Controller γίνεται με την εντολή:

```
php artisan make:controller PostController --resource
```

Και η δήλωση της διαδρομής σε μία μόνο γραμμή:

```
Route::resource('posts', 'PostController');
```

Η ενέργειες που διαχειρίζεται ένας Resource Controller συνοψίζονται στον παρακάτω πίνακα:

Πίνακας 2 "Διαθέσιμες ενέργειες Resource Controller"

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

## 5.8 URL Generation

Η Laravel παρέχει διάφορους “helpers” οι οποίοι βοηθούν στην δημιουργία (generate) URLs για την εφαρμογή. Φυσικά, αυτοί αποδεικνύονται εξαιρετικά χρήσιμοι στο «χτίσιμο» συνδέσμων στα templates και στα API responses, ή όταν δημιουργούνται redirect responses για κάποιο άλλο σημείο της εφαρμογής.

Οι helpers αυτοί μπορούν μεταξύ άλλων να δημιουργήσουν URLs τα οποία υποδεικνύουν συγκεκριμένες διαδρομές, μεθόδους που βρίσκονται σε controllers, redirects ή μηνύματα σφάλματος (π.χ. 404, 503 κ.τ.λ.)

## 5.9 Sessions

Οι HTTP καθοδηγούμενες εφαρμογές είναι ανιθαγενείς, για αυτό τον λόγο χρησιμοποιούνται τα sessions, τα οποία παρέχουν έναν τρόπο για την αποθήκευση πληροφοριών σχετικά με τον χρήστη ενδιάμεσα σε πολλά αιτήματα. Η διαχείριση των sessions στην Laravel συμβαίνει με διάφορους τρόπους, από τους οποίους ο προγραμματιστής μπορεί να επιλέξει εκείνον που εξυπηρετεί περισσότερο τις ανάγκες και τις απαιτήσεις της εφαρμογής του.

Οι ρυθμίσεις σχετικά με την διαχείριση αποθηκεύονται στο αρχείο config/session.php. Ως προεπιλογή, η Laravel είναι ρυθμισμένη να χρησιμοποιεί τον οδηγό “file” για τα sessions. Οι οδηγοί που είναι έτοιμοι προς χρήση με κάθε εγκατάσταση του framework είναι:

- 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

## 5.10 Logging

Για την καλύτερη κατανόηση του τι συμβαίνει στην εφαρμογή, η Laravel παρέχει έναν στιβαρό μηχανισμό καταγραφής (logging) που επιτρέπει την καταγραφή μηνυμάτων σε αρχεία, σε ένα system error log ή ακόμα και σε ειδοποιήσεις της ομάδας ανάπτυξης στο Slack.

Το framework χρησιμοποιεί για αυτή την υλοποίηση την βιβλιοθήκη Monolog, η οποία παρέχει υποστήριξη για μια ποικιλία δυνατών χειριστών καταγραφής. Έτσι η ρύθμιση αυτών των χειριστών μέσα στο περιβάλλον της Laravel γίνεται «παιχνιδάκι».

## 6. Frontend

### 6.1 Blade Templates

Το Blade είναι ένας απλός αλλά ταυτόχρονα ισχυρός μηχανισμός templating που συμπεριλαμβάνει η Laravel. Σε αντίθεση με άλλους δημοφιλείς μηχανισμούς PHP templating, το Blade δεν περιορίζει την χρήση PHP κώδικα στα views. Στην πραγματικότητα όλα τα Blade views μεταγλωττίζονται σε απλό PHP κώδικα και τοποθετούνται στην cache μέχρι να τροποποιηθούν ξανά, βελτιώνοντας έτσι την απόδοση της εφαρμογής. Τα Blade views χρησιμοποιούν την κατάληξη “.blade.php” και τυπικά αποθηκεύονται στον κατάλογο “resources/views”.

Η χρήση του παραπάνω μηχανισμού προσφέρει πολλά οφέλη, με μεγαλύτερα την κληρονομικότητα των templates και την χρήση των sections. Μια τυπική διαμόρφωση Blade views είναι η ύπαρξη ενός master layout το οποίο μετά επεκτείνεται από μεμονωμένα views. Μέσα σε αυτό το master layout δηλώνονται κάποια sections τα οποία κληρονομούνται από κάθε view που κάνει extend αυτό το layout και μπορεί να είναι διαφορετικό από view σε view. Βλέπουμε έτσι να επιτυγχάνεται μια άριστη επαναχρησιμοποίηση κώδικα.

```
<!-- 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
```

Σχήμα 8 "Παράδειγμα Laravel Blade"

## 6.2 Τοπικοποίηση (Localization)

Τα χαρακτηριστικά τοπικοποίησης της Laravel παρέχουν έναν πρακτικό τρόπο για την ανάκτηση συμβολοσειρών (strings) σε διάφορες γλώσσες, επιτρέποντας με αυτόν τον τρόπο την εύκολη υποστήριξη πλήθους γλωσσών μέσα στην εφαρμογή. Τα strings των γλωσσών βρίσκονται αποθηκευμένα σε αρχεία στον κατάλογο “resources/lang”.

Υπάρχουν δύο τρόποι για την υλοποίηση της τοπικοποίησης: Τα “Short Keys” και τα “Translation Strings As Keys”. Η επιλογή της μεθόδου υλοποίησης έγκειται καθαρά στις απαιτήσεις της εφαρμογής και στις προσωπικές προτιμήσεις του προγραμματιστή.

Στην μέθοδο “Short Keys”, εντός του καταλόγου “resources/lang” υπάρχουν υποκατάλογοι για κάθε γλώσσα που υποστηρίζεται από την εφαρμογή. Για κάθε γλώσσα υπάρχει αρχείο(a) τα οποία περιέχουν τα μεταφρασμένα strings με ένα αναγνωριστικό κλειδί μικρού μεγέθους (εξ' ου και η ονομασία).

```
resources // resources/lang/en/messages.php
  lang
    en
      messages.php // resources/views/foo.blade.php
    lang
      el
        messages.php
```

Σχήμα 9 "Παράδειγμα υλοποίησης τοπικοποίησης με χρήση Short Keys"

Η δεύτερη μέθοδος, “Translation Strings As Keys” χρησιμοποιείται συνήθως σε εφαρμογές που έχουν μεγάλες μεταφραστικές απαιτήσεις, διότι ο ορισμός “short keys” μπορεί γρήγορα να εξελιχθεί σε τροχοπέδη κατά την αναφορά τους στα views. Τα αρχεία μεταφράσεων που χρησιμοποιούν αυτή την μέθοδο είναι της μορφής JSON και υπάρχουν στον κατάλογο “resources/lang”. Σε αντίθεση με την προηγούμενη μέθοδο κάθε γλώσσα δεν έχει τον προσωπικό της κατάλογο αλλά το προσωπικό της αρχείο JSON και είναι της μορφής isoCode.json. Για παράδειγμα, αν η εφαρμογή υποστηρίζει την μετάφραση σε ελληνικά θα έπρεπε να υπάρχει το αρχείο “resources/lang/el.json”.



The screenshot shows a file structure and a code editor. On the left, there's a tree view of files: 'resources' contains a 'lang' folder, which contains an 'el.json' file. On the right, the code editor shows PHP code. It includes a comment // resources/lang/el.json, a return statement with an array containing two items: 'I love programming!' and its Greek translation 'Αγαπώ τον προγραμματισμό!', followed by a closing bracket ]. Below this, another comment // resources/views/foo.blade.php is shown, followed by {{ \_\_('I love programming!') }} and @lang('I love programming!').

```
// resources/lang/el.json
return [
    'I love programming!'
        => 'Αγαπώ τον προγραμματισμό!'
];
// resources/views/foo.blade.php
{{ __('I love programming!') }}
@lang('I love programming!')
```

Σχήμα 10 "Παράδειγμα υλοποίησης τοπικοποίησης με χρήση Translation Strings As Keys"

### 6.3 Compiling Assets (Laravel Mix)

Το Laravel Mix παρέχει ένα εξαιρετικό API για τον ορισμό βημάτων του Webpack<sup>2</sup>. Δημιουργήθηκε από τον Jeffrey Way και ουσιαστικά χρησιμοποιεί ορισμένους pre-processors του Webpack. Στην πράξη, εξομαλύνει σε μεγάλο βαθμό την ρύθμιση και την χρήση του Webpack απαιτώντας μόνο την περιγραφή των βημάτων για την μεταγλώττιση αρχείων JavaScript ή CSS. Το Laravel Mix περιέχεται εξ' ορισμού στο "package.json" οποιασδήποτε εγκατάστασης Laravel και για την εγκατάσταση του απαιτείται απλά η εγκατάσταση των Node Dependencies (εντολή "npm install"). Μέχρι σήμερα υποστηρίζονται για μεταγλώττιση CSS οι pre-processors για SASS, LESS, Stylus, PostCSS, Plain CSS και για JavaScript οι Vue.js, React, Vanilla JS. Παρ' όλα αυτά με τις κατάλληλες παρεμβάσεις και ρυθμίσεις είναι δυνατόν η μεταγλώττιση οποιασδήποτε βιβλιοθήκης ή Framework.

---

<sup>2</sup> Static Module Bundler για μοντέρνες εφαρμογές JavaScript (<https://webpack.js.org/>)

## 7. Βάσεις Δεδομένων

### 7.1 Εισαγωγή

Η Laravel καθιστά την αλληλεπίδραση με βάσεις δεδομένων εξαιρετικά απλή υπόθεση, παρέχοντας μια πληθώρα επιλογών όπως raw SQL, Query Builder και το Eloquent ORM. Προς το παρόν οι τύποι βάσεων δεδομένων που υποστηρίζονται είναι τέσσερις: MySQL, PostgreSQL, SQLite, SQL Server. Φυσικά για την διασύνδεση με έναν τύπο βάσης δεδομένων που δεν υποστηρίζονται μπορούν να χρησιμοποιηθούν ορισμένα composer packages τα οποία έχουν αναπτυχθεί για ακριβώς αυτόν τον σκοπό (π.χ. για διασύνδεση με MongoDB).

Η ρυθμίσεις για την βάση δεδομένων βρίσκονται στο αρχείο “config/database.php”. Εκεί ορίζονται όλες οι διαθέσιμες βάσεις δεδομένων καθώς και ποια θα χρησιμοποιείται ως προεπιλογή. Αξίζει να σημειωθεί πως η Laravel επιτρέπει την διασύνδεση με πολλαπλές βάσεις δεδομένων.

### 7.2 Migrations

Τα migrations είναι κάτι σαν το version control για την βάση δεδομένων. Η χρήση τους επιτρέπει στην ομάδα ανάπτυξης την εύκολο τροποποίηση και διαμοιρασμό του σχήματος της βάσης δεδομένων (DB Schema) της εφαρμογής. Συνήθως τα migrations χρησιμοποιούν το Laravel schema builder για την κατασκευή του σχήματος της βάσης δεδομένων. Η συνηθέστερη πρακτική χρήσης, είναι η δημιουργία ενός migration για κάθε αλλαγή σε κάποιο πίνακα (δημιουργία, προσθήκη-αφαίρεση στήλης, διαγραφή).

Η δημιουργία ενός migration γίνεται μέσω της εντολής artisan “make:migration”. Το νέο αρχείο δημιουργείται στον κατάλογο “database/migrations”. Κάθε αρχείο migration έχει στο όνομα του ένα timestamp το οποίο επιτρέπει στην Laravel να καθορίσει την σειρά τους.

Κάθε migration κλάση αποτελείται από δύο μεθόδους: την “up” και την “down”. Η “up” χρησιμοποιείται για την προσθήκη πινάκων, στηλών ή ευρετηρίων στην βάση δεδομένων, ενώ η μέθοδος “down” αναστρέφει τις ενέργειες της “up”. Μέσα σε αυτές τις δύο μεθόδους χρησιμοποιείται όπως αναφέρθηκε το Laravel schema builder, το οποίο αποτελεί ένα ευανάγνωστο συντακτικό περιγραφής SQL Queries. Υποστηρίζει όλα τα πεδία που υποστηρίζει και ο εκάστοτε τύπος βάσης δεδομένων.

Μετά την συγγραφή κάποιου migration η εκτέλεσή του γίνεται με την εντολή “php artisan migrate”. Σε περίπτωση που επιθυμούμε την επαναφορά της βάσης δεδομένων στην μορφή που βρισκόταν στο προηγούμενο migration μπορούμε να εκτελέσουμε την εντολή “php artisan migrate:rollback”.

```

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');
    }
}

```

Σχήμα 11 "Παράδειγμα κλάσης Migration"

### 7.3 Seeding

To framework της Laravel περιλαμβάνει μια απλή μέθοδο για το seeding της βάσης δεδομένων. Με τον όρο seeding εννοούμε την δημιουργία εγγραφών σε πίνακες της βάσης δεδομένων. Αυτό μπορεί να εξυπηρετήσει τόσο στην δημιουργία εγγραφών για σκοπούς δοκιμής κατά την ανάπτυξη όσο και στην αρχικοποίηση των πινάκων στο production περιβάλλον. Όλες οι seed κλάσεις βρίσκονται στον κατάλογο “database/seeds”. Η ονομασία τους μπορεί να είναι οποιαδήποτε, αλλά συνηθίζεται να ακολουθούν ένα συγκεκριμένο μοτίβο της μορφής “NameTableSeeder”, π.χ. “UsersTableSeeder”. Οι seeders που πρέπει να καλεστούν από την Laravel δηλώνονται στην μέθοδο “call” της κλάσης “database/seeds/DatabaseSeeder.php”.

Η δημιουργία των seeders γίνεται μέσω της εντολής “php artisan make:seeder NameTableSeeder”. Μία κλάση seeder περιέχει μόνο μία μέθοδο, την μέθοδο “run”. Αυτή η μέθοδος καλείται μέσω της εντολής “db:seed”. Στην μέθοδο αυτή δηλώνονται οι εγγραφές που θα συμπληρώσουν τον πίνακα. Για τον ορισμό αυτών των εγγραφών υπάρχουν δύο τρόποι: Χειροκίνητα μέσω του Query Builder ή με την χρήση των Eloquent Model Factories.

Τα Model Factories είναι ένας πρακτικός τρόπος για την δημιουργία μεγάλου αριθμού εγγραφών. Ουσιαστικά ένα Model Factory περιγράφει ένα model της εφαρμογής.

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

Σχήμα 12 "Παράδειγμα κλάσης Seeder"

## 7.4 Raw Queries

Ο πρώτος τρόπος επικοινωνίας με την βάση δεδομένων είναι μέσω Raw Queries, δηλαδή με SQL Queries. Στο framework υλοποιείται μέσω του Façade “DB”, το οποίο υποστηρίζει τις μεθόδους select, update, insert, delete και statement.

Οι μέθοδοι select, update, insert και delete περιγράφονται από το όνομα τους. Οποιοδήποτε άλλο query που δεν καλύπτεται από αυτές τις μεθόδους μπορεί να εκτελεστεί μέσω της μεθόδου statement.

Επί προσθέτως, το Façade “DB” περιέχει την μέθοδο transaction, ή οποία όπως γίνεται κατανοητό από το όνομα της εκτελεί ένα σετ ενεργειών για μια συναλλαγή με την βάση δεδομένων.

Αυτός ο τρόπος επικοινωνίας αποτελεί συνήθως τελευταία επιλογή διότι αυξάνει την πολυπλοκότητα του κώδικα. Χρησιμοποιείται συνήθως όταν πρέπει να υλοποιηθεί μια λύση που δεν καλύπτεται από τους επόμενους τρόπους επικοινωνίας που περιγράφονται παρακάτω.

```

class UserController extends Controller
{
    /**
     * Show a list of all of the application's users.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::select('select * from users where active = ?', [1]);

        return view('user.index', ['users' => $users]);
    }
}

```

Σχήμα 13 "Παράδειγμα χρήσης Raw Queries"

## 7.5 Query Builder

To Laravel Query Builder είναι ένα εύχρηστο και πρακτικό interface για την δημιουργία και την εκτέλεση queries της βάσης δεδομένων. Μπορεί να χρησιμοποιηθεί για τις περισσότερες ενέργειες με την βάση δεδομένων της εφαρμογής και λειτουργεί με όλους τους υποστηριζόμενους τύπους της βάσης δεδομένων. To Query Builder χρησιμοποιεί το PDO parameter binding για την προστασία από επιθέσεις SQL injection. Αυτή η μέθοδος επικοινωνίας αποτελεί μια λύση ανάμεσα στα Raw Queries που περιεγράφηκαν στην προηγούμενη ενότητα και στο Eloquent ORM που θα περιγραφεί στην συνέχεια.

Για την μεγαλύτερη κατανόηση αυτού του τρόπου επικοινωνίας, μπορεί να γίνει η παρομοίωση πως το Query Builder είναι μια πιο περιγραφική μέθοδος για την διατύπωση SQL queries, με ένα συντακτικό που παραπέμπει σε αντικειμενοστραφή προγραμματισμό. Για αυτό τον λόγο από το Query Builder υποστηρίζεται το μεγαλύτερο μέρος των SQL queries, όπως SELECT, INSERT, WHERE, JOIN, UNION, AVG, MIN, MAX κ.τ.λ.

```

class UserController extends Controller
{
    public function index()
    {
        $users = DB::table('users')->get();

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

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

```

Σχήμα 14 "Παράδειγμα χρήσης Query Builder"

## 7.6 Eloquent ORM

Ο τρίτος και τελευταίος τρόπος επικοινωνίας με την βάση δεδομένων είναι το Eloquent ORM. Συνοπτικά θα μπορούσε να αναφερθεί πως το Eloquent ORM προσφέρει μια απλή και όμορφη εφαρμογή του Active Record για την αλληλεπίδραση με την βάση δεδομένων. Θα μπορούσε να γίνει παραλληλισμός του Query Builder και του Eloquent ORM λέγοντας πως οι Eloquent κλάσεις (models) είναι σαν μια ισχυρή υλοποίηση του Query Builder, με σημαντική διαφορά πως εντός του κώδικα τα queries δεν γίνονται απ' ευθείας στην βάση δεδομένων αλλά σε κάποιο model το οποίο το framework το δρομολογεί στον αντίστοιχο πίνακα της βάσης δεδομένων. Ακολουθεί εκτενέστερη ανάλυση του Eloquent ORM στο επόμενο κεφάλαιο.

## 8. Eloquent ORM

### 8.1 Εισαγωγή

Όπως αναφέρθηκε στην προηγούμενη ενότητα, ο μηχανισμός Eloquent ORM που περιέχεται στην Laravel παρέχει μία όμορφη και απλή υλοποίηση του Active Record για την αλληλεπίδραση με την βάση δεδομένων. Κάθε πίνακας της βάσης δεδομένων έχει ένα αντίστοιχο Model το οποίο χρησιμοποιείται για την αλληλεπίδραση με τον πίνακα. Το ORM στην ονομασία αποτελεί αρχικά του “Object Relational Mapping”, το οποίο είναι μια προγραμματιστική τεχνική για την μετατροπή δεδομένων από ασύμβατα type systems χρησιμοποιώντας αντικειμενοστραφείς γλώσσες προγραμματισμού.

Όπως γίνεται εύκολα κατανοητό τα Models στα οποία έγινε αναφορά στην ενότητα 5.5 δεν είναι απλά κάποιες κλάσεις αλλά αποτελούν τα Eloquent Models. Αυτό σημαίνει πως επιτρέπουν την εκτέλεση ορισμένων ενεργειών που παρέχονται από το Eloquent ORM. Σε αυτό το σημείο είναι χρήσιμο να γίνει υπενθύμιση πως συνηθίζεται τα models να ονομάζονται με το όνομα του πίνακα που αντιστοιχούν σε ενικό αριθμό (π.χ. Posts). Σε περίπτωση που δεν ακολουθείται αυτό το μοτίβο μπορεί να γίνει δήλωση του πίνακα εντός του Model στην μεταβλητή “protected \$table”. Επίσης το Eloquent υποθέτει πως κάθε πίνακας έχει σαν κύριο κλειδί την στήλη “id” η οποία είναι τύπου “INCREMENT INT”. Και πάλι ο εντός του κώδικα μπορεί να δηλωθεί κάποια άλλη στήλη σαν κύριο κλειδί. Τέλος, εξ’ ορισμού το Eloquent περιμένει πως κάθε πίνακας έχει δύο στήλες “created\_at” και “updated\_at”, οι οποίες αποτελούν τα timestamps κάθε εγγραφής του πίνακα. Σε περίπτωση που δεν υπάρχει η επιθυμία για την ύπαρξη timestamps δίνεται η δυνατότητα απενεργοποίησης τους στην μεταβλητή “public \$timestamps=false;”

### 8.2 CRUD Operations – Read

Μετά την δημιουργία ενός μοντέλου και του αντίστοιχου πίνακα στην βάση δεδομένων μπορεί να ξεκινήσει η ανάκτηση δεδομένων από την βάση. Μέσω του Eloquent ORM που όπως προειπώθηκε είναι μια ισχυρή υλοποίηση του Query Builder δίνεται η δυνατότητα για ανάκτηση μίας ή και παραπάνω εγγραφής σε συνδυασμό και με πιθανές συνθήκες που πρέπει να ισχύουν. Είναι προφανές πως η συγγραφή ερωτημάτων μέσω του Eloquent ORM είναι αρκετά εύκολη για κάποιον με μέτρια γνώση της SQL πόσο μάλλον για κάποιον που είναι περισσότερο εξοικειωμένος με την συγγραφή πολύπλοκων ερωτημάτων.

Σε περίπτωση ανάκτησης πολλαπλών εγγραφών μέσω των μεθόδων “all” και “get”, επιστρέφεται από την Laravel ένα αντικείμενο της κλάσης “Illuminate\Database\Eloquent\Collection”. Η συγκεκριμένη κλάση προσφέρει έναν μεγάλο αριθμό helper μεθόδων για την ευκολότερη και φυσικά αποτελεσματικότερη

διαχείριση των αποτελεσμάτων. Επίσης η μέθοδος “cursor” επιτρέπει την ανάγνωση των εγγραφών ενός πίνακα με την χρήση SQL Cursors, επιτυγχάνοντας αισθητά μικρότερη χρήση της μνήμης.

Στο σχήμα που ακολουθεί φαίνονται κάποια παραδείγματα ανάγνωσης από τον πίνακα της βάσης δεδομένων

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

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

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

Σχήμα 15 "Παράδειγμα CRUD - Read"

### 8.3 CRUD Operations – Create

Η δημιουργία εγγραφών επιτυγχάνεται μέσα από μια εξαιρετικά απλή λογική. Αρχικά δημιουργείται ένα αντικείμενο της κλάσης (ή πιο σωστά ένα model instance), στην συνέχεια προσδίδονται τιμές σε αυτό το αντικείμενο και τέλος καλείται η μέθοδος “save”. Ένας γρηγορότερος τρόπος είναι το λεγόμενο “Mass Assignment”, στο οποίο κατά την δημιουργία του model instance, το αντικείμενο δέχεται σαν παράμετρο ένα πίνακα από τις τιμές που θα δημιουργηθούν. Για την λειτουργία αυτής της μεθόδου πρέπει να έχουν δηλωθεί όλα τα κατάλληλα πεδία εντός του model στην μεταβλητή “protected \$fillable = []”.

```

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();
    }
}

```

Σχήμα 16 "Παράδειγμα CRUD - Create"

## 8.4 CRUD Operations - Update

Παρόμοια λογική και υλοποίηση με την δημιουργία έχει και η τροποποίηση και ενημέρωση εγγραφών. Η μόνη διαφορά εντοπίζεται στο πρώτο βήμα, καθώς αντί για την δημιουργία νέου αντικειμένου η εγγραφή ανακτάται από την βάση δεδομένων σε ένα αντικείμενο ή πιο σωστά σε ένα model instance. Στην συνέχεια δίνονται οι ενημερωμένες τιμές και η εγγραφή αποθηκεύεται μέσω της μεθόδου "save". Παρομοίως με το "Mass Assignment" υπάρχει και το «Mass Update». Για την εξάλειψη της ανάγκης ανάκτησης πολλαπλών εγγραφών και την αποφυγή συγγραφής βρόγχων εντός του κώδικα η Laravel παρέχει ένα ευανάγνωστο τρόπο σύνταξης.

```

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();
    }
}

```

Σχήμα 17 "Παράδειγμα CRUD - Update"

## 8.5 CRUD Operations – Delete

Η διαγραφή μιας εγγραφής γίνεται μέσω της μεθόδου “delete” σε ένα model instance. Επίσης παρέχεται η δυνατότητα του λεγόμενου “Soft Delete”. Το “Soft Delete” σημαίνει πως η εγγραφή δεν διαγράφεται στην πραγματικότητα από τον πίνακα αλλά ένα πεδίο με το όνομα “deleted\_at” παίρνει μια τιμή του τύπου “TIMESTAMP”.

```
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]);
    }
}
```

Σχήμα 18 "Παράδειγμα CRUD - Delete"

## 8.6 Relationships (Σχέσεις)

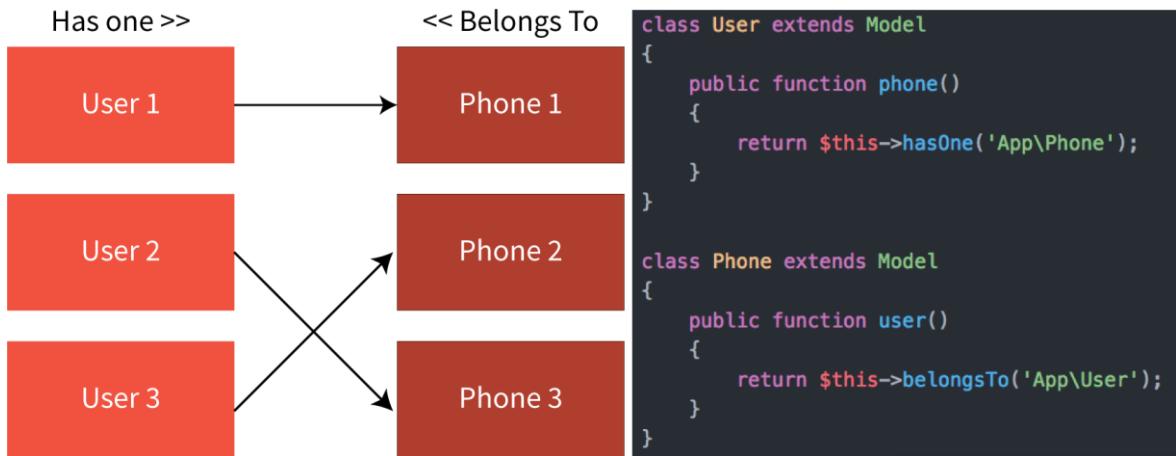
Συχνά υπάρχουν αναφορές ανάμεσα στους πίνακες μιας βάσης δεδομένων. Μέσω του Eloquent η διαχείριση αυτών των σχέσεων γίνεται εξαιρετικά εύκολη αφού υποστηρίζει μια πληθώρα σχέσεων. Οι σχέσεις αυτές, ή όπως είναι η επίσημη ονομασία “Eloquent Relationships”, δηλώνονται σαν μέθοδοι μέσα στις κλάσεις των Eloquent models. Όπως και τα Eloquent Models έτσι και τα αντίστοιχα Eloquent Relationships αποτελούν μια ισχυρή υλοποίηση του Query Builder, αφού η δήλωση τους σαν μέθοδοι παρέχουν την δυνατότητα method chaining και δυνατότητες querying στην βάση δεδομένων.

Επίσης ο έξυπνος μηχανισμός του Eloquent αντιστοιχίζει τα ξένα κλειδιά των πινάκων βάση της ονομασίας τους. Όπως είπαμε συνηθίζεται κάθε πίνακας να έχει μια στήλη id η οποία αποτελεί και το κύριο κλειδί. Όταν σε έναν διαφορετικό πίνακα υπάρχει μια στήλη η οποία έχει σαν όνομα την ονομασία κάποιου πίνακα στον ενικό αριθμό και ακολουθείται από μια κάτω παύλα και την λέξη “id”, τότε το Eloquent συμπεραίνει πως αυτή η στήλη αποτελεί το ξένο κλειδί του αντίστοιχου πίνακα.

Παρακάτω περιγράφονται συνοπτικά όλοι οι τύποι relationships μαζί με ένα παράδειγμα προς κατανόησή τους:

### One to One ('Ένα προς Ένα):

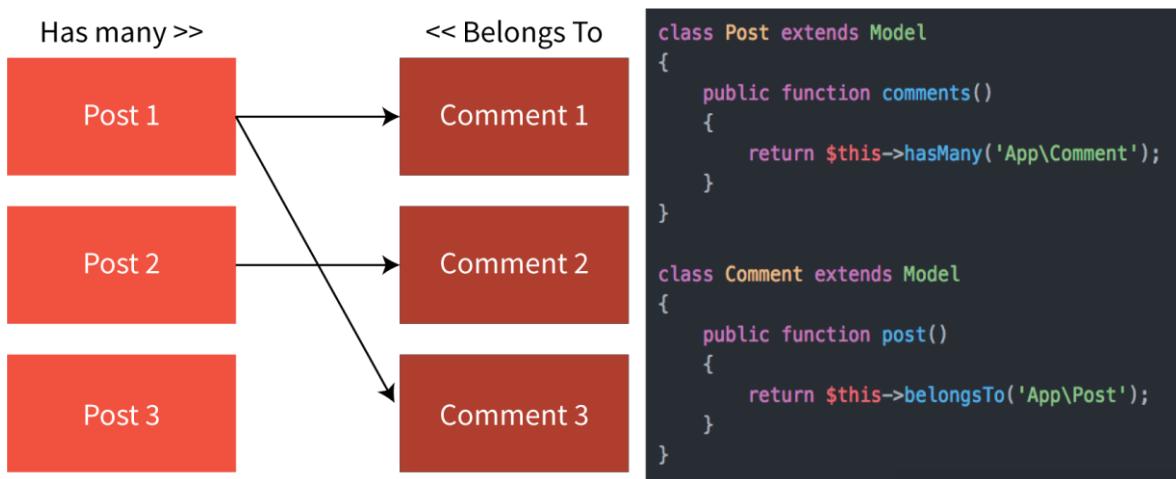
Αυτός ο τύπος σχέσης είναι και ο πιο βασικός. Ένα model **έχει** κάποιο άλλο model, ενώ το δεύτερο **ανήκει** στο πρώτο.



Σχήμα 19 "Η σχέση One to One"

### One to Many ('Ένα προς Πολλά):

Σε αυτή τη σχέση ένα model έχει ένα πλήθος από άλλα models. Για παράδειγμα ένα Post έχει ένα πλήθος Comments, ενώ παράλληλα ένα Comment ανήκει σε ένα Post.

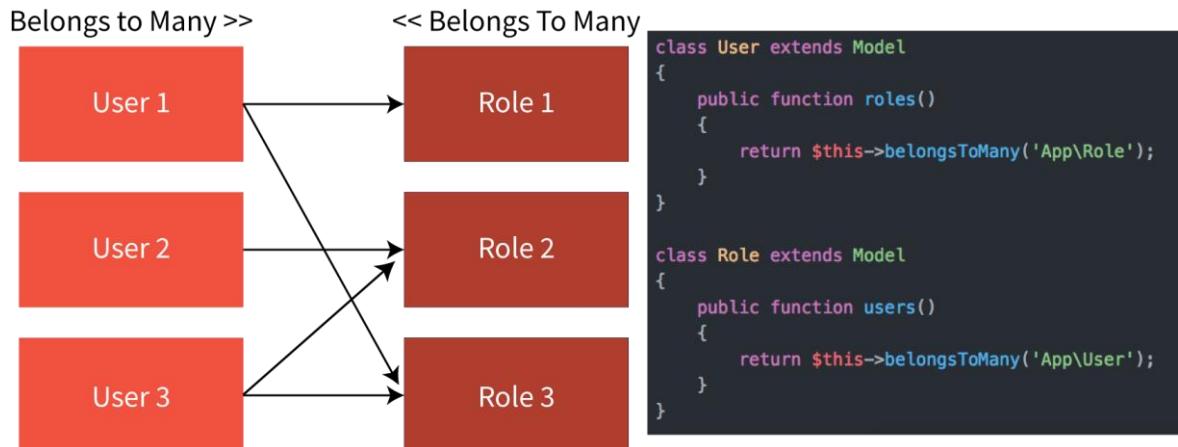


Σχήμα 20 "Η σχέση One to Many"

### Many to Many (Πολλά προς Πολλά):

Η Many-to-many σχέση είναι ελαφρώς πιο πολύπλοκη από τις δύο προηγούμενες. Ένα παράδειγμα μιας τέτοιας σχέσης είναι ένας "user" ο οποίος έχει διάφορα "roles", τα οποία "roles" μοιράζονται και από άλλους χρήστες της εφαρμογής. Για αυτή την σχέση χρειάζονται τρείς πίνακες: users, roles και role\_user. Ο πίνακας role\_user παράγεται από

την αλφαριθμητική σειρά των σχετικών models και περιέχει τις στήλες “user\_id” και “role\_id”



Σχήμα 21 "Η Σχέση Many to Many"

### Has Many Through (Πολλά μέσω)

Η συγκεκριμένη σχέση παρέχει μια «παράκαμψη» για την πρόσβαση σε απομακρυσμένες σχέσεις μέσω μιας ενδιάμεσης σχέσης. Για παράδειγμα ένα model “Country” μπορεί να έχει πολλά “Post” models μέσω ενός ενδιάμεσου model “User”. Ενώ ο πίνακας “posts” δεν έχει κάποιο ξένο κλειδί για το “Country”, με την σχέση Has Many Through δίνεται πρόσβαση μέσω του “\$country->posts”. Είναι φανερό πως με μόνο μια μικρή γραμμή κώδικα, η Laravel πραγματοποιεί πολλαπλά queries και μάλιστα με SQL JOINS.

```

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');
    }
}

```

Σχήμα 22 "Η Σχέση Has Many Through"

## Polymorphic (Πολυμορφικές)

Οι πολυμορφικές σχέσεις επιτρέπουν σε ένα model να ανήκει σε περισσότερα από ένα models σε μια ενιαία συσχέτιση. Για παράδειγμα οι χρήστες μπορούν να σχολιάζουν τόσο σε άρθρα όσο και σε βίντεο. Με τις πολυμορφικές σχέσεις μπορούμε να χρησιμοποιήσουμε 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

Σχήμα 23 "Η σχέση Polymorphic"

## Many to Many Polymorphic (Πολλά προς Πολλά Πολυμορφικές)

Πέραν των απλών πολυμορφικών σχέσεων υπάρχουν και οι Many to Many Polymorphic σχέσεις. Για παράδειγμα ένα “Post” και ένα “Video” μπορούν να μοιράζονται μια πολυμορφική σχέση με ένα “Tag”. Χρησιμοποιώντας μια many-to-many polymorphic σχέση, δίνεται η δυνατότητα για πρόσβαση σε μια λίστα μοναδικών tags τα οποία υπάρχουν τόσο στα posts όσο και στα videos.

```

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');
    }
}

```

The screenshot shows a code editor with two panels. The left panel contains the PHP code for the `Post` and `Tag` models. The right panel shows the corresponding database schema with three tables: `posts`, `videos`, and `tags`. Each table has an `id` column and a `name` column. The `posts` table has a many-to-many relationship with both `tags` and `videos` via the `taggables` table, which includes columns for `tag\_id`, `taggable\_id`, and `taggable\_type`.

Σχήμα 24 "Η σχέση Many To Many Polymorphic"

## 8.7 Eager Loading

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. Παρ’ όλα αυτά το Eloquent μπορεί να φορτώσει τις σχέσεις με την μέθοδο του “eager loading”, δηλαδή την στιγμή που γίνεται το query στο model-γονέα. Το eager loading ελαττώνει το πρόβλημα των N+1 queries και επιτυγχάνεται με την μέθοδο “with” κατά την εκτέλεση των queries.

```

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

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

```

Σχήμα 25 "Η χρήση του Eager Loading"

## 8.8 Accessors & Mutators

Οι Accessors και οι Mutators βοηθούν στην διαμόρφωση των τιμών των χαρακτηριστικών κατά την ανάκτηση ή τον ορισμό τους αντίστοιχα. Μπορούν να αποδειχθούν ιδιαίτερα χρήσιμοι αφού η διαμόρφωση που πετυχαίνουν ορίζεται μια

φορά εντός της κλάσης του model χωρίς να χρειάζεται να γράφεται κάθε φορά η ρουτίνα που εκτελούν.

```
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);
        }
    }
}
```

Σχήμα 26 "Παράδειγμα accessor & mutator"

## 9. Εμβάθυνση (Digging Deeper)

### 9.1 Αυθεντικοποίηση

Ένα από τα προβλήματα το οποίο αποσκοπούσε η δημιουργία της Laravel να λύσει ήταν η απλή και γρήγορη δημιουργία και εφαρμογή ενός ασφαλούς συστήματος αυθεντικοποίησης. Έτσι, από την πρώτη της beta έκδοση μέχρι και σήμερα η Laravel κάνει την υλοποίηση ενός συστήματος αυθεντικοποίησης να μοιάζει εξαιρετικά απλή και εύκολη.

Η δημιουργία όλου του σκελετού για την αυθεντικοποίηση πετυχαίνεται με την εκτέλεση μιας και μοναδικής εντολής: “`php artisan make:auth`”. Μετά το πέρας της συγκεκριμένης εντολής έχουν δημιουργηθεί όλα τα απαραίτητα `views`, `models`, `routes` κ.τ.λ. που χρειάζεται η εφαρμογή και πλέον εφαρμόζει ένα πλήρως ασφαλές και κατανοητό σύστημα αυθεντικοποίησης των χρηστών της. Τις περισσότερες φορές δεν απαιτείται καμία παρέμβαση στις ρυθμίσεις της αυθεντικοποίησης, χωρίς αυτό να σημαίνει όμως πως δεν δίνεται η δυνατότητα για τροποποίηση όλων των δομικών της στοιχείων.

### 9.2 Caching

Όπως ειπώθηκε και σε προηγούμενες ενότητες η Laravel παρέχει ένα εκφραστικό, ενοποιημένο API για διάφορα caching backends. Οι ρυθμίσεις της cache βρίσκονται στο αρχείο “`config/cache.php`”. Σε αυτό το αρχείο μπορεί να δηλωθεί ο cache driver που θα χρησιμοποιείται από την εφαρμογή καθώς και οι απαραίτητες ρυθμίσεις τους. Το framework υποστηρίζει ορισμένα δημοφιλή caching backends όπως το Memcached και το Redis. Με την χρήση της Cache παρατηρείται αισθητή διαφορά στην απόδοση και στους χρόνους απόκρισης της εφαρμογής.

### 9.3 Events

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 οποίοι δεν εξαρτώνται και δεν συνδέονται μεταξύ τους.

### 9.4 Mail

Η χρήση του email σε μια εφαρμογή Laravel είναι εξαιρετικά απλή μιας και παρέχει ένα απλό API για την βιβλιοθήκη SwiftMail. Η συγκεκριμένη βιβλιοθήκη παρέχει drivers για SMTP, Mailgun, SparkPost, Amazon SES και την μέθοδο mail της PHP. Απαραίτητη

προϋπόθεση για την χρήση της βιβλιοθήκης είναι η εγκατάσταση της βιβλιοθήκης Guzzle HTTP (composer require guzzlehttp/guzzle).

Με την απλή αυτή υλοποίηση 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”. Παρατηρείται λοιπόν πως ακόμα και η αποστολή emails γίνεται με μια αντικειμενοστραφή προσέγγιση.

## 9.5 Notifications

Εκτός από την αποστολή emails, η Laravel παρέχει υποστήριξη για την αποστολή ειδοποιήσεων μέσα από μια ποικιλία καναλιών μετάδοσης όπως email, SMS (μέσω του Nexmo) και Slack. Οι ειδοποιήσεις επίσης είναι δυνατό να αποθηκεύονται στην βάση δεδομένων και να προβάλλονται στο Web interface.

Πιστή στην αντικειμενοστραφή προσέγγιση, η Laravel αναπαριστά τις ειδοποιήσεις με μια κλάση η οποία βρίσκεται στον κατάλογο “app/Notifications”. Παρατηρείται εύκολα πως η υλοποίηση είναι αρκετά παρόμοια με την υλοποίηση του Mail.

## 9.6 Queues

Μέσω των queues επιτρέπεται η αναβολή ορισμένων εργασιών που απαιτούν μεγάλη επεξεργασία και χρόνο (π.χ αποστολή email) συνεπώς και μεγάλους πόρους, για κάποιο χρονικό διάστημα. Το γεγονός αυτό αυξάνει αισθητά την ταχύτητα απόκρισης της εφαρμογής όταν δέχεται Web Requests. Μια πιο απλή εξήγηση είναι πως η εφαρμογή περιμένει να εκτελέσει τις εργασίες της ουράς όταν δεν απασχολείται από κάποιο web request.

To framework παρέχει ένα ενοποιημένο API για μια πληθώρα queue backends όπως τα Beanstalk, Amazon SQS, Redis ή ακόμα και μια σχεσιακή βάση δεδομένων.

## 9.7 Scheduling

Η συνηθέστερη πρακτική για μια εφαρμογή διαδικτύου είναι η δημιουργία ενός Cron entry για κάθε εργασία (task) που πρέπει να εκτελεστεί σε ένα συγκεκριμένο χρονοδιάγραμμα (schedule). Αυτή όμως η πρακτική μπορεί να αποδειχθεί επίπονη καθώς αυτό το schedule δεν υποβάλλεται στο source control και συνεπώς κάθε νέο entry πρέπει να δηλώνεται κατευθείαν στο production περιβάλλον.

Αυτό το πρόβλημα ήρθε να λύσει το Laravel Scheduler το οποίο επιτρέπει την σαφή και κατανοητή διατύπωση όλων αυτών των tasks μέσα στην ίδια την εφαρμογή. Με την χρήση του scheduler απαιτείται μόνο ένα cron entry στον server. Το χρονοδιάγραμμα δηλώνεται στην μέθοδο “schedule” του αρχείου “app\Console\Kernel.php”.

Το μοναδικό cron entry που απαιτείται είναι το “`* * * * * php /path-to-your-project/artisan schedule:run`”, το οποίο ουσιαστικά «ακούει» κάθε ένα λεπτό το Laravel Scheduler.

## 10. Παράδειγμα: Υλοποίηση Εφαρμογής «Λίστα Καθηκόντων»

### 10.1 Εισαγωγή

Στο παρόν κεφάλαιο θα γίνει μια αναλυτική παρουσίαση των βημάτων που απαιτούνται για την δημιουργία μιας απλής εφαρμογής που θα λειτουργεί σαν μια «λίστα καθηκόντων» ή αλλιώς to-do list. Ο χρήστης θα μπορεί να δημιουργεί έναν λογαριασμό μέσω του οποίου θα μπορεί να δημιουργεί διάφορα καθήκοντα. Για χάρη ευκολίας τα καθήκοντα αυτά θα είναι διαθέσιμα σε όλους τους χρήστες της εφαρμογής και όχι μόνο στον χρήστη που τα δημιούργησε. Μέσα από αυτό το παράδειγμα θα παρουσιαστούν στην πράξη κάποια από τα πιο βασικά χαρακτηριστικά της Laravel και έχει ως σκοπό την καλύτερη εξοικείωση με το framework.

### 10.2 Εγκατάσταση

Φυσικά το πρώτο πράγμα που πρέπει να γίνει είναι η εγκατάσταση του framework στο τοπικό περιβάλλον ανάπτυξης. Για τους σκοπούς το παραδείγματος θα χρησιμοποιηθεί σαν περιβάλλον το “Laravel Valet” για Mac OS, παρ’ όλα αυτά η διαδικασία είναι η ίδια για όλα τα περιβάλλοντα (Homestead, XAMPP κ.τ.λ.). Η εγκατάσταση γίνεται με την εντολή:

```
$- laravel new tasks
```

Η παραπάνω εντολή θα δημιουργήσει έναν νέο κατάλογο με το όνομα “tasks” (το όνομα του project) στον οποίο θα υπάρχει η αρχική εγκατάσταση της Laravel. Αξίζει να σημειωθεί ότι για η χρήση της παραπάνω εντολής προϋποθέτει τις κατάλληλες ρυθμίσεις όπως περιγράφονται στην ενότητα 2.3.

Μετά την εγκατάσταση του framework μπαίνουμε στον κατάλογο που δημιουργήθηκε αφού όλες οι επόμενες εντολές θα πρέπει να εκτελούνται εντός αυτού.

```
$- cd tasks
```

### 10.3 Σύνδεση με την Βάση Δεδομένων και υλοποίηση συστήματος αυθεντικοποίησης

Είναι λογικό πως η εφαρμογή θα πρέπει να χρησιμοποιεί μια βάση δεδομένων για την αποθήκευση των tasks όπως και για τα στοιχεία των χρηστών της εφαρμογής. Ο τύπος της βάσης δεδομένων μπορεί να είναι ένας από αυτούς που περιεγράφηκαν στην ενότητα 7.1. Στο παράδειγμα αυτό θα χρησιμοποιήσουμε την MySQL. Μετά την δημιουργία της βάσης δεδομένων (μέσω Terminal, Sequel Pro, phpMyAdmin κ.α.) πρέπει να γίνει η κατάλληλη τροποποίηση του αρχείου “.env”. Υπενθυμίζεται πως το

αρχείο αυτό περιέχει ευαίσθητες πληροφορίες σχετικά με την εφαρμογή (ενότητα 3.3). Κάποιες από αυτές τις πληροφορίες είναι και τα στοιχεία σύνδεσης με την βάση δεδομένων. Επεξεργαζόμαστε λοιπόν τα αντίστοιχα πεδία και θέτουμε τις τιμές που υπάρχουν για το περιβάλλον μας.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:fIioSwMrhppjphfbLs4zw3QllSQQwHD1lc7FwcnNFyg=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tasks
DB_USERNAME=username
DB_PASSWORD=password
```

Πλέον η εφαρμογή είναι έτοιμη να επικοινωνήσει με την βάση δεδομένων που έχουμε ορίσει. Επόμενο μας βήμα είναι η χρήση του ενσωματωμένου συστήματος αυθεντικοποίησης που παρέχει η Laravel. Το μόνο που απαιτείται είναι η εκτέλεση της εντολής:

```
$- php artisan make:auth
```

Με την εκτέλεση της παραπάνω εντολής το Framework δημιουργεί αυτόματα για εμάς όλες τις μεθόδους, routes, views κ.τ.λ. που χρειάζονται για ένα ισχυρό και ασφαλές σύστημα αυθεντικοποίησης. Για χάρη ευκολίας δεν θα ασχοληθούμε παραπάνω με την παραμετροποίηση του συστήματος αυτού παρά μόνο θα χρησιμοποιήσουμε τις δυνατότητες register, login και logout.

#### 10.4 Δημιουργία Migration και Model

Όπως αναφέρθηκε σε προηγούμενες ενότητες η δημιουργία και τροποποίηση των πινάκων της βάσης δεδομένων γίνεται μέσω του μηχανισμού των migrations. Θα δημιουργήσουμε λοιπόν τον πίνακα που στον οποίο θα αποθηκεύονται τα tasks μέσω ενός migration. Μέσω του terminal εκτελούμε την εντολή:

```
$- php artisan make:migration create_tasks_table --create=tasks
```

Η εντολή αυτή θα δημιουργήσει ένα αρχείο στον κατάλογο database/migrations της μορφής {{datetime}}\_create\_tasks\_table.php. Παρατηρούμε πως έχουμε ορίσει εμείς το

όνομα που θέλουμε να έχει το αρχείο (create\_tasks\_table) το οποίο ακολουθεί τις οδηγίες ονοματολογίας της Laravel. Το datetime στην αρχή του ονόματος υπάρχει αποκλειστικά για να γνωρίζει το framework την σειρά με την οποία πρέπει να εκτελεστούν τα migrations. Επίσης στην εντολή έχουμε συμπεριλάβει το προαιρετικό αλλά χρήσιμο flag “-create=task”, μέσω του οποίου η Laravel ρυθμίζει κατάλληλα το αρχείο του migration σχετικά με το όνομα του πίνακα που πρόκειται να δημιουργήσει. Στο αρχείο η μέθοδος “up” περιγράφει την δημιουργία και τα πεδία του πίνακα και η μέθοδος “down” τι συμβαίνει σε περίπτωση οπισθοδρόμησης του migration. Στην μέθοδο up προσθέτουμε ακόμα μία στήλη τύπου «String» η οποία θα περιέχει το όνομα του task.

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

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

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

Μετά την αποθήκευση του αρχείου μπορούμε να «τρέξουμε» τα migrations με την εντολή:

```
$- php artisan migrate
```

Η εντολή αυτή θα δημιουργήσει 4 πίνακες (migrations, password\_resets, tasks, users). Υπάρχουν τρόποι για το πέρασμα εγγραφών στους πίνακες αυτούς κυρίως για λόγους testing μέσω των Seeders (ενότητα 7.3) ή μέσω του Laravel Tinker (ενότητα 5.1). Για

λόγους ευκολίας δεν θα χρησιμοποιηθεί κανένας από τους παραπάνω τρόπους, αλλά οι εγγραφές θα καταχωρηθούν μέσω φορμών της εφαρμογής (όπως δηλαδή και στην κανονική ροή). Είναι λοιπόν ένα καλό σημείο να δημιουργήσουμε τον πρώτο χρήστη της εφαρμογής. Πηγαίνουμε στο “tasks.test/register”<sup>3</sup> και χρησιμοποιούμε το γραφικό περιβάλλον για την δημιουργία ενός χρήστη.

Υπενθυμίζεται πως κάθε πίνακας της βάσης δεδομένων είναι άρρηκτα συνδεδεμένος με ένα Eloquent Model (ενότητα 5.5), αφού αποτελεί την αναπαράστασή του στην εφαρμογή. Έτσι το επόμενο βήμα είναι η δημιουργία ενός Model για τον πίνακα tasks αφού εξ’ ορισμού προϋπάρχει το αντίστοιχο model για τον πίνακα users. Η δημιουργία του model γίνεται μέσω της εντολής:

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

Το model που θα δημιουργηθεί θα τοποθετηθεί στον κατάλογο “app”. Αξίζει να σημειωθεί πως ενώ μετά την δημιουργία του το model Task είναι «άδειο» η Laravel το αντιστοιχίζει πολύ έξυπνα στον πίνακα “tasks” αφού είναι το όνομα του model στον πληθυντικό. Επίσης αντιστοιχίζει το κύριο κλειδί στην στήλη “id”.

```
namespace App;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Task extends Model  
{  
    //  
}
```

## 10.5 Routing

Σε αυτό το σημείο είμαστε έτοιμοι να δηλώσουμε όλες τις διαδρομές που θα υπάρχουν στην εφαρμογή μας (ενότητα 5.2). Υπενθυμίζεται ότι τα routes χρησιμοποιούνται σαν δείκτες σε μεθόδους controllers ή εκτελούν μια ανώνυμη μέθοδο όταν ο χρήστης δημιουργεί αίτημα για μια σελίδα στην εφαρμογή. Τα routes δηλώνονται στο αρχείο “routes/web.php”.

Για αυτή την εφαρμογή γνωρίζουμε πως θα χρειαστούμε τουλάχιστον τρείς διαδρομές: μια για την προβολή όλων των tasks, μια για την δημιουργία ενός task και μια για την διαγραφή ενός task. Συμπληρώνουμε λοιπόν τον παρακάτω κώδικα στο αρχείο web.php:

---

<sup>3</sup> Το domain ενδέχεται να διαφέρει ανάλογα με το τοπικό περιβάλλον ανάπτυξης (π.χ. localhost:8000)

```

14 Route::get('/', function () {
15
16 });
17
18 Route::post('/task', function () {
19
20 });
21
22 Route::delete('/task/{task}', function () {
23
24 });
25
26 Auth::routes();
27
28 Route::get('/home', 'HomeController@index')->name('home');
29

```

Επίσης παρατηρούμε πως στο αρχείο αυτό υπάρχουν από πριν συμπληρωμένες δύο γραμμές (26 και 28). Αυτές οι γραμμές συμπληρώθηκαν αυτόματα μετά την εκτέλεση της εντολής “`php artisan make:auth`” και αφορούν όλες τις διαδρομές που εξυπηρετούν στην αυθεντικοποίηση των χρηστών.

Όσον αφορά τις διαδρομές τις οποίες καθορίσαμε εμείς η πρώτη (14-16) αφορά στην προβολή όλων των tasks όταν ο χρήστης ζητάει την αρχική σελίδα. Η δεύτερη (18-20) αφορά στην δημιουργία ενός task μέσω ενός POST request ενώ η Τρίτη και τελευταία (22-24) στην διαγραφή ενός task μέσω ενός DELETE request. Το συγκεκριμένο URL δέχεται μια παράμετρο (`{task}`), η οποία είναι το id του task που πρέπει να διαγραφεί. Επίσης αρχικά οι διαδρομές δεν θα εκτελούν κάποια μέθοδο ενός Controller αλλά μία ανώνυμη μέθοδο.

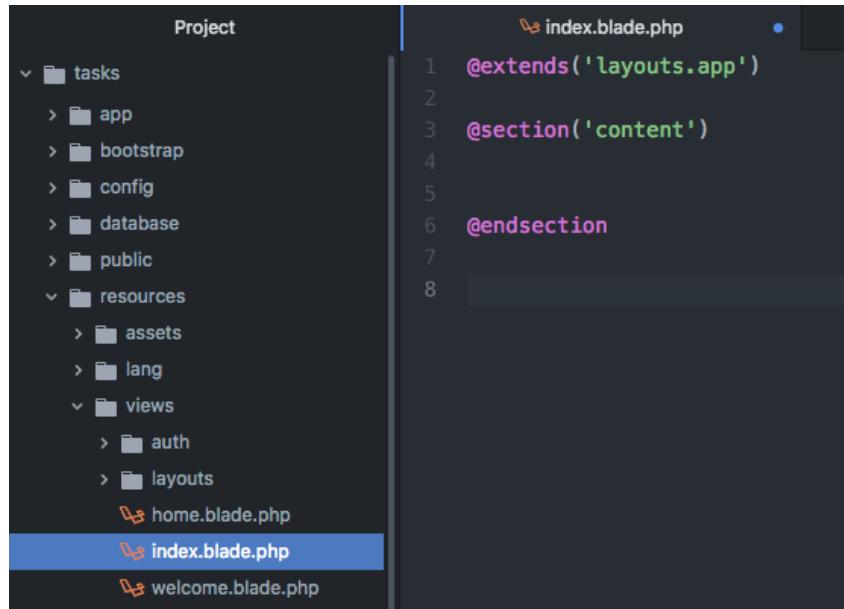
## 10.6 Views

Για την προβολή των tasks θα χρειαστούμε ένα αρχείο View (ενότητα 5.6). Δημιουργούμε ένα αρχείο view στον κατάλογο resources με την ονομασία “`index.blade.php`”. Υπενθυμίζεται σε αυτό το σημείο πως το `*.blade.php` δηλώνει πως το view χρησιμοποιεί τον μηχανισμό Blade Templating της Laravel (ενότητα 6.1). Το view αυτό θα «επεκτείνει» το βασικό layout που υπάρχει στο `resources/views/layouts/app.blade.php`<sup>4</sup>. Όλο το περιεχόμενο της σελίδας θα

---

<sup>4</sup> Στο αρχείο αυτό έχουν προστεθεί και τα απαραίτητα αρχεία για το Bootstrap 4. Περισσότερα στο <https://getbootstrap.com/docs/4.1/getting-started/introduction/>

περιέχεται στο section “content” του layout. Το αρχείο index.blade.php θα πρέπει να είναι αρχικά το εξής:



```
Project
  tasks
    app
    bootstrap
    config
    database
    public
  resources
    assets
    lang
    views
      auth
      layouts
      home.blade.php
      index.blade.php (highlighted)
      welcome.blade.php

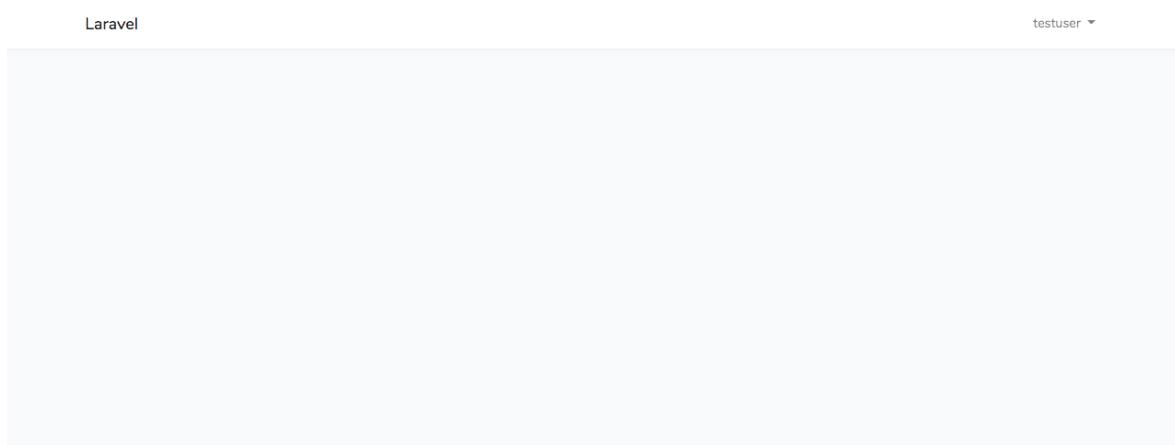
index.blade.php
1 @extends('layouts.app')
2
3 @section('content')
4
5
6 @endsection
7
8
```

Τώρα είναι η κατάλληλη στιγμή ώστε να ενημερώσουμε και το αντίστοιχο route που θέλουμε να επιστρέψει αυτό το view. Πηγαίνουμε στο “routes/web.php” και το τροποποιούμε ως εξής:

```
14 Route::get('/', function () {
15     return view('index');
16 });


```

Η προσθήκη της γραμμής 15 έχει ως αποτέλεσμα κάθε request στην αρχική σελίδα της εφαρμογής να επιστρέφει το view “index.blade.php”.



Εφόσον όλα δουλεύουν καλά μπορούμε να δημιουργήσουμε το γραφικό περιβάλλον της εφαρμογής με την χρήση στοιχείων από το CSS Framework “Bootstrap”. Προσθέτουμε τον παρακάτω κώδικα στο αρχείο “index.blade.php”:

```

1  @extends('layouts.app')
2
3  @section('content')
4
5  <div class="container">
6      <div class="row">
7          <div class="col-6">
8              <div class="card">
9                  <div class="card-body">
10
11                  <form>
12                      <div class="form-group">
13                          <input type="text" class="form-control" name="task" placeholder="Enter task">
14                      </div>
15                      <button type="submit" class="btn btn-primary">Submit</button>
16                  </form>
17
18              </div>
19          </div>
20      <div class="col-6">
21          <div class="card">
22              <div class="card-body">
23
24                  <table class="table table-striped">
25                      <thead>
26                          <tr>
27                              <th>Name</th>
28                              <th></th>
29                          </tr>
30                      </thead>
31                      <tbody>
32                          <tr>
33                              <td>Task #1</td>
34                              <td class="text-right"><a href="#" class="btn btn-sm btn-danger">Delete</a></td>
35                          </tr>
36                      </tbody>
37                  </table>
38
39              </div>
40          </div>
41      </div>
42  </div>
43 </div>
44 </div>
45
46 @endsection

```

Ο παραπάνω κώδικας αποτελεί απλά μια αρχική «σκαλωσιά» την οποία θα τροποποιήσουμε αργότερα ούτως ώστε να προβάλει δυναμικά τις εγγραφές από την βάση δεδομένων καθώς και μέσω της φόρμας να γίνεται καταχώρηση νέων tasks. Το αποτέλεσμα του πρέπει να είναι το παρακάτω:

The screenshot shows a Laravel application interface. At the top left is the Laravel logo, and at the top right is a dropdown menu set to "testuser". Below the header, there are two main sections. On the left, a form for creating a new task is displayed, featuring a text input field labeled "Enter task" and a blue "Submit" button. On the right, a table lists existing tasks. The table has two columns: "Name" and an empty column. The first task listed is "Task #1", which includes a red "Delete" button in its row.

## 10.7 Δημιουργία ενός Task

Για την δημιουργία ενός task απαιτούνται δύο βήματα: Πρώτον η τροποποίηση του view “index” ούτως ώστε να καταστεί λειτουργική η φόρμα δημιουργίας που δημιουργήσαμε και δεύτερον η υλοποίηση του κώδικα η οποία δημιουργεί το αντίστοιχο model και το αποθηκεύει στην βάση δεδομένων. Υπενθυμίζεται πως η δημιουργία ενός task γίνεται μετά από ένα POST Request στο URL “/task”. Τα δεδομένα που καταχωρούνται στην φόρμα αποθηκεύονται σε ένα αντικείμενο τύπου Request ούτως ώστε να είναι δυνατή η διαχείριση τους.

Αρχικά ας τροποποιήσουμε το view σύμφωνα με τον παρακάτω κώδικα:

```
<form action="{{ url('/task') }}" method="POST">
    @csrf
    <div class="form-group">
        <input type="text" class="form-control" name="task" placeholder="Enter task">
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Οι αλλαγές που έγιναν είναι τρείς:

1. Προστέθηκε το action της φόρμας. Μέσω του Blade μπορούμε να ορίσουμε σαν action ένα συγκεκριμένο URL με την παραπάνω σύνταξη. Στο παράδειγμα αυτό δεν διαφέρει σε τίποτα από την κανονική δήλωση του URL, αλλά μπορεί να αποδειχτεί εξαιρετικά χρήσιμο με άλλες μεθόδους που θα δούμε παρακάτω (π.χ. όνομα route).
2. Προστέθηκε η μέθοδος (POST)
3. Προστέθηκε το πεδίο “@csrf” το οποίο δημιουργεί ένα κρυφό input με το token του χρήστη για προστασία από επιθέσεις CSRF. Είναι απαραίτητο κάθε φορά που γίνεται submit μια φόρμα.

Μετά την τροποποίηση του view μπορούμε να προχωρήσουμε και στην υλοποίηση του κώδικα. Σε αυτό το παράδειγμα η υλοποίηση θα γίνει κατευθείαν μέσα στην ανώνυμη μέθοδο που καλείται από την διαδρομή στο αρχείο των routes. Η λογική του κώδικα είναι πως σε κάθε request δημιουργείται ένα Task Model το οποίο παίρνει τιμές από την φόρμα και μετά αποθηκεύεται στην βάση δεδομένων. Ο κώδικας για αυτή την υλοποίηση είναι εξαιρετικά απλός:

```

18 Route::post('/task', function (Illuminate\Http\Request $request) {
19     $newTask = new \App\Task;
20
21     $newTask->name = $request->task;
22
23     $newTask->save();
24
25     return redirect('/');
26 });

```

Αναλυτικότερα η κάθε γραμμή:

18. Δήλωση μεθόδου, URL, και ως παράμετρο στην ανώνυμη μέθοδο ένα αντικείμενο τύπου “Illuminate\Http\Request” στο οποίο περιέχονται όλα τα πεδία της φόρμας.
19. Δημιουργία αντικειμένου model Task
21. Τοποθέτηση στο πεδίο “name” του Task model της τιμής του πεδίου “task” από το αντίστοιχο πεδίο της φόρμας
23. Αποθήκευση του model στην βάση δεδομένων
25. Ανακατεύθυνση στην αρχική σελίδα

Βλέπουμε λοιπόν πόσο απλή υπόθεση είναι η δημιουργία εγγραφών στην βάση δεδομένων. Παρ’ όλα αυτά η δήλωση όλων των ενεργειών για την διαχείριση των αιτημάτων στην εφαρμογή σε ένα μόνο αρχείο είναι ευκολονόητο πως είναι αδύνατη καθώς η ανάγνωση του κώδικα καθίσταται αδύνατη. Για αυτόν τον λόγο χρησιμοποιούμε τους Controllers.

## 10.8 Μετάβαση στον Controller και προβολή των Tasks

Όπως αναφέρθηκε παραπάνω είναι αδύνατο όλη η λογική της εφαρμογής να βρίσκεται σε ένα μοναδικό αρχείο. Για αυτό τον λόγο χρησιμοποιούμε τους Controllers (ενότητα 5.7) που είναι άλλωστε βασικό συστατικό της αρχιτεκτονικής MVC. Έτσι στο αρχείο των routes δηλώνουμε απλά το URL, και την μέθοδο του Controller η οποία πρέπει να εκτελεστεί. Η οργάνωση των Controllers είναι στην ευχέρεια του προγραμματιστή μιας και υπάρχουν διάφορες μεθοδολογίες με συνηθέστερη αυτή της χρήσης ενός Controller για κάθε ένα model. Έτσι και στο παράδειγμα αυτό θα χρησιμοποιήσουμε έναν Controller (TasksController) ο οποίος θα περιέχει τρεις μεθόδους για τα αντίστοιχα routes. Ο Controller δημιουργείται μέσω της εντολής:

```
$- php artisan make:controller TasksController
```

Μετά την επιτυχή εκτέλεση της εντολής θα δημιουργηθεί το αρχείο “app/Http/Controllers/TasksController.php”. Σε αυτό το αρχείο θα δηλώσουμε τις τρεις προαναφερθείσες μεθόδους. Την μέθοδο “create()” για την δημιουργία των tasks, την μέθοδο “index()” για την προβολή των tasks και την μέθοδο “delete()” για την διαγραφή των tasks.

```
2 namespace App\Http\Controllers;
3
4 use Illuminate\Http\Request;
5
6 class TasksController extends Controller
7 {
8     public function create()
9     {
10        //
11    }
12
13    public function index()
14    {
15        //
16    }
17
18    public function delete()
19    {
20        //
21    }
22 }
23
24
```

Ο κώδικας που υλοποιήσαμε στην προηγούμενη ενότητα εντός του route είναι σχεδόν έτοιμος να χρησιμοποιηθεί πλέον στον Controller. Μετακινούμε λοιπόν τον κώδικα από το “web.php” στον Controller και τροποποιούμε τα routes κατάλληλα ούτως ώστε να μην χρησιμοποιούν πλέον ανώνυμες μεθόδους αλλά τις μεθόδους που ορίσαμε στον controller.

```
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Task;
7
8 class TasksController extends Controller
9 {
10    public function create(Request $request)
11    {
12        $newTask = new Task();
13
14        $newTask->name = $request->task;
15
16        $newTask->save();
17
18        return redirect('/');
19    }
20
21    public function index()
22    {
23        return view('index');
24    }
25
26    public function delete()
27    {
28        //
29    }
30 }
```

```

14 Route::get('/', 'TasksController@index');
15
16 Route::post('/task', 'TasksController@create');
17
18 Route::delete('/task/{task}', 'TasksController@delete');

```

Παρατηρούμε ότι στην κορυφή του Controller (γραμμές 5,6) γίνεται εισαγωγή των κλάσεων Request και Task έχοντας ως αποτέλεσμα να χρησιμοποιούνται πλέον μονολεκτικά σε αντίθεση με πριν (γραμμές 10,12).

Εφόσον πλέον μπορούμε πλέον να δημιουργούμε εγγραφές στον πίνακα της βάσης δεδομένων πρέπει να προχωρήσουμε στην υλοποίηση της προβολής τους. Η λογική της προβολής γίνεται στην μέθοδο “index” του controller μέσω τις οποίας οι επιστρεφόμενες εγγραφές από την βάση «περνούν» στο αντίστοιχο view όπου και υπάρχει η προβολή τους. Πιο συγκεκριμένα ο controller τροποποιείται ως:

```

21     public function index()
22     {
23         $tasks = Task::all();
24
25         return view('index')->with('tasks', $tasks);
26     }

```

Ειδικότερα στις γραμμές:

23. Query στην βάση δεδομένων για ΟΛΕΣ τις εγγραφές και όλα τα πεδία που υπάρχουν στον πίνακα “tasks” μέσω του αντίστοιχου model και καταχώρηση τους στην μεταβλητή “\$tasks”
25. Επιστροφή του κατάλληλου view μαζί με την μεταβλητή “\$tasks” για χρήση της στο view

Αφού πλέον το view γνωρίζει για την ύπαρξη μιας μεταβλητής “\$tasks” μπορούμε να προχωρήσουμε στην προβολή τους μέσω ενός βρόγχου με συντακτικό του Blade:

```

26 <table class="table table-striped">
27     <thead>
28         <tr>
29             <th>Name</th>
30             <th></th>
31         </tr>
32     </thead>
33     <tbody>
34         @foreach ($tasks as $task)
35         <tr>
36             <td>{{ $task->name }}</td>
37             <td class="text-right"><a href="#" class="btn btn-sm btn-danger">Delete</a></td>
38         </tr>
39         @endforeach
40     </tbody>
41 </table>

```

Αναλυτικότερα στις γραμμές:

34. Έναρξη του βρόγχου του Blade
36. Προβολή του attribute “name” του αντικειμένου “task”
39. Τέλος του βρόγχου

Το αποτέλεσμα των προηγούμενων τροποποιήσεων θα πρέπει να είναι παρόμοιο με το παρακάτω:

## 10.9 Διαγραφή των Tasks

Φυσικά για να είναι ολοκληρωμένη η εφαρμογή θα πρέπει να δίνεται και η δυνατότητα διαγραφής των tasks. Αυτό γίνεται μέσω του αντίστοιχου button που υπάρχει δίπλα από κάθε task. Η τεχνική λογική αυτής της ενέργειας είναι πως κάθε φορά που γίνεται κάποιο request στο URL /task/{task} (όπου {task}, το id του εκάστοτε που επιθυμούμε να διαγραφεί), καλείται η μέθοδος “delete()” του controller εντός της οποίας υπάρχει η λογική διαγραφής της εγγραφής από τον πίνακα.

Αρχικά θα ήταν καλό να δώσουμε ένα όνομα στην συγκεκριμένη διαδρομή. Αυτό θα μας βοηθήσει αργότερα στην κλήση της μέσα από το view. Τροποποιούμε κατάλληλα το αρχείο των routes:

```
18 Route::delete('/task/{task}', 'TasksController@delete')->name('task.delete');
```

Σε δεύτερη φάση πρέπει να τροποποιήσουμε το κομμάτι του view στο οποίο υπάρχει το κουμπί για την διαγραφή σύμφωνα με το παρακάτω:

```

33  <tbody>
34      @foreach ($tasks as $task)
35      <tr>
36          <td>{{ $task->name }}</td>
37          <td class="text-right">
38              <form action="{{ route('task.delete', $task->id) }}" method="POST">
39                  @method('DELETE')
40                  @csrf
41                  <button type="submit" class="btn btn-sm btn-danger">Delete</button>
42              </form>
43          </td>
44      </tr>
45  @endforeach
46 </tbody>

```

Αναλυτικότερα στις γραμμές:

38. Η διαγραφή γίνεται μέσω της καταχώρησης μιας φόρμας η οποία έχει σαν action το όνομα του route που δώσαμε προ ολίγου στην διαδρομή για την διαγραφή καθώς επίσης και μία παράμετρο που είναι το id του εκάστοτε task. Επί προσθέτως η μέθοδος της φόρμας δηλώνεται ως POST.
39. Επειδή έχουμε δηλώσει πως η μέθοδος με την οποία πρέπει να γίνει ένα αίτημα για διαγραφή είναι η DELETE, την δηλώνουμε εδώ μέσω του “Method Spoofing”. Έτσι παρακάμπτουμε την αρχική δήλωση πως η μέθοδος είναι POST.
40. CSRF token
41. Το κουμπί για την καταχώρηση της φόρμας

Πλέον είμαστε έτοιμοι να προχωρήσουμε και στην υλοποίηση της μεθόδου “delete()” στον controller:

```

28      public function delete(Task $task)
29      {
30          $task->delete();
31
32          return redirect('/');
33      }

```

Αναλυτικότερα στις γραμμές:

28. Πέρασμα παραμέτρου, αντικείμενο τύπου Task
30. Διαγραφή του αντικειμένου που περάστηκε ως παράμετρος (μέσω του Eloquent ORM, κεφ. 8)

## 10.10 Επίλογος

Το παραπάνω παράδειγμα αποτελεί την απόδειξη για την μεγάλη ευκολία και ταχύτητα που προσδίδει η Laravel στον προγραμματιστή διαδικτυακών εφαρμογών. Παρ' όλο που ήταν μια σχετικά απλή εφαρμογή χρησιμοποίησε αρκετά από τα βασικά χαρακτηριστικά και την λογική του framework, ούτως ώστε να αποτελέσει το κίνητρο

για κάποιον που θέλει να ασχοληθεί εκτενέστερα στο μέλλον. Φυσικά η Laravel δεν περιορίζεται μόνο σε αυτές τις πολύ απλές υλοποιήσεις αλλά μπορεί πολύ εύκολα να δημιουργήσει έργα μεγάλης κλίμακας σε σύντομο χρονικό διάστημα έχοντας πάντα σαν πρώτο στόχο την εύκολη ανάγνωση και ανάπτυξη του κώδικα της εφαρμογής.

Ο κώδικας του παραδείγματος βρίσκεται στο [GitHub](#).

## 11. LGthree – Εισαγωγή

### 11.1 Τι είναι το LGthree;

Το LGthree είναι ένα Laravel Web Application παροχής πληροφοριών για αεροδρόμια που βρίσκονται εντός των ορίων του ελληνικού FIR<sup>5</sup> καθώς και η παροχή χρήσιμων εργαλείων που δύνανται να βοηθήσουν τους πιλότους και τους ελεγκτές εναέριας κυκλοφορίας που χρησιμοποιούν τα αεροδρόμια αυτά. Αναπτύχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας και αποτελεί μια πιο επαγγελματική προσέγγιση για την εφαρμογή της Laravel καθώς και συναφών τεχνολογιών με σκοπό την εφαρμογή όλων όσων αναλύθηκαν στα προηγούμενα κεφάλαια.

### 11.2 Επισκόπηση Χαρακτηριστικών

Όπως προειπώθηκε το κύριο χαρακτηριστικό της εφαρμογής είναι η παροχή πληροφοριών σχετικά με τα ελληνικά αεροδρόμια. Αυτές οι πληροφορίες μπορεί να αφορούν ονομασίες, συντεταγμένες, διάδρομοι προσγείωσης-απογείωσης και άλλες πιο εξειδικευμένες πληροφορίες που θα αναφερθούν στην συνέχεια. Επί προσθέτως παρέχονται ορισμένα εργαλεία, όπως ο υπολογισμός του διαδρόμου προσγείωσης-απογείωσης εν χρήση ανάλογα με τις καιρικές συνθήκες που επικρατούν. Όλες αυτές οι παροχές υλοποιούνται με την χρήση μοντέρνων τεχνολογιών, αρχιτεκτονικής και σχεδιασμού με σκοπό την ευχάριστη εμπειρία χρήστη, την βέλτιστη απόδοση, την γρήγορη απόκριση αλλά και την δυνατότητα διόρθωσης bugs ή την προσθήκη νέων χαρακτηριστικών με σχετική ευκολία. Τέλος η εφαρμογή διαθέτει ένα λειτουργικό περιβάλλον διαχείρισης για την αλλαγή όλων των δεδομένων των αεροδρομίων.

### 11.3 Τεχνολογίες

Η εφαρμογή υλοποιεί αρκετές μοντέρνες τεχνολογίες που χρησιμοποιούνται κατά κόρον σήμερα στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών. Ορισμένες από πιο αξιοσημείωτες τεχνολογίες είναι οι εξής:

- **Laravel:** PHP Framework
- **MySQL:** Σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων
- **Vue.js:** JavaScript framework ανοιχτού κώδικα για την δημιουργία user interfaces.
- **jQuery:** Βιβλιοθήκη JavaScript
- **Sass:** Preprocessor scripting language για την διερμηνεία ή την μεταγλώττιση CSS

<sup>5</sup> Flight Information Region – Μια συγκεκριμένη περιοχή του εναέριου χώρου, στην οποία παρέχονται υπηρεσίες πληροφοριών πτήσεων. Η πιο συχνή διάρεση είναι ανά κράτος και στην συνέχεια εφαρμόζεται περαιτέρω τομεοποίηση.

- **npm:** JavaScript package manager. Αποτελεί το προεπιλεγμένο package manager για την γλώσσα προγραμματισμού Node.js
- **Bootstrap:** Front-end framework ανοιχτού κώδικα για τον σχεδιασμό ιστοσελίδων και διαδικτυακών εφαρμογών.
- **Webpack:** Static Module Bundler για μοντέρνες εφαρμογές
- **Pusher:** API για την μετάδοση ειδοποιήσεων πραγματικού χρόνου
- **Laravel Echo:** JavaScript βιβλιοθήκη για την παρακολούθηση γεγονότων. Είναι ειδικά σχεδιασμένη για τις εφαρμογές Laravel και αποτελεί μέρος του οικοσυστήματος της Laravel.

## 12. Χρήσιμες Ορολογίες

### 12.1 METAR

Οι Μετεωρολογικό σταθμοί σε όλο τον πλανήτη είναι επιφορτισμένοι με την παρατήρηση των καιρικών συνθηκών και την κοινοποίησή τους στους υπόλοιπους μετεωρολογικούς σταθμούς, έτσι ώστε όλοι οι ενδιαφερόμενοι να έχουν πρόσβαση στις καιρικές συνθήκες μιας περιοχής. Κάθε μετεωρολογικός σταθμός ενός αεροδρομίου οφείλει να εκδίδει τουλάχιστον δύο φορές την ώρα τα στοιχεία του καιρού κωδικοποιημένα σύμφωνα με τους διεθνείς κανόνες που έχουν θεσπιστεί για αυτό τον σκοπό (πρότυπα ICAO). Αυτά τα κωδικοποιημένα μηνύματα ονομάζονται METAR, ή όπως είναι η πλήρη ονομασία Meteorological Terminal Air Report.

Παρακάτω ακολουθεί ένα παράδειγμα METAR που περιέχει ορισμένα από τα πιο συνήθη και σημαντικά χαρακτηριστικά καθώς και η ερμηνεία του:

LGTS 181350Z 16013KT 6000 FEW030 BKN090 15/04 Q1011 NOSIG

- **LGTS:** Ο κωδικός ICAO του αεροδρομίου
- **181350:** Ημέρα και ώρα έκδοσης, 18 του τρέχοντος μήνα και 13:50 UTC
- **16013KT:** Άνεμος από 160° με ταχύτητα 4 κόμβοι
- **6000:** Ορατότητα 6 χιλιόμετρα
- **FEW030:** Επίπεδο νεφών FEW με βάση στα 3000 πόδια
- **BKN090:** Επίπεδο νεφών BROKEN με βάση στα 9000 πόδια
- **15/04:** Θερμοκρασία 15°C και σημείο δρόσου 4°C
- **Q1018:** Ατμοσφαιρική πίεση σε hPa, 1018 hPa. Ονομάζεται και QNH ή τοπική βαρομετρική
- **NOSIG:** Δεν αναμένεται κάποια σημαντική αλλαγή

Γενικώς τα συνήθη χαρακτηριστικά που απαρτίζουν ένα METAR είναι τα εξής:

- **Άνεμος:** Στην περίπτωση άπνοιας ο άνεμος δηλώνεται ως 00000KT, σε περίπτωση μεταβλητού ανέμου δίνονται οι ακραίες τιμές του (π.χ. 040V210), ενώ αν ο άνεμος είναι ριπαίος δίνεται η μέση ταχύτητα και η τιμή της μέγιστης ριπής (04013G35KT)
- **Ορατότητα:** Αν η ορατότητα είναι 10 χιλιόμετρα ή περισσότερο, αναγράφεται στο METAR ως 9999
- **Φαινόμενα:** Αν υπάρχουν καιρικά φαινόμενα, αυτά γράφονται μετά την ορατότητα σύμφωνα με τον κώδικα καιρού (π.χ RA σημαίνει βροχή).
- **Νέφη:** Τα νέφη δίνονται από το χαμηλότερο στρώμα προς το ψηλότερο. Αν δεν υπάρχουν νέφη κάτω από τα 5000 πόδια και η ορατότητα είναι μεγαλύτερη των

10 χιλιομέτρων τότε στο METAR αναγράφεται η λέξη CAVOK (Clouds and Visibility OK). Στον πίνακα

- **Θερμοκρασία και σημείο δρόσου:** Αν κάποια από τις δύο τιμές είναι αρνητική, τότε μπροστά από την θερμοκρασία αυτή αναγράφεται το γράμμα M(π.χ 02/M07).
- **QNH:** Δίνεται στρογγυλοποιημένο στην μικρότερη ακέραια τιμή
- **TREND:** Το τελευταίο κομμάτι είναι μια δίωρη πρόγνωση και μπορεί να είναι 3 λέξεις: NOSIG αν δεν αναμένεται σημαντική αλλαγή, BECMG για αναμενόμενη αλλαγή και TEMPO για παροδική αλλαγή. Οι δύο τελευταίες περιπτώσεις ακολουθούνται από το στοιχείο στο οποίο υπάρχει αλλαγή.

Πίνακας 3 "Όνομασίες κάλυψης ουρανού"

Κάλυψη Ουρανού σε όγδοα	Κάλυψη Ουρανού σε δέκατα	Όνομα	Συμβολισμός
0	0	Sky Clear	SKC
1	1	Few Clouds	FEW
2	2 με 3		
3	4	Scattered	SCT
4	5		
5	6		
6	7 με 8	Broken	BKN
7	9		
8	10	Overcast	OVC

## 12.2 TAF

Το TAF ή όπως αποτελεί αρκτικόλεξο του Terminal Aerodrome Forecast περιγράφει την πρόγνωση του καιρού για τον εκάστοτε μετεωρολογικό σταθμό και πιο συγκεκριμένα την πρόγνωση για τις καιρικές συνθήκες που αναμένεται να επικρατήσουν για μία περίοδο 24 ή 30 ωρών για μία περιοχή 5 στατικών μιλίων (8 χιλιομέτρων) πέριξ του αεροδρομίου.

Εκδίδεται συνήθως 4 φορές την ημέρα και η πρόγνωση που περιέχεται γίνεται από τους υπεύθυνους μετεωρολόγους μέσα από ένα πλήθος παραγόντων όπως τρέχουσες καιρικές συνθήκες, κατανόηση της διαδικασίας που επιδρά στην αλλαγή των καιρικών συνθηκών, εμπειρία, στατιστικά στοιχεία κ.α. Οι προγνώσεις αυτές περιέχουν όλες ή κάποιες από τις παρακάτω πληροφορίες:

- Στοιχεία ανέμου (ένταση, κατεύθυνση, ριπές)

- Ορατότητα
- Ποσότητα και τύπος υετού
- Νεφώσεις (τύπος, ποσότητα, κατακόρυφη έκταση)
- Αναταράξεις
- Ακραία καιρικά φαινόμενα όπως καταιγίδες, αμμοθύελλες κ.τ.λ.
- Αναμενόμενες αλλαγές στις συνθήκες της πρόγνωσης

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

Σχήμα 27 "Παράδειγμα πρόγνωσης TAF"

### **12.3 NOTAM**

Το NOTAM ή notice to airmen είναι ένα μήνυμα το οποίο περιέχει πληροφορίες σχετικά με τις εγκαταστάσεις, τις συνθήκες ή τις αλλαγές σε οποιαδήποτε αεροναυτική εγκατάσταση, υπηρεσία, διαδικασία. Επίσης μπορεί να περιλαμβάνει μηνύματα για ορισμένους κινδύνους οι οποίοι είναι σημαντικό να γνωστοποιηθούν άμεσα σε όσους εμπλέκονται με υπηρεσίες πτήσεων.

Τα NOTAM συνήθως εκδίδονται από εθνικές αρχές για μια πληθώρα λόγων όπως:

- Πιθανοί κίνδυνοι (π.χ εντοπισμός κοπαδιού πτηνών)
- Πτήσεις από σημαντικά άτομα (π.χ. Πρόεδροι κρατών)
- Κλείσιμο διαδρόμων, τροχοδρόμων κ.τ.λ.
- Μη λειτουργικά ραδιοβοηθήματα
- Στρατιωτικές ασκήσεις με αποτέλεσμα το κλείσιμο μέρους του εναέριου χώρου
- Μη λειτουργικά φώτα σε ψηλές κατασκευές ή στο αεροδρόμιο
- Προσωρινή τοποθέτηση εμποδίων πλησίον του αεροδρομίου (π.χ. γερανοί)

Τα NOTAM μεταδίδονται από την αρχή έκδοσης χρησιμοποιώντας τα γρηγορότερα διαθέσιμα μέσα σε όλους τους πιθανούς παραλήπτες.

### **12.4 Ενεργός Διάδρομος Προσγείωσης-Απογείωσης**

Για την επιλογή του ενεργού διαδρόμου προσγείωσης-απογείωσης λαμβάνονται υπ' όψης αρκετές παράμετροι με συνηθέστερες και καθοριστικότερες εξ' αυτών να είναι η κατεύθυνση και η ένταση του ανέμου. Η απόφαση για τον χαρακτηρισμό ενός διαδρόμου ως ενεργό σε ελεγχόμενα αεροδρόμια λαμβάνεται από τον ελεγκτή εναέριας

κυκλοφορίας και συγκεκριμένα τον ελεγκτή στην θέση TWR. Σε περίπτωση απουσίας λειτουργικού πύργου ελέγχου υπεύθυνος είναι ο πιλότος.

Άλλες παράμετροι που μπορούν να καθορίσουν την επιλογή ενός διαδρόμου προσγείωσης-απογείωσης ως «εν χρήση» είναι:

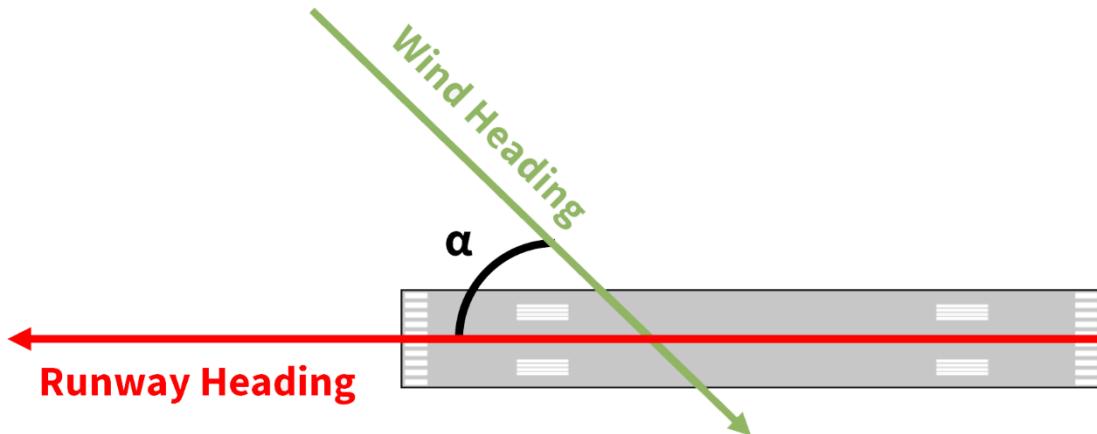
- **Επιδόσεις του αεροσκάφους:** Οι επιδόσεις του αεροσκάφους αποτελούν σημαντική παράμετρο για την επιλογή. Για παράδειγμα το μήκος ενός διαδρόμου μπορεί να μην είναι αρκετό για την ομαλή πέδηση του αεροσκάφους μετά την προσγείωση του.
- **Ροή της κίνησης:** Η ροή της κίνησης είναι επίσης ένας σημαντικός παράγοντας επιλογής διαδρόμου. Είναι πιθανόν ο ελεγκτής εναέριας κυκλοφορίας να χρησιμοποιεί περισσότερους από έναν διαδρόμους για την καλύτερη εξυπηρέτηση της κυκλοφορίας. Η συγκεκριμένη τεχνική είναι ιδιαίτερα δημοφιλής σε αεροδρόμια που έχουν περισσότερους από έναν διαδρόμους οι οποίοι είναι παράλληλοι μεταξύ τους. Για παράδειγμα, στο αεροδρόμιο «Ελευθέριος Βενιζέλος» των Αθηνών το οποίο αποτελείται από 2 διαδρόμους παράλληλους μεταξύ τους (03L/21R και 03R/21L) συνηθίζεται ο ένας διάδρομος να χρησιμοποιείται για προσγειώσεις ενώ ταυτόχρονα ο παράλληλος του χρησιμοποιείται για απογειώσεις βελτιώνοντας έτσι και την ροή της κυκλοφορίας και τον χρόνο εξυπηρέτησης.
- **Κανόνες του αεροδρομίου:** Κάθε αεροδρόμιο είναι πιθανόν να έχει κάποιους περιορισμούς και κάποιους κανόνες για την επιλογή του διαδρόμου που θα χρησιμοποιηθεί. Τέτοιοι περιορισμοί μπορούν να είναι τα λεγόμενα “noise abatement procedures”, τα οποία ουσιαστικά περιορίζουν την χρήση συγκεκριμένων διαδρόμων εντός ορισμένων χρονικών διαστημάτων. Αυτό συμβαίνει κυρίως όταν τα αεροσκάφη χρειάζονται να περάσουν πάνω από πόλεις οπότε και περιορίζεται η διέλευση τους για παράδειγμα κατά την διάρκεια της νύχτας. Άλλοι κανόνες που μπορούν να διέπουν ένα αεροδρόμιο είναι η επιλογή ενός διαδρόμου ως εν χρήση κατά την διάρκεια συγκεκριμένων καιρικών φαινομένων (π.χ. χαμηλή ορατότητα).

Για κάθε διάδρομο υπάρχει το αντίστοιχο wind component. Το wind component είναι ένας αριθμός που είναι ανάλογος της κατεύθυνσης του διαδρόμου, της κατεύθυνσης του ανέμου και της έντασης του. Η συνηθέστερη πρακτική είναι το αεροσκάφος να απογειώνεται και να προσγειώνεται έχοντας τον άνεμο όσο τον δυνατόν γίνεται μπροστά του, ή με πιο απλά λόγια έχοντας τον άνεμο «κόντρα». Αυτό συμβαίνει κυρίως για λόγους ασφαλείας, αφού στην μεν απογείωση ο άνεμος θα βοηθήσει το αεροσκάφος να σταματήσει γρηγορότερα σε περίπτωση εμφάνισης τεχνικού προβλήματος, στην δε προσγείωση ο άνεμος θα βοηθήσει την ελάττωση της ταχύτητας του αεροσκάφους.

Ο μαθηματικός υπολογισμός του wind component γίνεται μέσω του τύπου:

$$\text{Wind Component} = \text{Wind Speed} * \cos(a) \quad (11.1)$$

'Όπου:  $a = \text{Runway Heading} - \text{Wind Heading}$



Σχήμα 28 "Σχηματική απεικόνιση του wind component"

Στην περίπτωση που το wind component είναι θετικό τότε ο διάδρομος έχει μετωπικό άνεμο (headwind). Στην αντίθετη περίπτωση του αρνητικού αποτελέσματος ο διάδρομος έχει ούριο άνεμο (Tailwind). Καθίσταται λοιπόν σαφές πως ο διάδρομος που έχει το μεγαλύτερο wind component, συνεπώς και τον περισσότερο μετωπικό άνεμο, χρησιμοποιείται για τις προσγειώσεις και τις απογειώσεις.

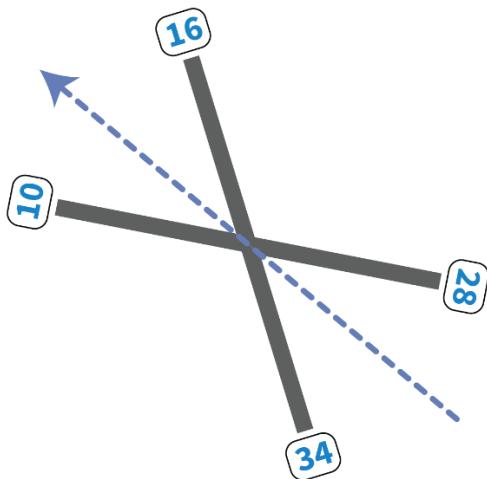
Παρακάτω ακολουθεί ένα παράδειγμα υπολογισμού του ενεργού διαδρόμου ενός αεροδρομίου. Για χάρη ευκολίας τα μοναδικά κριτήρια για την επιλογή είναι η κατεύθυνση και η ταχύτητα του ανέμου.

Έστω λοιπόν πως για το αεροδρόμιο της Θεσσαλονίκης ο άνεμος πνέει με κατεύθυνση  $140^\circ$  και ταχύτητα  $10\text{KT}$ . Είναι γνωστό πως το αεροδρόμιο έχει τέσσερις διαδρόμους προσγείωσης-απογείωσης που περιγράφονται στον ακόλουθο πίνακα<sup>6</sup>:

<sup>6</sup> Κατά την διάρκεια συγγραφής της εργασίας ο διάδρομος 10/28 είναι μη λειτουργικός λόγω εργασιών, παρ' όλα αυτά δεν λαμβάνεται υπ' όψη το γεγονός αυτό.

Πίνακας 4 "Διάδρομοι προσγείωσης-απογείωσης LGTS"

Αναγνωριστικό	Διεύθυνση
10	101°
16	163°
28	281°
34	343°



Σχήμα 29 "Σχηματική απεικόνιση διαδρόμων LGTS και ανέμου 140 μοιρών"

Υπολογίζεται το wind component για κάθε διάδρομο σύμφωνα με την σχέση 11.1:

$$WC_{10} = 10 * \cos(101 - 140) = 8$$

$$WC_{16} = 10 * \cos(163 - 140) = 9$$

$$WC_{28} = 10 * \cos(281 - 140) = -8$$

$$WC_{34} = 10 * \cos(343 - 140) = -9$$

Σύμφωνα λοιπόν με τους υπολογισμούς ο διάδρομος με το μεγαλύτερο wind component άρα και τον περισσότερο μετωπικό άνεμο είναι ο διάδρομος 16. Παρατηρείται επίσης πως οι διάδρομοι με αντίθετες διευθύνσεις όπως ο 16/34 έχουν ίδιο wind component κατ' απόλυτη τιμή αλλά με διαφορετικό πρόσημο. Το παραπάνω γεγονός είναι πολύ λογικό καθώς οι διευθύνσεις αυτών των διαδρόμων έχουν διαφορά 180°.

## 12.5 SID – STAR

Ο όρος SID (Standard Instrument Departure) είναι ένα συγκεκριμένο ATS Route<sup>7</sup> κατά την IFR<sup>8</sup> πλοϊγηση το οποίο ακολουθούν τα αεροσκάφη για να μεταβούν από την φάση της απογείωσης στην φάση της πορείας. Με τον όρο STAR (Standard Arrival Route) περιγράφεται ένα συγκεκριμένο ATS Route κατά την IFR πλοϊγηση κατά το οποίο τα αεροσκάφη μεταβαίνουν από την φάση της πορείας στο αρχικό σημείο προσέγγισης του αεροδρομίου.

Τα SIDs και τα STARs δημιουργούνται με σκοπό την επιτάχυνση της ασφαλούς και αποτελεσματικής ροής της εναέριας κυκλοφορίας που εκτελείται από και προς τους ίδιους ή διαφορετικούς διαδρόμους προσγείωσης-απογείωσης στο ίδιο ή σε γειτονικά αεροδρόμια. Αποσκοπούν στην αποσυμφόρηση πιθανώς αντικρουόμενης κυκλοφορίας με την χρήση συγκεκριμένων διαδρομών, επιπέδων, περιορισμών ταχύτητας και σημείων ελέγχου. Το πλήρωμα πτήσης συμμορφώνεται με τους δημοσιευμένους περιορισμούς εκτός και αν αυτοί οι περιορισμοί ακυρώνονται ή τροποποιούνται ρητώς από τον ελεγκτή εναέριας κυκλοφορίας.

Τυπικά, κάθε διάδρομος θα έχει έναν αριθμό από SIDs και STARs για να εξασφαλίσει ότι η εναέρια κυκλοφορία δεν καθυστερεί άσκοπα και ότι η ροή της κυκλοφορίας εκτελείται με ασφάλεια.

## 12.6 Charts (Χάρτες)

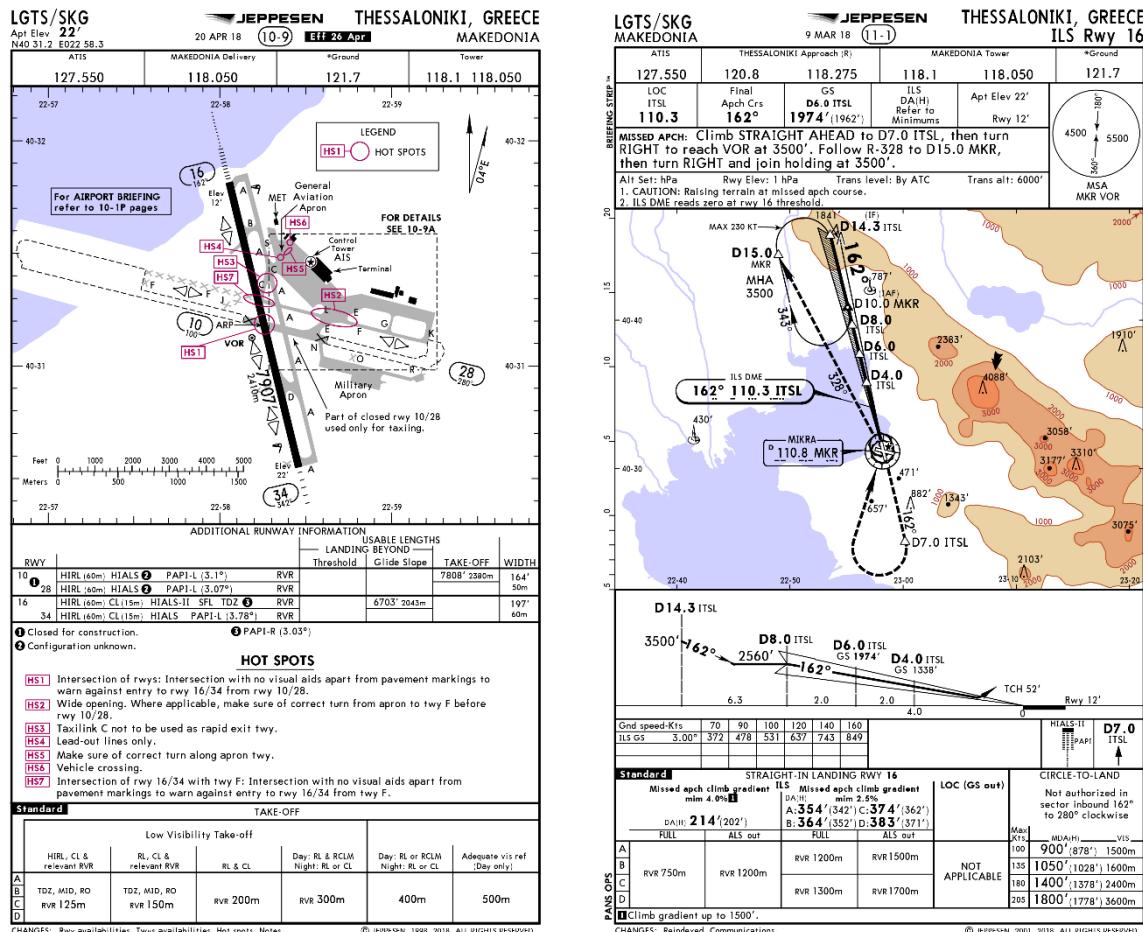
Οι χάρτες ή επίσημα AIP (Aeronautical Information Publication), είναι δημοσιεύσεις που εκδίδονται από την κρατική αρχή και παρέχουν αεροναυτικές πληροφορίες διαρκούς χαρακτήρα που είναι απαραίτητες στην αεροναυτιλία. Συγκεκριμένα περιέχουν λεπτομέρειες σχετικά με τους κανονισμούς, τις διαδικασίες και άλλες πληροφορίες σχετικά με τις ενέργειες των αεροσκαφών στην συγκεκριμένη χώρα όπου αναφέρονται. Εκδίδονται συνήθως από ή για λογαριασμό της αντίστοιχης διοίκησης πολιτική αεροπορίας και αποτελούν πηγή πληροφοριών για μόνιμη ενημέρωση και για προσωρινές αλλαγές μεγάλης διάρκειας.

Η δομή και τα περιεχόμενα των AIP τυποποιούνται από μια διεθνή συμφωνία μέσω του ICAO και συνήθως αποτελούνται από τρία μέρη: GEN (γενικά), ENR (καθ' οδόν – enroute) και AD (αεροδρόμια). Τα έγγραφα αυτά περιέχουν πλήθος χαρτών εκ των οποίων οι περισσότεροι βρίσκονται στο τμήμα AD όπου δημοσιεύονται λεπτομέρειες, χάρτες και γραφήματα.

<sup>7</sup> Μια συγκεκριμένη και δημοσιευμένη διαδρομή για την διοχέτευση της ροής της κυκλοφορίας

<sup>8</sup> Instrument Flight Rules – Η πλοϊγηση του αεροσκάφους με βάση τις πληροφορίες που λαμβάνει στα όργανα πλοϊγησής του και όχι με βάση την οπτική επαφή του πιλότου (VFR)

Τα AIP επικαιροποιούνται με τακτική αναθεώρηση σε σταθερό κύκλο. Για λειτουργικά σημαντικές αλλαγές πληροφοριών, χρησιμοποιείται ο κύκλος που είναι γνωστός ως AIRAC (Aeronautical Information and Control). Αναθεωρούνται σε κάθε νέο κύκλο AIRAC (28 ημέρες) ή μετά από 2 κύκλους AIRAC (56 ημέρες). Αυτές οι αλλαγές λαμβάνονται έγκαιρα ώστε οι χρήστες των αεροναυτικών δεδομένων να μπορούν να προχωρήσουν στις απαραίτητες τροποποιήσεις.



Σχήμα 30 "Παραδείγματα χαρτών για το αεροδρόμιο LGTS"

## 12.7 QNH

Το QNH είναι ένας κωδικός που προσδιορίζει την ατμοσφαιρική πίεση προσαρμοσμένη στο μέσο επίπεδο της θάλασσας (MSL – mean sea level). Πρόκειται για μια ρύθμιση πίεσης που χρησιμοποιείται από πιλότους, ελεγκτές εναέριας κυκλοφορίας και τους ραδιοφάρους καιρού η οποία όταν τίθεται στο υψόμετρο του αεροσκάφους προσαρμόζει το υψόμετρο στα όργανα του αεροσκάφους πάνω από το MSL για μια καθορισμένη περιοχή. Η μονάδα μέτρησης στην Ευρώπη είναι το hectopascal (hPa) ή αλλιώς millibar ενώ στις Ηνωμένες Πολιτείες μετριέται σε ίντσες υδραργύρου. Η

επίσημη ελληνική αναφορά στην ελληνική φρασεολογία γίνεται ως «τοπική βαρομετρική». Η standard τιμή του QNH είναι τα 1013.2 hPa ή 29.92inHg ή 1 atm.

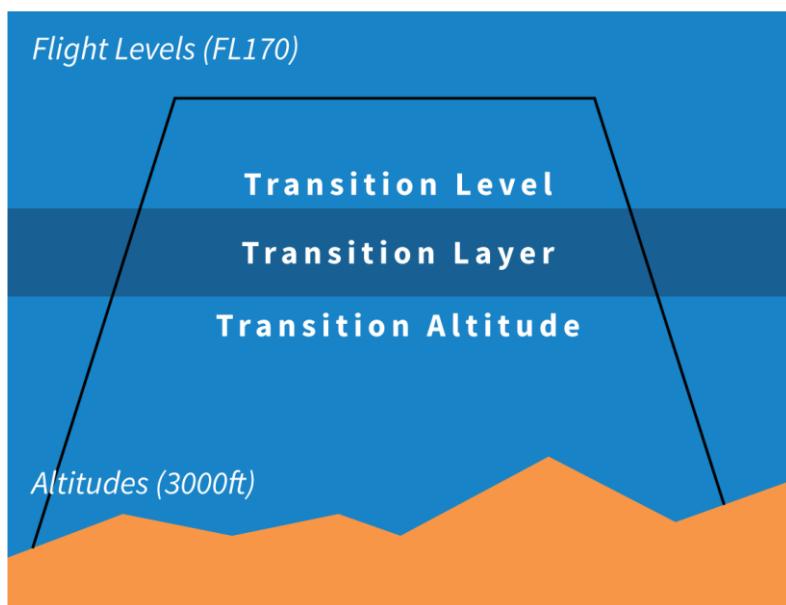
## 12.8 Transition Altitude – Transition Layer - Transition Level

Οι τρεις αυτοί όροι είναι εξαιρετικά μεγάλης σημασίας για την σωστή αναφορά των υψομέτρων των αεροσκαφών και συνεπώς την σωστή και προπάντων ασφαλή ροή της κυκλοφορίας.

Ο όρος transition altitude αναφέρεται το υψόμετρο στο οποίο από αυτό και κάτω η κάθετη θέση του αεροσκάφους ελέγχεται με αναφορά στο υψόμετρο δηλαδή σε πόδια π.χ. 3000ft.

Με τον όρο transition level εννοούμε το χαμηλότερο επίπεδο πτήσης (Flight Level) που είναι διαθέσιμο για χρήση και είναι πάνω από το transition altitude. Η αναφορά γίνεται σε επίπεδα πτήσης π.χ. FL170

Τρίτος και τελευταίος όρος είναι το transition layer το οποίο είναι ο εναέριος χώρος ανάμεσα στο transition altitude και το transition level.



Σχήμα 31 "Σχηματική απεικόνιση transition altitude, layer, level"

Στις Ηνωμένες Πολιτείες και τον Καναδά το transition altitude είναι καθορισμένο σταθερά στα 18000, σε αντίθεση με την Ευρώπη και σχεδόν το σύνολο του υπόλοιπου κόσμου όπου το transition altitude ποικίλει από αεροδρόμιο σε αεροδρόμιο. Και πάλι, είναι μια σταθερή τιμή η οποία δημοσιεύεται στην τεκμηρίωση του αεροδρομίου, στους χάρτες κ.τ.λ. Το transition level υπολογίζεται με βάση το τρέχων QNH ενώ λαμβάνεται υπ' όψη και το πάχος του transition layer που έχει καθοριστεί.

Για τον υπολογισμό του transition level απαιτούνται τα ακόλουθα βήματα:

1. Υπολογισμός transition altitude σε standard πίεση:

$$TA_{std} = TA + 28 (1013 - LocalQNH) \quad (11.2)$$

2. Υπολογισμός Equivalent Level:

$$EL = TA_{std}/100 \quad (11.3)$$

3. Υπολογισμός transition level ανάλογα με το πάχος του transition layer:

Πάχος	Τελευταίο Ψηφίο FL	Τύπος	Παράδειγμα
0ft – 599ft	0 ή 5	$EL \leq TL \leq EL + 5$	$FL53 \leq TL \leq FL58$ $\rightarrow TL = 55$
0ft – 999ft	0	$EL \leq TL \leq EL + 10$	$FL53 \leq TL \leq FL58$ $\rightarrow TL = 60$
1000ft – 1499ft	0 ή 5	$EL + 10 \leq TL \leq EL + 15$	$FL63 \leq TL \leq FL68$ $\rightarrow TL = 65$
1000ft – 1999ft	0	$EL + 10 \leq TL \leq EL + 20$	$FL63 \leq TL \leq FL73$ $\rightarrow TL = 70$

Παράδειγμα:

Έστω ότι QNH = 1019, TA = 5000ft, πάχος transition layer = 500ft

$$TA_{std} = 5000 + 28 (1013 - 1019) = 4832ft$$

$$EFL = \frac{4832}{100} = 48.32 = FL48$$

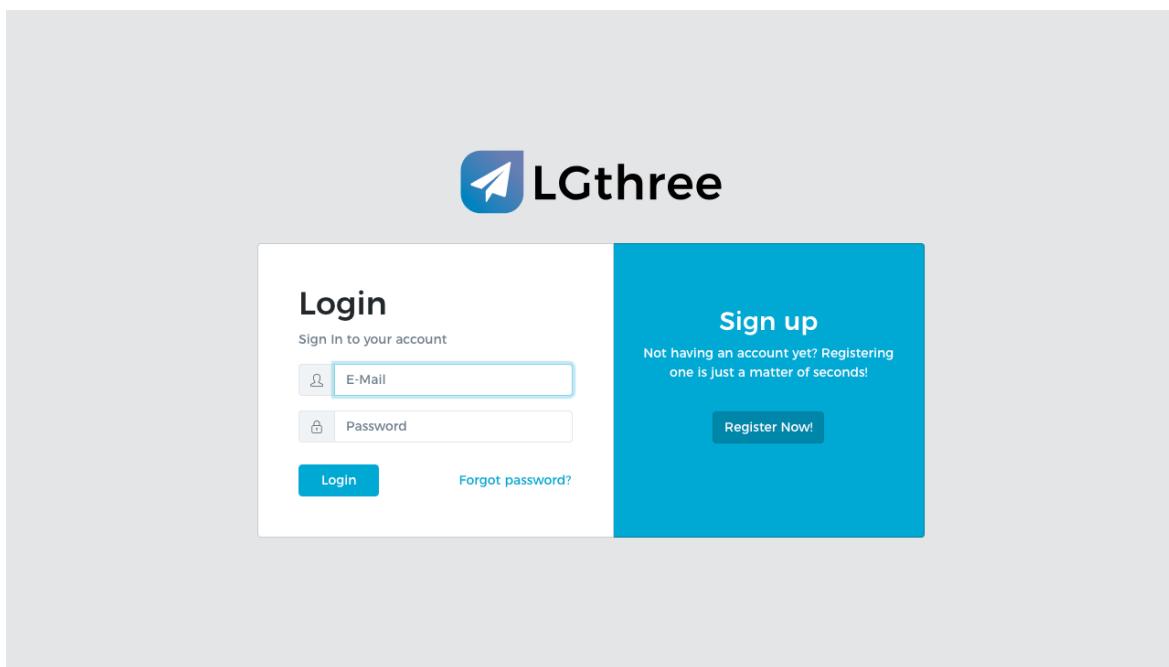
$$FL48 \leq TL \leq FL53 \rightarrow TL = FL50$$

## 13. Χαρακτηριστικά και Τεχνολογίες

### 13.1 Αυθεντικοποίηση

Για την πρόσβαση στην εφαρμογή απαραίτητη προϋπόθεση είναι η δημιουργία ενός λογαριασμού. Αυτό γίνεται στην σελίδα (/register) όπου ο χρήστης δημιουργεί έναν λογαριασμό με τα στοιχεία του. Μετά από την δημιουργία του λογαριασμού έχει πρόσβαση στην εφαρμογή. Επίσης δίνεται η δυνατότητα για επαναφορά κωδικού μέσω email.

Όλο το σύστημα αυθεντικοποίησης βασίζεται στο ενσωματωμένο σύστημα της Laravel χωρίς όμως τα προεπιλεγμένα views που έχουν σχεδιαστεί καινούρια. Επίσης όλα τα routes της εφαρμογής υλοποιούν ένα middleware το οποίο ελέγχει πριν κάθε request αν ο χρήστης είναι αυθεντικοποιημένος.



Σχήμα 32 "Η σελίδα σύνδεσης στην εφαρμογή"

### 13.2 Παροχή Πληροφοριών

Όπως ειπώθηκε σε προηγούμενες ενότητες ο βασικός σκοπός του LGthree είναι η παροχή πληροφοριών σχετικά με όλα τα αεροδρόμια που βρίσκονται εντός του ελληνικού εναερίου χώρου. Οι πληροφορίες αυτές περιέχουν μεταξύ άλλων τα γενικά στοιχεία του αεροδρομίου (ICAO code, όνομα, πόλη κ.α.), διαδρόμους προσγείωσης-απογείωσης, METAR, TAF, NOTAM, SID, STAR κ.τ.λ.

Για την προβολή αυτών των πληροφοριών ο χρήστης μπορεί να διαλέξει το αεροδρόμιο που επιθυμεί μέσα από έναν αριθμό επιλογών. Ευρισκόμενος στην αρχική σελίδα έχει

την δυνατότητα να πλοηγήθει στα διαθέσιμα αεροδρόμια μέσω του μενού που βρίσκεται στα αριστερά. Το μενού αυτό υπάρχει και στις υπόλοιπες σελίδες. Επί προσθέτως μπορεί να επιλέξει ένα αεροδρόμιο από τον χάρτη που εμφανίζεται ή να προχωρήσει σε γραπτή αναζήτηση του αεροδρομίου στο πεδίο που βρίσκεται πάνω από τον χάρτη. Η εφαρμογή προσπαθεί να αναζητήσει το δεδομένο του χρήστη στον κωδικό, το όνομα και την πόλη του αεροδρομίου. Επίσης είναι δυνατό ο χρήστης να έχει πρόσβαση σε αεροδρόμια που έχει επισημάνει ως αγαπημένα από το μενού που εμφανίζεται πατώντας τις τρείς μπάρες στην πάνω δεξιά γωνία.



Σχήμα 33 "Η αρχική σελίδα του LGthree"

Μετά την επιλογή κάποιου αεροδρομίου παρουσιάζονται στον χρήστη όλες οι πληροφορίες που υπάρχουν στην βάση δεδομένων για αυτό το αεροδρόμιο. Κάποιες πληροφορίες παρουσιάζονται στατικά π.χ. όνομα ενώ άλλες παρουσιάζονται και τροποποιούνται δυναμικά κατά το runtime της εφαρμογής π.χ. ενεργός διάδρομος. Οι πληροφορίες είναι χωρισμένες σε tabs για την ευκολότερη πλοήγηση τους. Για τις στατικές πληροφορίες η διαδικασία είναι πολύ απλή. Μετά το HTTP request στο URL της μορφής /airports/{icao} του χρήστη για κάποιο αεροδρόμιο ο router δρομολογεί το αίτημα στον ανάλογο controller και την ανάλογη μέθοδο η οποία ανακτά από την βάση δεδομένων όλες τις εγγραφές που σχετίζονται με αυτό το αεροδρόμιο καθώς και υπολογίζει τους ενεργούς διαδρόμους. Μετά την ανάκτηση των εγγραφών σε μία μεταβλητή η μέθοδος επιστρέφει το κατάλληλο view μαζί με την μεταβλητή. Τέλος στο view παρουσιάζονται όλες οι στατικές τιμές. Για την προβολή των δυναμικών τιμών θα γίνει αναφορά σε επόμενη ενότητα.

The screenshot shows a flight tracking interface for the airport LGTS (Makedonia, Thessaloniki). On the left is a sidebar with a list of airports: LGAX Alexandria, LGBL Volos (Nea Anchialos), LGHI Chios, LGKV Kavala, LGKZ Kozani, LGLM Lemnos, LGLR Larissa, LGMT Mytilene, LGSD Thessaloniki, LGSK Skiathos, LGSY Skyros, LCTG Tanagra, and LGTS Thessaloniki. The main panel displays the following details for LGTS:

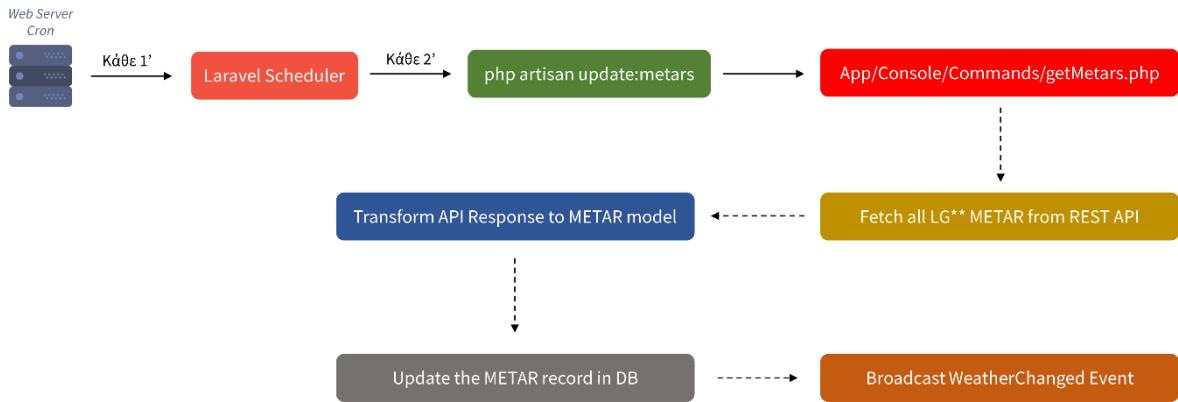
Details	
ICAO	LGTS
IATA	SKG
Name	Makedonia
City	Thessaloniki
Major	X
Elevation	22 ft.
Transition altitude	6000 ft.
Transition Level (Based on the QNH)	FL 75
MSA	8000 ft.
Latitude	40.519722
Longitude	22.870617

Σχήμα 34 "Στιγμιότυπο από την σελίδα για το αεροδρόμιο LGTS"

### 13.3 Προγραμματισμένες Ενημερώσεις

Ορισμένες από τις πληροφορίες των αεροδρομίων μεταβάλλονται διαρκώς κατά την διάρκεια της ημέρας π.χ. καιρικές συνθήκες. Η ανάκτηση αυτών των πληροφοριών γίνεται προγραμματιστικά μέσω του Laravel Scheduler, το οποίο όπως αναφέρθηκε σε προηγούμενη ενότητα είναι επιφορτισμένο να εκτελεί συγκεκριμένες εργασίες. Στην περίπτωση της εφαρμογής το Scheduler εκτελεί τρεις εντολές artisan: “php artisan update:metars”, “php artisan update:tafs” και “php artisan update:notams” ανά διαστήματα που έχουν οριστεί αναλόγως. Η λειτουργία της κάθε εντολής είναι ευνόητη λόγω της ονομασίας της. Η λογική υλοποίησης των εντολών αυτών είναι παρόμοια:

Αρχικά, γίνεται ένα αίτημα στο ανάλογο REST API το οποίο επιστρέφει κάποια αποτελέσματα. Τα αποτελέσματα αυτά επεξεργάζονται από την εφαρμογή και τροποποιούνται έτσι ώστε να αναπαριστούν ένα model instance της εφαρμογής. Μετά την δημιουργία των models αυτά αποθηκεύονται στην βάση δεδομένων στους ανάλογους πίνακες. Μετά και την επιτυχή αποθήκευση η εφαρμογή μεταδίδει ένα even προκειμένου να γίνει ακρόαση από τους πελάτες για λόγους που θα εξηγηθούν στην συνέχεια.



Σχήμα 35 "Απλουστευμένη σχηματική απεικόνιση της εντολής update:metars"

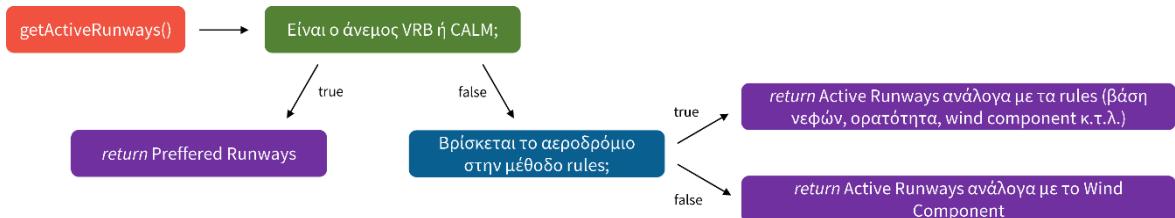
### 13.4 Επιλογή Ενεργού Διαδρόμου

Ο υπολογισμός και η επιλογή του ενεργού διαδρόμου προσγείωσης-απογείωσης γίνεται μέσω της μεθόδου `Airport::getActiveRunways()`, η οποία βρίσκεται στο trait `"app/Repositories/ActiveRunwayLogic.php"` και το οποίο χρησιμοποιείται από το model `Airport`. Η κλήση αυτής 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.

Στο σχήμα 36 απεικονίζεται η λογική υπολογισμού του διαδρόμου και συνεπώς ολόκληρου του trait στην υπεραπλουστευμένη μορφή της:

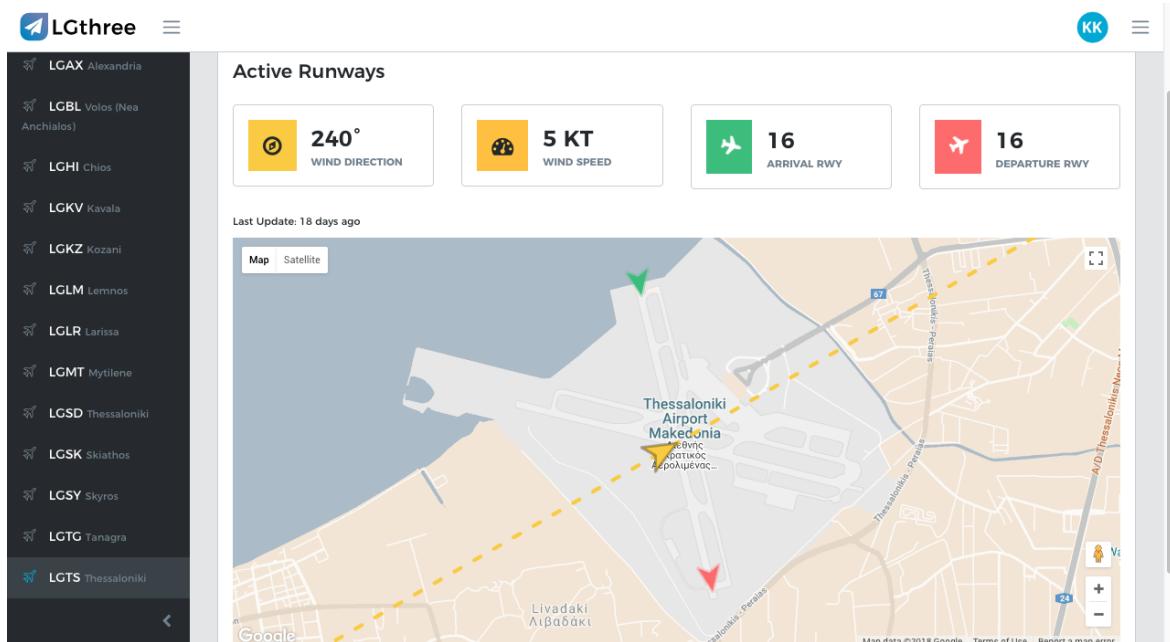


Σχήμα 36 "Σχηματική απεικόνιση του trait ActiveRunwayLogic"

Πρακτικά, κάθε φορά που καλείται η μέθοδος `getActiveRunways()`<sup>9</sup> για ένα αεροδρόμιο, η εφαρμογή ελέγχει εάν ο άνεμος είναι μεταβλητός ή επικρατεί άπνοια. Αν ικανοποιείται μία από τις δύο συνθήκες τότε σαν ενεργοί διάδρομοι ορίζονται εκείνοι που έχουν

<sup>9</sup> Η μέθοδος αποτελεί μέθοδο του **αντικειμένου** `Airport`, συνεπώς ακολουθείται αντικειμενοστραφή προσέγγιση όπως προτείνει η Laravel

οριστεί ως προεπιλεγμένοι. Σε διαφορετική περίπτωση η εφαρμογή ελέγχει αν το αεροδρόμιο βρίσκεται στην μέθοδο airportsRulesForActiveRunway του trait. Στην συγκεκριμένη μέθοδο έχουν οριστεί ορισμένοι περιορισμοί και κανόνες για διάφορα αεροδρόμια σχετικά με την επιλογή των ενεργών διαδρόμων. Τέτοιοι περιορισμοί και κανόνες μπορεί να είναι διάφορες νεφώσεις, τιμές ορατότητας, κατώφλι wind component, ώρες που εφαρμόζονται noise abatement procedures κ.α.. Εάν το αεροδρόμιο υπάρχει στην μέθοδο airportsRulesForActiveRunway τότε υπολογίζεται ανάλογα και ο ενεργός διάδρομος. Σε διαφορετική περίπτωση που το αεροδρόμιο δεν έχει δηλωθεί στην μέθοδο rules τότε η εφαρμογή υπολογίζει και επιστρέφει τους ενεργούς διαδρόμους βάση του wind component.



Σχήμα 37 "Στιγμιότυπο προβολής ενεργού διαδρόμου"

### 13.5 Ενημερώσεις Πραγματικού Χρόνου

Όπως προαναφέρθηκε η εφαρμογή εκτός από κάποιες στατικές πληροφορίες καλείται να διαχειριστεί και κάποια δυναμικά δεδομένα που μεταβάλλονται κατά την διάρκεια της ημέρας. Τέτοια δεδομένα είναι οι αλλαγές στις καιρικές συνθήκες που επικρατούν στα αεροδρόμια και πιο συγκεκριμένα το METAR του αεροδρομίου. Γίνεται λοιπόν εύκολα αντιληπτό πως οι καιρικές συνθήκες επηρεάζουν άμεσα την λειτουργία του αεροδρομίου για αυτό και θα πρέπει να παρέχονται στους χρήστες την στιγμή που η εφαρμογή τις έχει στην διάθεσή της.

Για την υλοποίηση αυτών των ενημερώσεων πραγματικού χρόνου οι οποίες όχι μόνο δεν απαιτούν τον χρήστη να ανανεώσει την σελίδα που βρίσκεται αλλά καθιστούν την άμεση προβολή τους χρησιμοποιείται ένας συνδυασμός τεχνολογιών τόσο στο backend

όσο και στο frontend της εφαρμογής. Η υλοποίηση θα μελετηθεί από το αρχικό χρονικό σημείο μέχρι και την προβολή των πληροφοριών στον πελάτη.

Αρχικά το Laravel Scheduler εκτελεί την εντολή “`php artisan update:metars`”. Όπως αναφέρθηκε στην ενότητα 12.3 η εντολή αυτή υλοποιείται στο αρχείο “`app\Console\Commands/getMetars.php`”. Το αρχείο αυτό θα ζητήσει από το REST API τα METAR όλων των ελληνικών αεροδρομίων. Υστερα θα κοιτάξει αν υπάρχει κάποια αλλαγή με το υπάρχον METAR συγκρίνοντας την τιμή του “`raw_text`”. Σε περίπτωση που υπάρχει αλλαγή η απάντηση του API θα μορφοποιηθεί κατάλληλα ώστε να αναπαριστά το model METAR και θα αποθηκευτεί στην βάση δεδομένων. Μετά και την επιτυχή αποθήκευση του METAR η Laravel δημιουργεί ένα νέο event με την ονομασία “`WeatherChanged`” το οποίο περιέχει ως παράμετρο το αεροδρόμιο για το οποίο υπήρξε η αλλαγή.

```
event(new WeatherChanged($airport));
```

Σχήμα 38 "Ο κώδικας δημιουργίας του event WeatherChanged"

Ο μηχανισμός της Laravel θα αντιστοιχίσει την δημιουργία του event με το αρχείο event που βρίσκεται στον κατάλογο “`app/Events/WeatherChanged.php`”. Το event αυτό έχει ως λειτουργία την μετάδοση του event και των δεδομένων του σε ένα κανάλι και πιο συγκεκριμένα στο κανάλι “`weather-tracker-airport_id`” το οποίο επικοινωνεί με το API του Pusher<sup>10</sup>. Μετά και από αυτή την μετάδοση η διαχείριση του event έχει φύγει από την κυριότητα του backend της εφαρμογής και πλέον υπεύθυνο για την μετάδοσή του στους ανάλογους clients είναι μόνο το Pusher.

Για την ακρόαση του event απαιτούνται κάποιες JavaScript βιβλιοθήκες. Αυτές είναι οι Laravel Echo και pusher-js. Μετά την εγκατάσταση των βιβλιοθηκών στην εφαρμογή το Laravel Echo γίνεται import στο αρχείο `/resources/assets/js/bootstrap.js` και ρυθμίζεται έτσι ώστε να ακούει όλες τις μεταδόσεις που γίνονται από το pusher και αφορούν την εφαρμογή.

```
import Echo from 'laravel-echo'
window.Pusher = require('pusher-js');

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
    encrypted: true
});
```

Σχήμα 39 "Η εισαγωγή και ρύθμιση του Laravel Echo"

<sup>10</sup> Για την μετάδοση του event μέσω ενός driver σε κάποιο εξωτερικό API και όχι στο εσωτερικό της εφαρμογής η κλάση του event πρέπει να κάνει implement το interface ShouldBroadcast.

Μετά την μετάδοση ενός γεγονότος από το pusher στους subscribers ενός καναλιού και την ακρόαση του γεγονότος από το Laravel Echo σειρά έχει η διαχείριση του event μέσω του JavaScript Framework Vue.js (<https://vuejs.org/>). Πριν γίνει η περιγραφή διαχείρισης των event πρέπει πρώτα να γίνει αναφορά σε κάποια βασικά στοιχεία του Vue.js.

Ένα μεγάλο πλεονέκτημα χρήσης του Vue είναι τα Vue Components, τα οποία είναι επαναχρησιμοποιούμενα κομμάτια κώδικα που περιέχουν τόσο γραφικό περιβάλλον στο tag <template> όσο και προγραμματιστική λογική στο tag <script>. Τα components βοηθούν στον κατακερματισμό του κώδικα σε μικρότερα κομμάτια και στην υλοποίηση και επεξεργασίων την πληροφοριών που περιέχουν μέσω του Vue.js. Φυσικά και τα πλεονεκτήματα χρήσης JavaScript frameworks αντί των «παραδοσιακών» βιβλιοθηκών είναι πάρα πολλά, όπως οργάνωση κώδικα και λογικής, ευανάγνωστο συντακτικό κ.λπ. Συγκεκριμένα η Laravel ενθαρρύνει την χρήση του Vue.js χωρίς αυτό να σημαίνει ότι δεν λειτουργεί το ίδιο καλά με άλλα frameworks όπως το Angular ή το React.

```
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' })
```

Σχήμα 40 "Παράδειγμα Vue Component"

Η εφαρμογή χρησιμοποιεί Vue Components για όλες τις πληροφορίες που ενημερώνονται δυναμικά είτε λόγω ενημέρωσης πραγματικού χρόνου π.χ. καρτέλα Active Runways είτε λόγω αλληλεπίδρασης με τον χρήστη π.χ. αναζήτηση αεροδρομίου στην αρχική σελίδα.

Ορισμένα Vue Components της εφαρμογής είναι επιφορτισμένα πέρα από την προβολή, επεξεργασία και μορφοποίηση των πληροφοριών και με το καθήκον της ακρόασης ορισμένων καναλιών μέσω του Laravel Echo. Κάθε φορά που υπάρχει κάποιο μήνυμα σε κάποιο κανάλι μέσω του Vue ενημερώνονται οι κατάλληλες μεταβλητές και συνεπώς τα κατάλληλα στοιχεία του DOM για την προβολή των νεών αυτών πληροφοριών στον

πελάτη. Η εφαρμογή αποτελείται από έναν ικανό αριθμό components αλλά για ευκολία θα παρουσιαστεί η διαδικασία προβολής του METAR ενός αεροδρομίου.

Στην καρτέλα Weather & NOTAM του αεροδρομίου παρουσιάζεται μεταξύ άλλων το METAR του αεροδρομίου την στιγμή που ο χρήστης έκανε το request. Αυτό δεν γίνεται με απ' ευθείας προβολή από την βάση δεδομένων (μέσω του Laravel Blade) αλλά με την χρήση ενός Vue Component. Στον αντίστοιχο αρχείο υπάρχει ένα custom tag “`<metar>`” το οποίο ουσιαστικά αναπαριστά το Vue Component που βρίσκεται στον κατάλογο “resources/assets/js/components/Metar.vue” και δέχεται μέσω του Laravel Blade σαν δεδομένα το id του αεροδρομίου και το raw\_text του METAR.

```
<metar
  :airport_id="{{ $airport->id }}"
  :raw_text="{{ $airport->metar->raw_text }}"
>
</metar>
```

Σχήμα 41 "Η κλήση του component Metar.vue"

Στο component Metar.vue υπάρχουν τα δύο tags για τα οποία έγινε αναφορά προηγουμένως. Στο tag `<template>` υπάρχει ο HTML κώδικας ο οποίος αναπαριστά το METAR του αεροδρομίου. Στο tag `<script>` συμβαίνει όμως όλη η προγραμματιστική λογική. Αρχικά το Vue αρχικοποιεί μια μεταβλητή η οποία περιέχει την τιμή του METAR που έχει περαστεί στο component ως δεδομένο. Η τιμή που έχει αυτή η μεταβλητή αποτυπώνεται στον HTML κώδικα, οπότε μπορεί να μεταβάλλεται κατά την εκτέλεση της εφαρμογής και να ενημερώνεται αντίστοιχα και ο HTML κώδικας χωρίς περαιτέρω ενέργειες. Μετά την αρχικοποίηση της μεταβλητής, μέσω του Laravel Echo το component ακούει για το γεγονός “WeatherChanged” στο κανάλι “weather-tracker-airport\_id”, όπου `airport_id` το id του εκάστοτε αεροδρομίου. Όταν λοιπόν μεταδοθεί κάποιο γεγονός το οποίο θα περιέχει την νέα τιμή του METAR, το component θα αλλάξει την τιμή της μεταβλητής με την νέα τιμή του METAR και θα ενημερώσει αμέσως το DOM. Όλη αυτή η διαδικασία ήταν μια ενημέρωση πραγματικού χρόνου για το METAR ενός αεροδρομίου.

Η εφαρμογή βασίζεται στα Vue Components για όλες τις ενημερώσεις πραγματικού χρόνου όπως οι ενεργοί διάδρομοι με αρκετά πιο πολύπλοκη όμως λογική και υλοποίηση.

```

<template>
<div class="row">
    <div class="col-12">

        <div class="card card-accent-primary mt-4">
            <div class="card-header">
                METAR
            </div>
            <div class="card-body">
                {{ rawTextNew }}
            </div>
        </div>

    </div>
</template>

<script>
export default {
    props: ['airport_id', 'raw_text', 'weather_station'],
    data() {
        return {
            rawTextNew: this.raw_text,
        }
    },
    mounted() {
        Echo.channel('weather-tracker.' + this.airport_id)
            .listen('WeatherChanged', (airport) => {
                this.rawTextNew = airport.metar.raw_text
            });
    }
}
</script>

```

Σχήμα 42 "Το Vue Component Metar.vue"

### 13.6 Περιβάλλον Διαχείρισης

Το LGthree διαθέτει ένα πλήρως λειτουργικό περιβάλλον διαχείρισης για την προσθήκη, τροποποίηση και διαγραφή εγγραφών από την βάση δεδομένων. Επίσης περιέχει κάποιες πρόσθετες δυνατότητες όπως διαχείριση χρηστών, δοκιμές καιρού κ.α. Μέσω του περιβάλλοντος διαχείρισης επιτυγχάνεται ο εύκολος χειρισμός των δεδομένων της εφαρμογής αφού πετυχαίνει να είναι εύχρηστο, απλό και κατανοητό σχετικά με την χρήση του.

Η υλοποίηση του δεν διαφέρει και πολύ από το frontend της εφαρμογής αφού χρησιμοποιεί εργαλεία όπως Laravel, Vue.js, MySQL κ.τ.λ. Για την πρόσβαση στο περιβάλλον διαχείρισης χρησιμοποιείται το middleware “admin” το οποίο ελέγχει σε κάθε request προς το URL “/admin/\*” ένα ο χρήστης είναι πρώτον αυθεντικοποιημένος και δεύτερον διαχειριστής.

Home

AIRPORTS

- All Airports
- Athens East
  - LGAV Athens**
  - LGEL Elefsina
  - LGIK Ikaria
  - LGIR Heraklion
  - LGKJ Kastelorizo
  - LGKN Marathon
  - LGKO Kos
  - LGKP Karpathos
  - LGKS Kasos
  - ...

Edit Airport [LGAV]

Point	Runway	Departure	Init Alt/FL	Release FL	Add SID
ABLON	21R	ABLON 1G	5000ft	FL160	<button>Edit</button> <button>Delete</button>
ABLON	03R	ABLON 1T	FL100	FL160	<button>Edit</button> <button>Delete</button>
ABLON	03L	ABLON 2D	FL100	FL160	<button>Edit</button> <button>Delete</button>
ABLON	21L	ABLON 3F	6000ft	FL160	<button>Edit</button> <button>Delete</button>
ASTOV	21L	ASTOV 1F	6000ft	FL160	<button>Edit</button> <button>Delete</button>
ASTOV	21R	ASTOV 1G	5000ft	FL160	<button>Edit</button> <button>Delete</button>
ASTOV	03R	ASTOV 1T	9000ft	FL160	<button>Edit</button> <button>Delete</button>
ASTOV	03L	ASTOV 2D	FL100	FL160	<button>Edit</button> <button>Delete</button>

Σχήμα 43 "Απόσπασμα από το περιβάλλον διαχείρισης"

## **Συμπεράσματα**

Η παρούσα πτυχιακή εργασία είχε ως σκοπό μια εισαγωγική παρουσίαση του -κατά πολλούς- ισχυρότερου framework για την γλώσσα της PHP και την προτροπή για περαιτέρω ενασχόληση από τους πιθανούς αναγνώστες. Η έκδοση με το κείμενο και πολύ περισσότερο οι αντίστοιχες διαφάνειες είναι δομημένες με τέτοιο τρόπο ούτως ώστε όχι μόνο να είναι απολύτως κατανοητές από το μεγαλύτερο μέρος των αναγνωστών-ακροατών αλλά και να τους παρουσιάσουν το πόσο εύκολη μπορεί να είναι η εκμάθηση αλλά και η ίδια η υλοποίηση στα πλαίσια μιας πραγματικής διαδικτυακής εφαρμογής. Άλλωστε είναι γνωστό πως στον προγραμματισμό καλύτερος γίνεσαι μόνο μέσα από την διαρκή ενασχόληση και την ανάπτυξη εφαρμογών και συστημάτων και όχι μόνο από βιβλιογραφικές πηγές. Τελικός σκοπός του πρώτου μέρους της εργασίας είναι η παρουσίαση της σε φοιτητές-επαγγελματίες-ερασιτέχνες σε μία μορφή σεμιναρίου-ενημέρωσης με σκοπό την περαιτέρω γνωστοποίηση της Laravel.

Επί προσθέτως η εφαρμογή LGthree αποσκοπεί σε δύο σκοπούς: Ο πρώτος της σκοπός είναι να «αποδείξει» κατά κάποιο τρόπο πως μια πολύπλοκη εφαρμογή μπορεί να υλοποιηθεί σε μεγάλο βαθμό από τα εισαγωγικά βήματα τα οποία παρουσιάστηκαν στο πλαίσιο της εργασίας. Ο δεύτερος σκοπός της εφαρμογής είναι η πιθανή χρήση της και στον πραγματικό κόσμο μέσα από μια ανάπτυξη που θα εξελίσσεται και θα βελτιώνεται διαρκώς. Παράδειγμα μιας τέτοιας χρήσης θα ήταν η χρήση της από τους πιλότους κατά την διάρκεια του pre-flight briefing, δηλαδή μιας διαδικασίας που γίνεται πριν από κάθε πτήση από το πλήρωμα του αεροσκάφους προκειμένου να μάθουν όλα τα απαραίτητα στοιχεία που θα τους φανούν χρήσιμα κατά την διάρκεια αυτής. Επιπλέον, με την συνεχώς αυξανόμενη χρήση των ασύρματων δικτύων και την έλευση του 5G στα επόμενα χρόνια, η εφαρμογή θα μπορούσε να αποτελεί εξέλιξη των υπαρχόντων εφαρμογών που χρησιμοποιούν οι αεροπορικές εταιρίες και περιέχουν χρήσιμες πληροφορίες σχετικά με τα αεροδρόμια, τα αεροσκάφη κ.τ.λ. αλλά όχι με δεδομένα τα οποία αλλάζουν κατά την διάρκεια της πτήσης και για τα οποία απαιτείται σύνδεση στο διαδίκτυο προκειμένου να ανακτηθούν.

## Βιβλιογραφία

- “11 Best PHP Frameworks for Modern Web Developers in 2018”, *coderseye.com*
- “Aeronautical Information Publications (AIPs)”, *skybrary.aero*
- “Bootstrap · The most popular HTML, CSS, and JS library in the world”, *getbootstrap.com*
- “Concepts”, *webpack.js.org*
- “Cross-Site Request Forgery (CSRF)”, *owasp.org*
- “Develop Faster with the Laravel PHP Framework”, *developer.com*
- Daniel Gafitescu (June 6, 2013), “Goodbye CodeIgniter, Hello Laravel”, *sitepoint.com*
- Erwan L’hotellier (October 1, 2015), “Headwind & Crosswind Calculation”, *ivao.aero*
- Maks Surguy (July 27, 2013), “History of Laravel PHP framework, Eloquence emerging”, *maxoffsky.com*
- “Introduction – Composer”, *getcomposer.org*
- Eric Barnes (January 30, 2015), “Laravel 5”, *laravel-news.com*
- “Laravel 5.6 Documentation”, *laravel.com*
- Martin Bean (April 2015), “Laravel 5 Essentials”, *books.google.com*
- “laravel/framework: Release v5.6.24”, *github.com*
- Jeffrey Way, “Learn practical, modern web development, through expert screencasts”, *laracasts.com*
- “Meteorological Terminal Air Report (METAR)”, *skybrary.aero*
- “Notice to Airmen (NOTAM)”, *skybrary.aero*
- “Pusher Documentation”, *pusher.com*
- “QNH, QFE, QNE and QFF explained!”, *forums.eagle.ru*
- “Sass: Syntactically awesome style sheets”, *sass-lang.com*
- “SIDs and STARs”, *skybrary.aero*
- “Transition Altitude/Level”, *skybrary.aero*
- Jeff Atwood (May 05, 2008), “Understanding Model-View-Controller”, *blog.codinghorror.com*

“Vue.js Documentation”, [vuejs.org](http://vuejs.org)

“vuejs/vue: Release v2.5.16”, [github.com](https://github.com/vuejs/vue)

“Weather Forecast”, [skybrary.aero](http://skybrary.aero)

Leff Avraham, Rayfield James T. (September 2001), “Web-Application Development Using the Model/View/Controller Design Pattern”, *IEEE Enterprise Distributed Object Computing Conference*

Jeffrey Way (November 28, 2012), “Why Laravel is Taking the PHP Community by Storm”, [code.tutsplus.com](http://code.tutsplus.com)