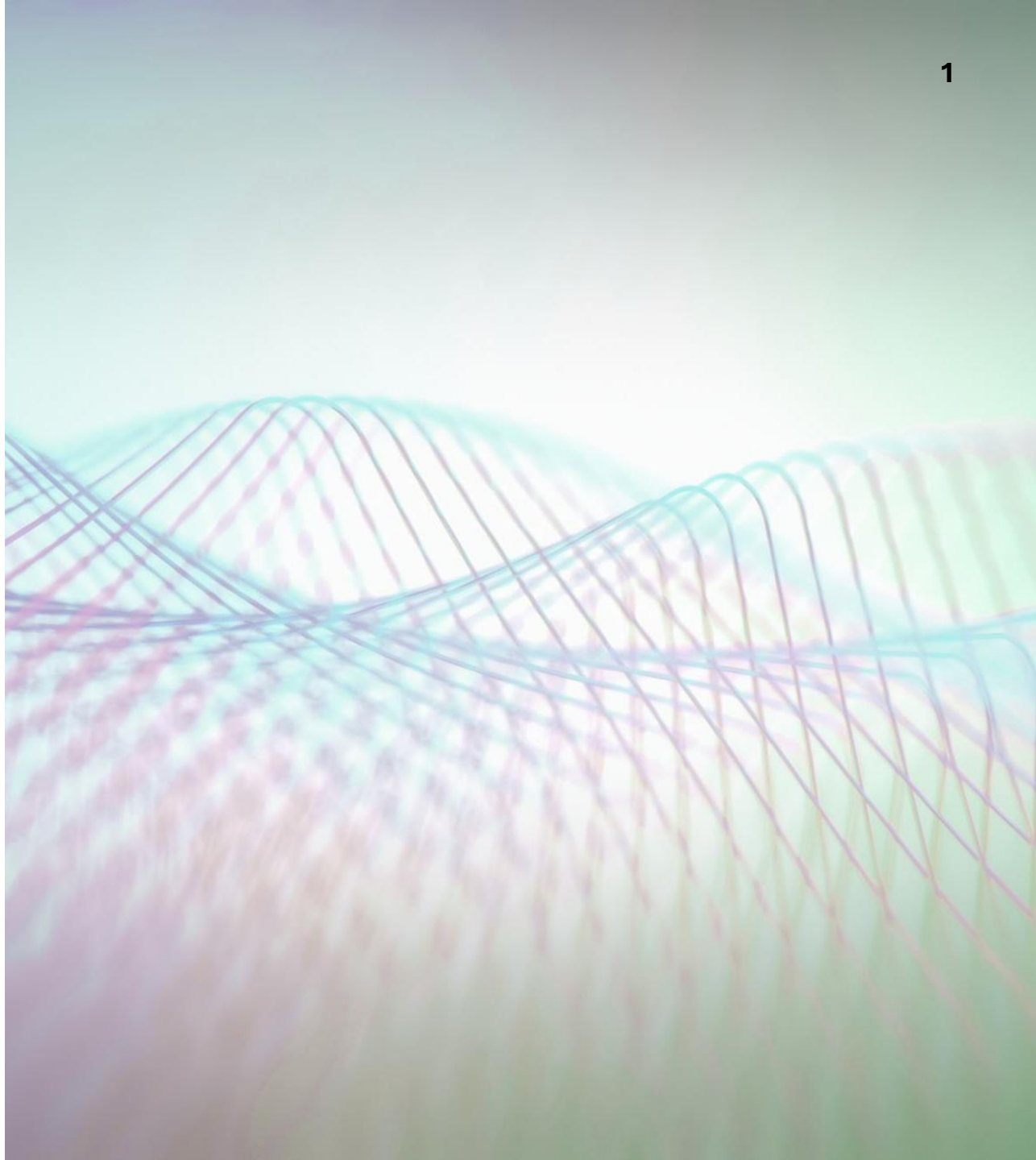
Τσουρδιού Γεώργιος – Χρήστος

Ανάπτυξη Διαδικτυακών Συστημάτων
& Εφαρμογών

Laravel PHP
React.js



Τι είναι Framework ;

είναι ουσιαστικά έτοιμα κομμάτια κώδικα που χρησιμοποιούμε για ευκολότερη ανάπτυξη λογισμικού.

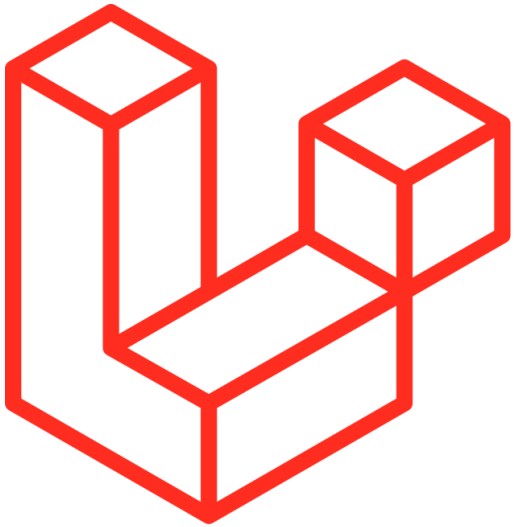
συνήθως χρησιμοποιεί πολλές γλώσσες για εσωτερικές λειτουργίες του.

Ένα framework

έχει κώδικα ο οποίος δεν μπορεί να μεταβληθεί, παραμόνο να επεκταθεί σε ορισμένες περιπτώσεις.

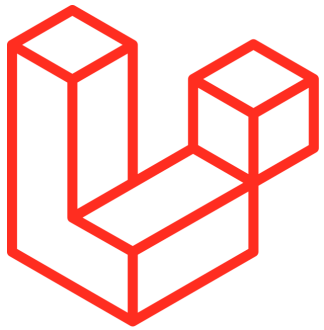
παρέχει συνήθως API’s για διάφορες λειτουργίες, επιπλέον βιβλιοθήκες κώδικα, compilers και υποστήριξη.

Τι είναι η Laravel ;

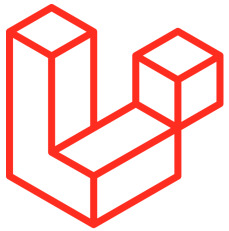


Η Laravel είναι ένα Full-Stack framework για Web-Development γραμμένο σε PHP.

Είναι ένα “progressive” και Scalable framework.



Γνωστά Projects που χρησιμοποιούν Laravel



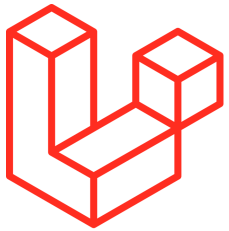
Η Laravel σαν Framework

https://laravel.com/

Η Laravel σαν framework, όπως προαναφέρθηκα είναι ένα scalable και progressive framework.

Progressive : Η Laravel με την εκτενή συλλογή από video tutorials, το εξαιρετικό documentation της και με την βοήθεια μίας μεγάλης κοινότητας που υποστηρίζει το συγκεκριμένο framework μπορεί να βοηθήσει έναν καινούργιο developer να ξεκινήσει αργά και με τα βασικά στοιχεία της. Επίσης παρέχει πιο advanced επιπέδου εργάλεία (για πιο έμπειρους προγραμματιστές) για unit testing (testing του κώδικα), queues (ουρές) και real-time events (γεγονότα σε πραγματικό χρόνο).

Scalable : Είτε η εφαρμογή που θές να υλοποιήσεις απευθύνεται σε μερικές χιλιάδες κόσμο είτε σε εκατομμύρια, η Laravel είναι ικανή να υποστηρίξει με μεγάλη άνεση όλες τις περιπτώσεις. Βασισμένη στην PHP η οποία είναι από μόνη της “optimized” για να διαχειρίζεται μεγάλο φόρτο, καθώς και με τη χρήση διαφόρων εργαλείων μπορεί να γίνει scale σχεδόν χωρίς όριο (σε serverless εφαρμογές).



MVC

Η Laravel ακολουθεί το δημοφιλές αρχιτεκτονικό μοτίβο λογισμικού MVC (Model-View-Controller).

Model

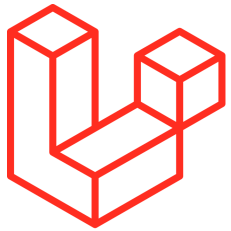
Είναι το βασικό κομμάτι του μοτίβου, καθώς και το δυναμικό κομμάτι του.
Στις περισσότερες περιπώσεις μπορεί να αντιπροσοπεύει δεδομένα από μία βάση δεδομένων.

Controller

Είναι το κομμάτι του μοτίβου που παίρνει κάποιο Model (καθώς και user input, αν υπάρχει), κάνει την απαραίτητη επεξεργασία με λογική που ορίζει ο προγραμματιστής και τα επιστρέφει ώς Response (JSON).

View

Είναι το κομμάτι της εφαρμογής που βλέπει ο χρήστης (GUI).
Θεωρείται γραφική αναπαράσταση ενός ή περισσοτέρων μοντέλων.



Το Front-End με την Laravel

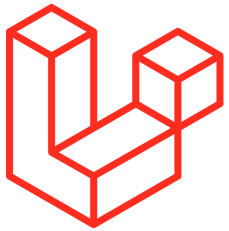
Blade.PHP

Μία ελαφριά Templating γλώσσα (όπως και η HTML), όπου σου επιτρέπει να ενσωματώσεις PHP (If's, Loops κτλπ) στην HTML.

Vue.js ή React.js

Στην περίπτωση που κάποιος δεν επιθυμεί να χρησιμοποιήσει την Blade για την δημιουργία του Front-End κομματιού, μπορεί να καταφύγει σε βιβλιοθήκες της JavaScript όπως η Vue.js ή η React.js. Υπάρχουν αρκετά εργαλεία για να διευκολύνουν την "γεφύρωση" του Back-End και του Front-End στην περίπτωση κάποιος χρησιμοποιήσει τις Vue.js ή React.js, όπως το Inertia

(συνήθως χωρίς να χρειάζεται API).



Composer

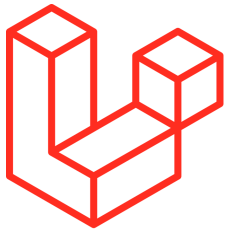
Ο composer είναι ένας dependency manager για την PHP.

Ο Composer χρησιμοποιείται από την Laravel για την διαχείριση των dependencies της, οπότε είναι απαραίτητο εργαλείο για την ανάπτυξη Project σε Laravel.

- Μπορεί να δημιουργήσει καινούργιο Laravel Project :

composer create-project laravel/laravel *app_name*.

- Μπορείς μέσω του Composer, να κάνεις import διάφορες βιβλιοθήκες κτλπ.



Artisan

https://laravel.com/docs/9.x/artisan

Εντολή του Artisan που εμφανίζει όλες τις διαθέσιμες εντολές.

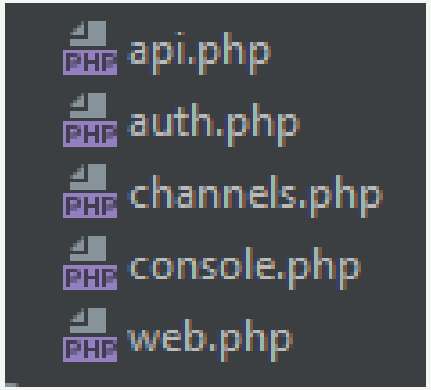
```
php artisan list
```

Η Laravel περιλαμβάνει ένα command line interface, το Artisan, το οποίο περιέχει πολλά scripts για την διευκόλυνση κάποιων διαδικασιών.

Οι cmd εντολές του Artisan, είναι της μορφής «**php artisan** *εντολή*».

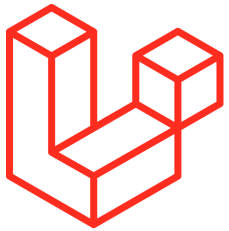
Routing

https://laravel.com/docs/9.x/routing



Καθώς η συγκεκριμένη φωτογραφία είναι από δικό μου Project, εμφανίζονται ακόμα 2 κλάσεις. Οι κλάσεις auth.php, console.php δεν θα μας απασχολήσουν σε αυτήν την παρουσίαση.

Η Laravel μας παρέχει 3 κλάσεις μέσα στις οποίες μπορούμε να δηλώσουμε τα Routes. Την api.php, web.php, channels.php. Κυρίως θα μας απασχολήσουν οι api.php και web.php, καθώς η τρίτη κλάση channels.php σχετίζεται με τα Broadcasting Channels.



Middleware

https://laravel.com/docs/9.x/middleware

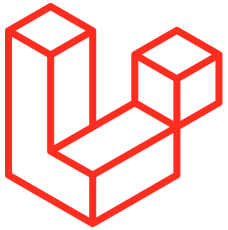
Ένα Middleware είναι στην ουσία μία κλάση στην οποία μπορούμε να ορίσουμε “φίλτρα”, μέσα από τα οποία θα περνάνε συγκεκριμένα HTTP Requests.
Μας βοηθάνε στην ομαδοποιήση των φίλτρων αυτών και την δήλωση τους σε κλάσεις με σκοπό την επαναχρησιμοποιήση. Μερικά από τα Middleware που χρησιμοποιεί η Laravel **είναι** :

- το Web middleware (όλα τα routes που είναι δηλωμένα στο web.php περνάνε από αυτό),
- το Api middleware (όλα τα routes που είναι δηλωμένα στο api.php περνάνε από αυτό) ,

Ο προγραμματιστής μπορεί να φτιάξει το δικό/α του Middleware με τη χρήση του Artisan με την εντολή

“**php artisan make : middleware** *όνομα_Middleware*”

Routing (2)



Η Laravel μας παρέχει αρκετές μεθόδους για να κάνει την δήλωση των routes πολύ εύκολη.

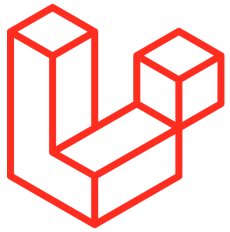
```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
```

Route ::

- post (“uri”,callback) για POST routes*,
- get (“uri”,callback) για GET routes*,
- delete (“uri”,callback) για DELETE routes*,
- patch (“uri”,callback) για PATCH routes* και
- put (“uri”,callback) για PUT routes*.

* Όπου “uri” το path του route και callback η ενέργεια που πρέπει να εκτελεστεί σε περίπτωση που γίνει request σε αυτό το route.

Routing (3)



Callbacks

(1)

```php
Route::get('/greeting', function () {
    return 'Hello World';
});
```

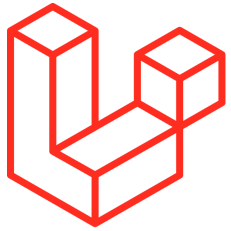
(2)

```php
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Σαν callbacks μπορούμε να χρησιμοποιήσουμε (1) ένα function που θα δηλώσουμε εκείνη τη στιγμή μέσα στο route (δεν συνιστάται διότι δεν μπορεί να ξαναχρησιμοποιηθεί αν δεν δηλωθεί ξανά).

Μπορούμε επίσης (2) να κάνουμε μία αναφορά σε ένα function που είναι δηλωμένο σε κάποιον controller, περνώντας σαν callback έναν πίνακα που περιέχει το όνομα του controller και το function που θέλουμε να χρησιμοποιήσουμε.

Routing (4)



User Input και Dependency Injection

(1)

```php
use Illuminate\Http\Request;

Route::get('/users', function (Request $request) {
    // ...
});
```

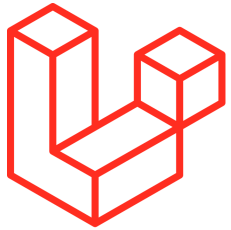
(2)

Στις περισσότερες περιπτώσεις χρειάζεται να περάσουμε με το request στον server και κάποια δεδομένα που συνήθως δίνονται από τον χρήστη.

Τα δεδομένα αυτά είναι αποθηκευμένα στην μεταβλητή $request.

Το μόνο που χρειάζεται να κάνουμε είναι να κάνουμε “import” την Κλάση Request΄ (όπως φαίνεται στην φωτογραφία (1)) και η Laravel αυτόματα κάνει inject την μεταβλητή αυτή στα callbacks ώστε να μπορείς να την χρησιμοποιήσεις (2).

Routing (5)



Named Routes, Groups

(1)

```php
Route::get('/user/profile', function () {
    //
})->name('profile');
```

(2)

```php
Route::middleware(['first', 'second'])->group(function () {
    Route::get('/', function () {
        // Uses first & second middleware...
    });

    Route::get('/user/profile', function () {
        // Uses first & second middleware...
    });
});
```

(3)

```php
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

(4)

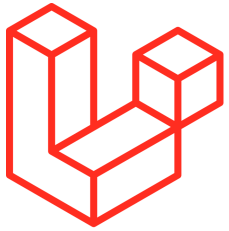
```php
use App\Http\Controllers\OrderController;

Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

(1) Για την πιο εύκολη αναγνώριση και χρήση των routes σε περίπτωση που έχουμε πάρα πολλά, μπορούμε να δώσουμε ένα μοναδικό όνομα στο κάθε route (1).

(2) Για καλύτερη οργάνωση των routes που σχετίζονται με κάποια συγκεκριμένη λειτουργία της εφαρμογής, μπορούμε να τα εντάξουμε σε groups (2).

Μπορούμε να δηλώσουμε middleware, από όπου όλα τα routes του group θα περάσουν (2), prefix (3), καθώς και έναν controller σε κάθε group (4).

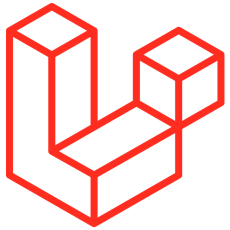
Laravel και Databases



https://laravel.com/docs/9.x/database

Η δημιουργία βάσεων, πινάκων, συσχετίσεων μεταξύ των πινάκων κτλπ, γίνεται «παιχνιδάκι» με την χρήση facades (interfaces) της Laravel.

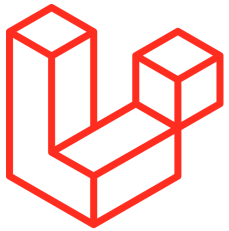
Πχ, με το Schema facade , μπορούμε να δημιουργήσουμε πίνακες, καθώς και να τους τροποποιήσουμε.



Σύνδεση σε μία βάση με την Laravel.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=
DB_USERNAME=
DB_PASSWORD=
```

Κατά την δημιουργία ενός Laravel project, δημιουργείται αυτόματα και ένα αρχείο που ονομάζεται .env. Είναι ένα πολύ σημαντικό αρχείο το οποίο περιέχει πολλές λεπτομέρειες για το project. Μερικές από αυτές είναι και οι συνδέσεις με τις όποιες βάσεις δεδομένων.
Στην εικόνα, βλέπουμε ένα παράδειγμα σύνδεσης με μία τοπική βάση mysql.



Migrations

https://laravel.com/docs/9.x/migrations

Τα Migrations είναι στην ουσία η δομή ενός πίνακα σε μία βάση. Ο κάθε πίνακας δημιουργείται μέσα σε μία κλάση Migration, όπου ορίζουμε τις στήλες του, τα κλειδιά του (κύρια και ξένα). Μπορούμε έτσι, πολύ εύκολα να τρέξουμε τα migrations σε οποιαδήποτε βάση θέλουμε και να πάρουμε την δομή που έχουμε δημιουργήσει, χωρίς να χρειαστεί να ξαναχτίσουμε την βάση από την αρχή (εξού και το όνομα τους Migrations, από το Migrate).

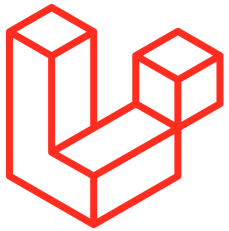
Μπορούμε να δημιουργήσουμε ένα νέο migration με το Artisan, με την χρήση της εντολής “php artisan : make migration *όνομα_Migration”*.

Migrations (2)

Μέσα στην κλάση του Migration, δημιουργούμε ένα Schema με την χρήση του Schema Facade της Laravel. Παίρνει σαν παράμετρο ένα αντικείμενο του τύπου Blueprint που είναι στην ουσία ο πίνακας που δημιουργούμε. Μπορούμε έπειτα να ορίσουμε στήλες και κλειδιά (κύρια και ξένα).

```php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->timestamps();
});
```

Το Schema facade κάνει expose μεθόδους που αντιστοιχούν σε τύπους δεδομένων που μπορού να αποθηκευτούν σε μία βάση. Όπως φαίνεται και στην εικόνα, id(), string(), timestamps() κτλπ.



Factories και Seeders

https://laravel.com/docs/9.x/seeding

Οι Seeders, είναι κλάσεις μέσα από τις οποίες μπορούμε να δημιουργήσουμε δεδομένα στην βάση

(συνήθως dummy data)

Μπορούμε να δημιουργήσουμε έναν Seeder με το Artisan, με την εντολή "php artisan make : seeder *όνομα_Model_Seeder*".

Τα Factories χρησιμοποιούνται από τους Seeders για την δημιουργία των Models

(δίνουν τιμές στις στήλες/attributes του Model).

Μπορούμε να δημιουργήσουμε ένα Factory με το Artisan, με την εντολή "php artisan make : factory *όνομα_Model_Factory*".

```php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => fake()->name(),
            'email' => fake()->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.
            'remember_token' => Str::random(10),
        ];
    }
}
```

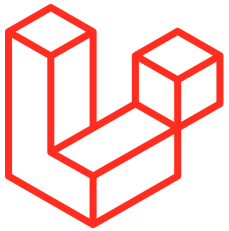
Factories

Στην εικόνα βλέπουμε ένα factory, για το Model User.

Όπως θα δούμε και αργότερα, όταν ένας Seeder καλεί ένα Factory για να δημιουργήσει ένα Model, στην ουσία καλείται το function definition. Αυτό το function δίνει σε ένα νέο Model, τις τιμές στα attributes του.

Με τη χρήση της μεθόδου fake(), μπορούμε να δώσουμε ψεύτικες τιμές στα πεδία του Model. Στην μέθοδο fake() γίνεται chain το είδος του πεδίου (πχ name,email,now κτλπ).

Τα πεδία στα οποία δίνουμε τιμές μέσω των factories θα πρέπει να αναφέρονται στον πίνακα “protected $fillable” του Model όπως θα δούμε και παρακάτω.

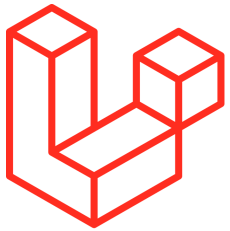


Seeders

```php
use App\Models\User;

/**
 * Run the database seeders.
 *
 * @return void
 */
public function run()
{
    User::factory()
            ->count(50)
            ->hasPosts(1)
            ->create();
}
```

Μία κλάση Seeder στην πιο απλή της μορφή, περιέχει το function run(), το οποίο καλεί το factory του αντίστοιχου model (μπορεί να καλέσει οποιοδήποτε factory). Μπορούμε να κάνουμε chain και επιπλέον μεθόδους, όπως για παράδειγμα το count, όπου ορίζουμε τον αριθμό των Models που θα δημιουργηθούν. Το τελευταίο chain που θα γίνει, πρέπει να είναι το create.

Seeding The Database

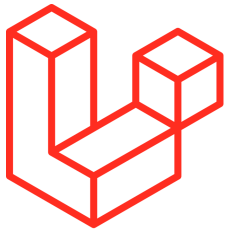


Εφόσων έχουμε δημιουργήσει τα factories που θέλουμε να χρησιμοποιήσουμε, καθώς και τα αντίστοιχα seeders που θα αξιοποιήσουν τα factories, μπορούμε να καλέσουμε στην κλάση (η οποία δημιουργείται αυτόματα με κάθε καινούργιο project) όλους τους seeders που έχουμε δημιουργήσει.

Με την χρήση του Artisan, μπορούμε να κάνουμε migrate και seed με μία εντολή, περνώντας μερικά options :

“php artisan migrate : fresh –seed”

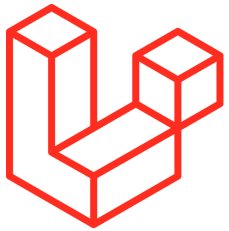
```php
/**
 * Run the database seeders.
 *
 * @return void
 */
public function run()
{
    $this->call([
        UserSeeder::class,
        PostSeeder::class,
        CommentSeeder::class,
    ]);
}
```



Eloquent ORM

https://laravel.com/docs/9.x/eloquent

Όπως και με το Artisan, η Laravel περιλαμβάνει και το Eloquent. Το Eloquent είναι ένας Object-relational mapper, ο οποίος κάνει τις “συναλλαγές” της εφαρμογής με βάσεις δεδομένων ευκολότερες (στις περισσότερες φορές δεν χρειάζεται να γράψεις ούτε μία γραμμή sql). Με τον όρο Object-relational mapper, εννοούμε πώς ο κάθε πίνακας (migration) από μία βάση, αντιστοιχεί σε ένα Model στην εφαρμογή μας. Μέσω του model, μπορούμε να κάνουμε select, insert, delete και update δεδομένα από την βάση.



Eloquent Models

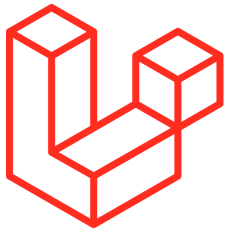
Για την δημιουργία ενός Eloquent model, μπορούμε να χρησιμοποιήσουμε το Artisan με την εντολή

“php artisan make : model *όνομα_Model*”.

Υπάρχουν κάποιοι “**Κανόνες/Συμβάσεις**” που πρέπει να ακολουθούμε έτσι ώστε η Laravel να μπορέσει να λειτουργήσει σωστά και να μήν χρειαστεί παραπάνω δουλειά από τον προγραμματιστή. Σχετικά με τα Models :

- Το όνομα του Model πρέπει να είναι τέτοιο έτσι ώστε το όνομα του πίνακα (migration) στον οποίο αναφέρεται να είναι ο πληθυντικός σε “snake case”. (πχ αν το migration μας έχει το όνομα users_table τότε το αντίστοιχο Model θα πρέπει να είναι User).
- Η Laravel όταν φτιάχνουμε τα migrations, υποθέτει πώς το κύριο κλειδί είναι μία στήλη στον πίνακα που ονομάζεται ID. Σε αντίθετη περίπτωση πρέπει να δηλωθεί στο αντίστοιχο Model με την μεταβλητή

“protected $primaryKey”



Eloquent Models (2)

$fillable

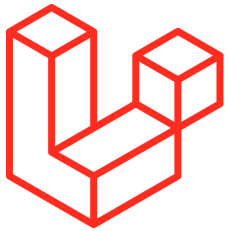
Η μεταβλητή $fillable μέσα σε μία Model Class, περιέχει τα πεδία (attributes) της κλάσης, τα οποία μπορούν να πάρουν τιμή από τον χρήστη. Αυτό γίνεται για λόγους ασφαλείας.

(Mass Assignment Vulnerability)

$hidden

Η μεταβλητή $hidden μέσα σε μία Model Class, περιέχει τα πεδία (attributes) της κλάσης, τα οποία δεν συμπεριλαμβάνονται στο JSON αντικείμενο (που αντιπροσωπεύει ένα αντικείμενο αυτής της κλάσης) που επιστρέφεται από ένα request.

Ένα καλό παράδειγμα είναι, ο κωδικός και το token ενός χρήστη, όταν ζητάμε μέσω ενός αιτήματος get να μας επιστρέψει έναν user.



Select με το Eloquent

Με το Eloquent μπορούμε να φτιάξουμε δυναμικά queries με την χρήση των μεθόδων που μας παρέχει.

```php
use App\Models\Flight;

foreach (Flight::all() as $flight) {
    echo $flight->name;
}
```

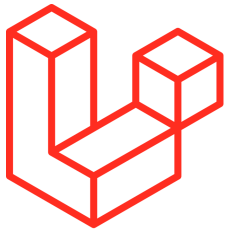
Η μέθοδος all() του Model, επιστρέφει όλες τις γραμμές του αντίστοιχου πίνακα.

Στην ουσία “Select * From Table”

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

Στην περίπτωση που θέλουμε να βάλουμε conditions, μπορούμε να χρησιμοποιήσουμε την μέθοδο where() (μπορούμε να κάνουμε chain πολλές where συνθήκες με την χρήση της orWhere ή της where(), χωρίς OR πολλαπλές chained where() λαμβάνονται πως έχουν το AND ανάμεσα τους).

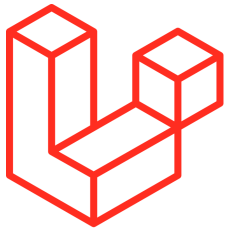
Insert με το Eloquent



Για να κάνουμε Insert σε κάποιον πίνακα με το Eloquent, αρκεί να δημιουργήσουμε ένα νέο αντικείμενο του Model του αντίστοιχου πίνακα, να περάσουμε τα attributes και να το κάνουμε save().

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

    $flight = new Flight;

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

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

Update με το Eloquent

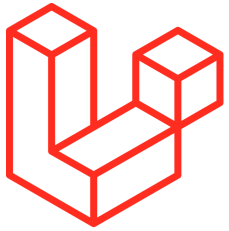


Με σχεδόν τον ίδιο τρόπο όπως και με το Insert, μπορούμε να κάνουμε update σε μία γραμμή του εκάστοτε πίνακα. Με την μέθοδο find του Model μπορούμε να πάρουμε **μία** γραμμή περνώντας σαν παράμετρο μία τιμή του κύριου κλειδιού (στις περισσότερες φορές το ID). Στο Model που θα μας επιστρέψει, απλά αλλάζουμε την τιμή του attribute που επιθυμούμε και ξανακάνουμε save().

Προσοχή, στην περίπτωση που η μέθοδος find() δεν λάβει σαν παράμετρο κάποιον ακέραιο (int), δεν επιστρέφει μία γραμμή, αλλά ένα Collection (Θα αναφερθώ αργότερα).

```php
use App\Models\Flight;

$flight = Flight::find(1);

$flight->name = 'Paris to London';

$flight->save();
```

Delete με το Eloquent

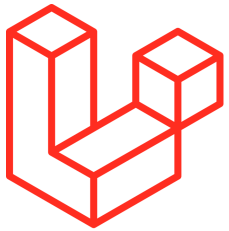


Για να κάνουμε Delete γραμμές από κάποιον πίνακα, αρκεί να χρησιμοποιήσουμε κάποια από τις μεθόδος που προανέφερα για να μας επιστρέψουν τις γραμμές που θέλουμε να διαγράψουμε και να κάνουμε chain την μέθοδο delete(). Μπορούμε επίσης να χρησιμοποιήσουμε την μέθοδο destroy() για να διαγράψουμε μία ή περισσότερες γραμμές απλά με την χρήση του κύριου κλειδιού.

Προσοχή, στην περίπτωση που η μέθοδος find() δεν λάβει σαν παράμετρο κάποιον ακέραιο (int), δεν επιστρέφει μία γραμμή, αλλά ένα Collection (Θα αναφερθώ αργότερα).

```php
use App\Models\Flight;

$flight = Flight::find(1);

$flight->delete();
```

```php
Flight::destroy(1);

Flight::destroy(1, 2, 3);

Flight::destroy([1, 2, 3]);
```



Relationships

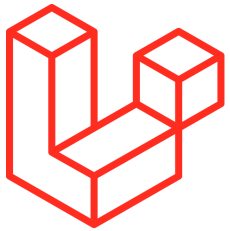
https://laravel.com/docs/9.x/eloquent-relationships

Το Eloquent μας διευκολύνει ιδιαίτερα με τις σχέσεις μεταξύ των πινάκων στην βάση μας.

Το Eloquent υποστηρίζει τις παρακάτω σχέσεις.

- One To One
- One To Many
- Many To Many
- Has One Through
- Has Many Through
- One To One (Polymorphic)
- One To Many (Polymorphic)
- Many To Many (Polymorphic)

Relationships (2)

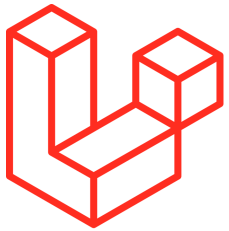


Οι σχέσεις μεταξύ 2 ή περισσότερων Models, γίνονται με την χρήση functions στις αντίστοιχες αυτές κλάσεις.

Για να μπορέσει η Laravel να ερμηνεύσει σωστά αυτές τις σχέσεις, όπως είπαμε και σε προηγούμενες διαφάνειες πρέπει να τηρήσουμε κάποιους κανόνες (σε αντίθετη περίπτωση, πρέπει σε κάθε μία από τις παρακάτω functions, να περάσουμε και τα προαιρετικά parameters για να δείξουμε τα κύρια και τα ξένα των πινάκων που αναφερόμαστε.

```php
class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone()
    {
        return $this->hasOne(Phone::class);
    }
}
```

```php
return $this->hasOne(Phone::class, 'foreign_key', 'local_key');
```



One to One

Από τις πιο βασικές σχέσεις μεταξύ 2 πινάκων είναι η « Ένα προς Ένα».

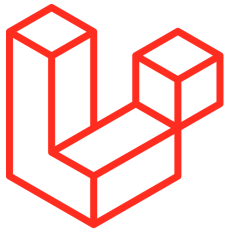
Σε μία εφαρμογή συνεργείου αυτοκινήτων, ένας πελάτης (Client) μπορεί να συνδέεται μόνο με ένα αυτοκίνητο (Car).

Στην κλάση Client :

```
public function Car() {
    return $this->hasOne(Car::class);
}
```

Στην κλάση Car :

```
public function Owner () {
    return $this->belongsTo(Client::class);
}
```



One to Many

Μία ακόμα πολύ συχνή σχέση μεταξύ 2 πινάκων είναι η « Ένα προς Πολλά».

Σε μία εφαρμογή σαν το Facebook, ένας Post μπορεί να έχει Comments.

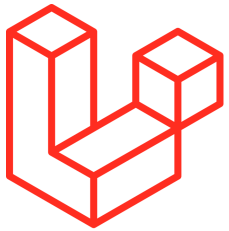
Η σχέση του Post με τα Comments του αντιπροσωπεύει μία τέτοιου είδους σχέση.

Στην κλάση Post :

```
public function Post() {
    return $this->hasMany(Comment::class);
}
```

Στην κλάση Car :

```
public function Comments () {
    return $this->belongsTo(Post::class);
}
```



One To One (Polymorphic)

Καθώς εμβαθύνουμε λίγο στις σχέσεις, μπορούμε να δούμε πώς μερικές φορές οι σχέσεις μεταξύ πινάκων μπορούν να γίνουν λίγο πιο περίπλοκες.

Για παράδειγμα, όπως και στο παράδειγμα με το Facebook, μία εικόνα μπορεί να ανήκει σε κάποιο post, είτε να ανήκει σε κάποιον χρήστη

(εικόνα προφίλ πχ).

Εδώ η εικόνα εμπλέκεται σε μία πολυμορφική σχέση όσον αφορά τον “ιδιοκτήτη” της.

Στην μία περίπτωση θα είναι ένα post, σε μία άλλη θα είναι ένας χρήστης.

Posts

id - integer

name - string

Users

id - integer

name - string

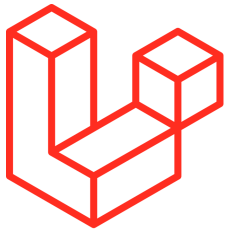
Images

id - integer

url - string

imageable_id - integer

imageable_type - string



One To One (Polymorphic) (2)

Στην κλάση Post :

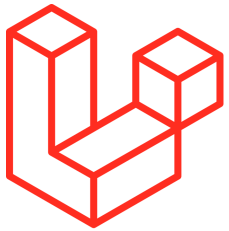
```
public function Image() {
  return $this->morphOne (Image::class, 'imageable');
}
```

Στην κλάση User :

```
public function Image(){
    return $this->morphOne (Image::class, 'imageable');
}
```

Στην κλάση User :

```
public function imageable() {
    return $this->morphTo();
}
```



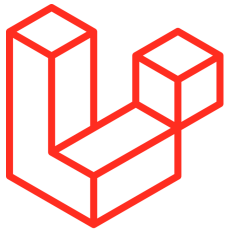
Retrieving The Relationships

1. `$owner = Car::find(x) -> Owner;` (Όπου x το id του αμαξιού)
2. `$post_title = Comment::find(x) ->post->title;` (Όπου x το id του comment)

Στις πολυμορφικές σχέσεις, ισχύει η (1) καθώς και

3. `$image_owner = Image::find(x) ->imageable;` (Όπου x το id της εικόνας , σε αυτήν την

περίπτωση ο image_owner θα μπορεί να είναι είτε τύπου User, είτε τύπου Post, αναλόγως με το τι έχουμε δηλώσει)



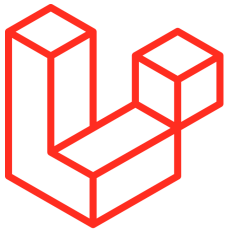
Eloquent Collections

Τα Eloquent Collections είναι συλλογές (όπως αναφέρει και το όνομά τους) από Models. Στα Queries που είδαμε και στις προηγούμενες διαφάνειες, όπως για παράδειγμα το all(), το find() (σε περίπτωση που δεν είναι int η παράμετρος που θα περάσουμε) επιστρέφουν collections.

Η Laravel μας παρέχει αρκετές μεθόδους για την διαχείριση των collections

https://laravel.com/docs/9.x/eloquent-collections#available-methods

https://laravel.com/docs/9.x/collections#available-methods



Μερικά ακόμα Features της Laravel

Απλή αναφορά μερικών ακόμα δυνατοτήτων της Laravel.

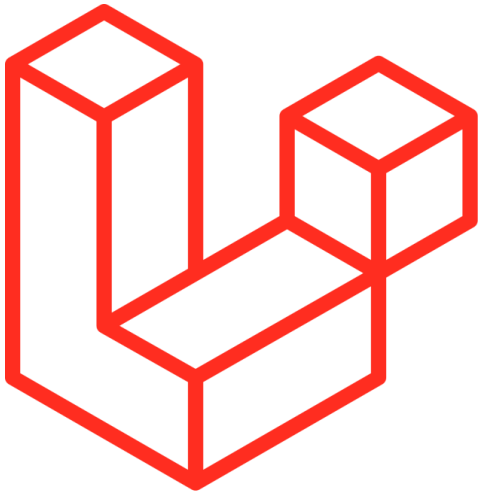
Δεν θα γίνει περαιτέρω εξήγηση των παρακάτω, διότι είναι πιο advanced λειτουργίες και δεν αφορούν την σημερινή παρουσίαση.

1. Built-In Authentication System
2. Authorization
3. Encryption
4. Testing
5. Real-Time Events (με χρήση Web-Sockets)
6. Mail

Προφανώς η Laravel έχει ακόμα αμέτρητες εφαρμογές και δυνατότητες, μπορείτε να τις εξερευνήσετε και μόνοι σας στο

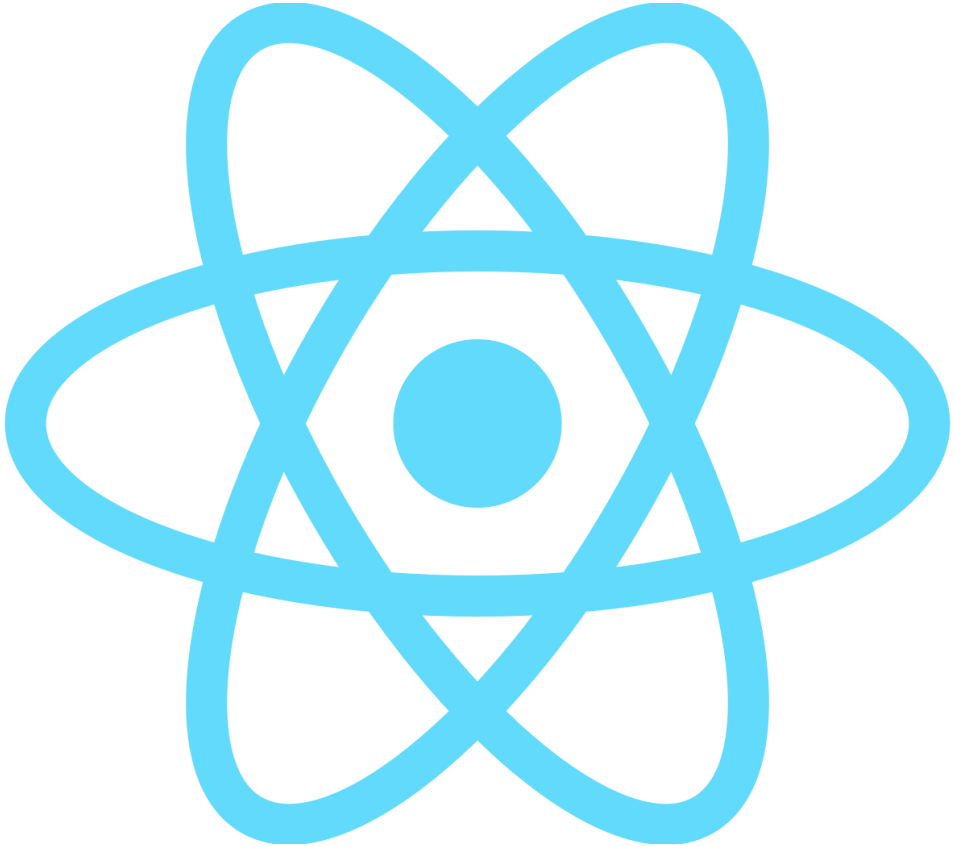
https://laravel.com/

Χρήση του Local development server της Laravel



Με την εντολή php artisan serve, μπορούμε να εκκινήσουμε τον development server και να ξεκινήσουμε να δοκιμάζουμε το API μας, είτε ολόκληρη την εφαρμογή μέσω browser.

Μπορούμε για ευκολία του local development, να χρησιμοποιήσουμε τον development server της Laravel.

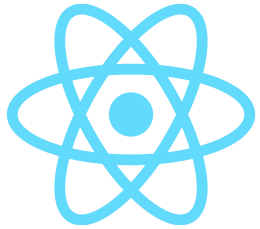


React.js

Η React είναι μία βιβλιοθήκη γραμμένη σε JavaScript, η οποία χρησιμοποιείται για την δημιουργία UI.

Δημιουργήθηκε από την Meta και συντηρείται από αυτήν.

Έχει σχετικά απλή σύνταξη, είναι component based, μπορεί να χρησιμοποιηθεί για την ανάπτυξη web-applications, καθώς και για native applications με την React-Native.



Components

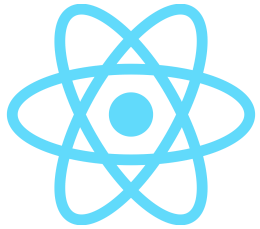
Η React όπως προαναφέρθηκα, είναι Component Based.

Αυτό σημαίνει ότι το UI που θα δημιουργήσουμε μπορούμε να το χωρίσουμε σε μικρότερα κομμάτια (τα οποία μπορούν να ξαναχρησιμοποιηθούν) που ονομάζονται Components.

Τα components μπορεί να είναι είτε Class-Components ή Function-Components (και στις 2 περιπτώσεις είναι σε ξεχωριστό αρχείο το κάθε component).

Για την React τα Class και τα Function Components είναι το ίδιο πράγμα, δεν υπάρχουν ιδιαίτερες διαφορές στην απόδοση.

Θα αναφερθώ και θα αναλύσω μόνο τα Function-Components.



Function Components

Τα Function Components είναι ουσιαστικά functions, ακριβώς όπως και οι κλασσικές functions της JavaScript.

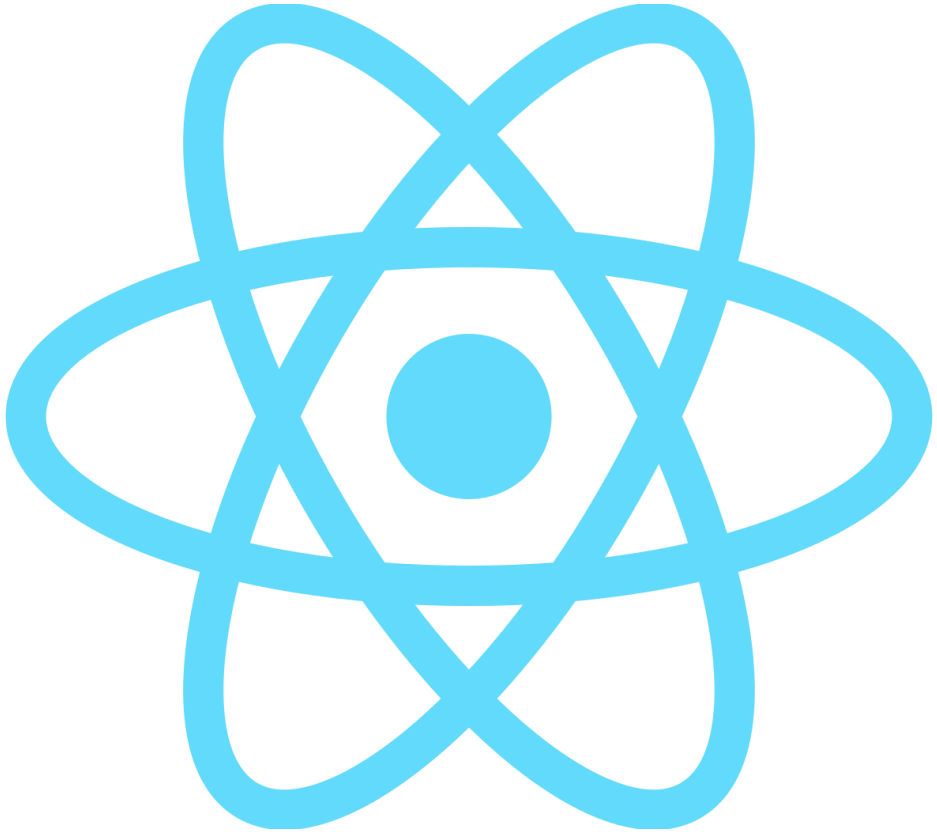
Δέχονται μία παράμετρο, τα λεγόμενα Props.

Το κάθε component είναι ξεχωριστό και διαχειρίζεται μόνο του το δικό του state (θα αναλυθεί αργότερα).

Μπορεί να επιστρέψει μόνο ένα element.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Function Components (2)



Μέσα στα Function Components μπορούμε κανονικά να γράψουμε κώδικα JavaScript, να δηλώσουμε μεταβλητές, functions.

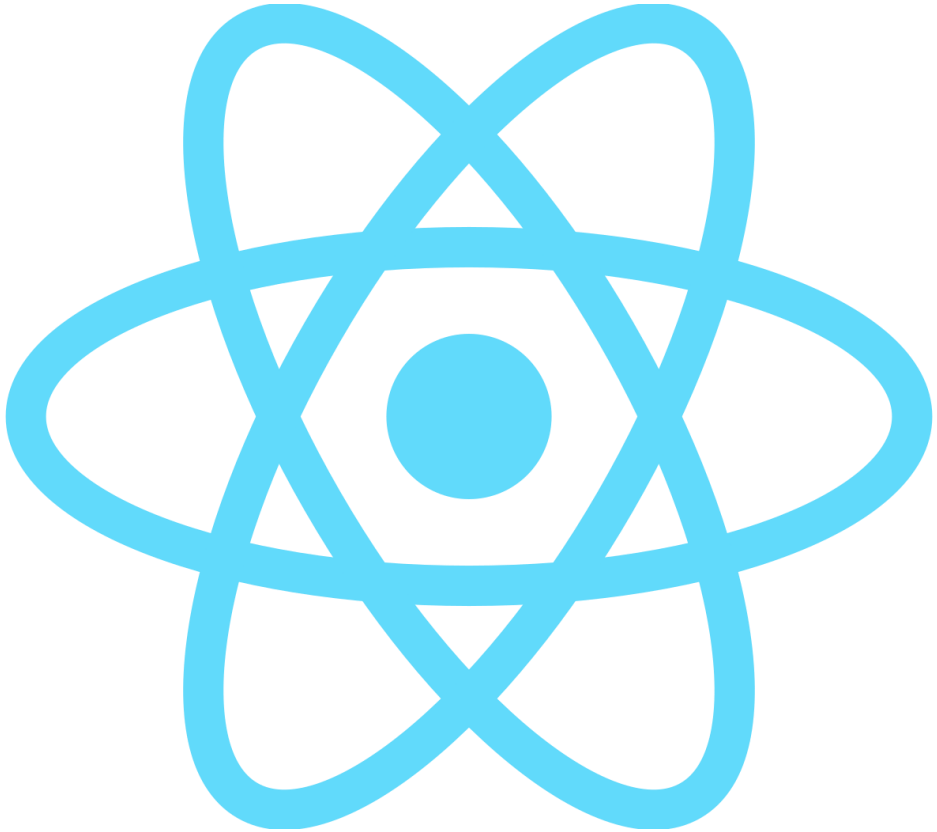
Η React έχει έναν αυστηρό κανόνα ο οποίος πρέπει να τηρείται σε κάθε περίπτωση για να λειτουργεί χωρίς προβλήματα ο κώδικας μας.

Τα props είναι read-only.

JSX

JSX από το JavaScript Syntax Extension, είναι μία επέκταση στην ουσία της σύνταξης της JavaScript έτσι ώστε να μπορούμε να ενσωματώσουμε JavaScript στην html. Θυμίζει πάρα πολύ template language (την γνωστή HTML πχ), αλλά έχει και όλες τις δυνατότητες της JavaScript.

```
const element = <h1>Hello  world </h1>;
```

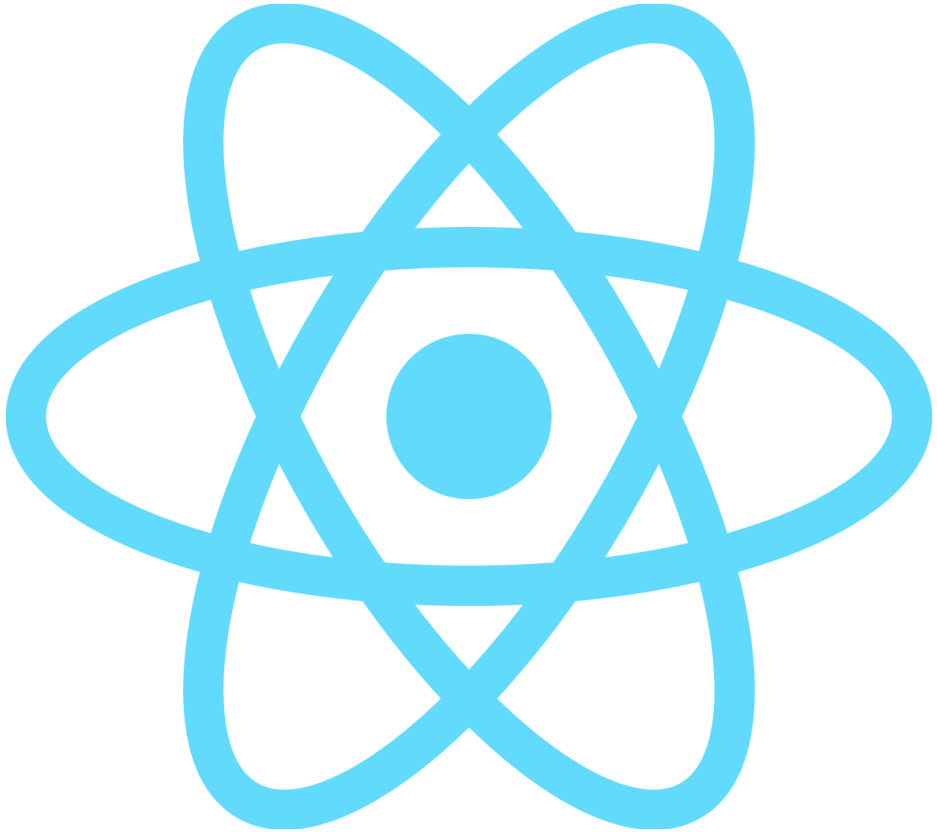


JSX (2)

Με την JSX μπορούμε όπως προαναφέρθηκα να συνδυάσουμε HTML και JavaScript, δίχως να χρειάζεται να τα ξεχωρίσουμε σε διαφορετικά αρχεία.

```
const name = " Γιώργος ";

const element = <h1> Hello  {name} </h1>;
```

Output : **Hello Γιώργος**

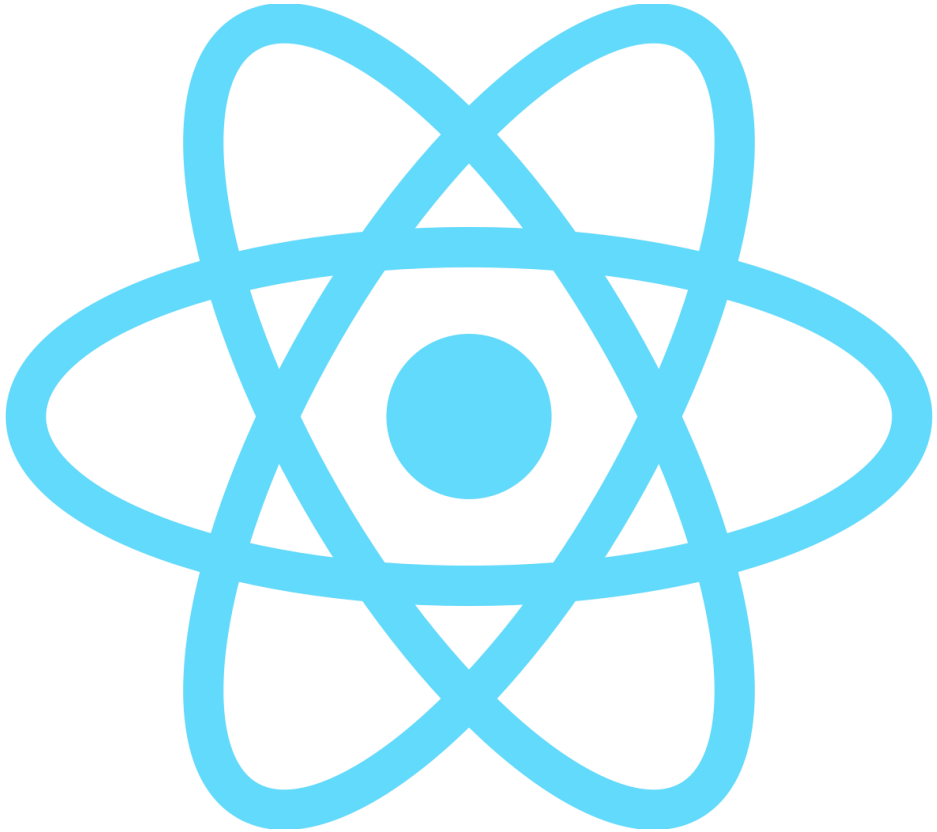


JSX (3)

Μετά το compile, οι expressions της JSX μετρατρέπονται σε JavaScript functions, οπότε μπορούμε να τις χρησιμοποιήσουμε μέσα σε if, μέσα σε loops, να τις αναθέσουμε σε μεταβλητές (όπως γίνεται στο παρακάτω παράδειγμα), να τις επιστρέψουμε από functions (όπως γίνεται με τα Function – Components).

```
const name = " Γιώργος ";

const element = <h1>Hello  {name} </h1>;
```

Output : **Hello Γιώργος**



JSX (4)

Η χρήση των function components, γίνεται επίσης με σύνταξη JSX.

Πχ Έχουμε το component UserInfo.

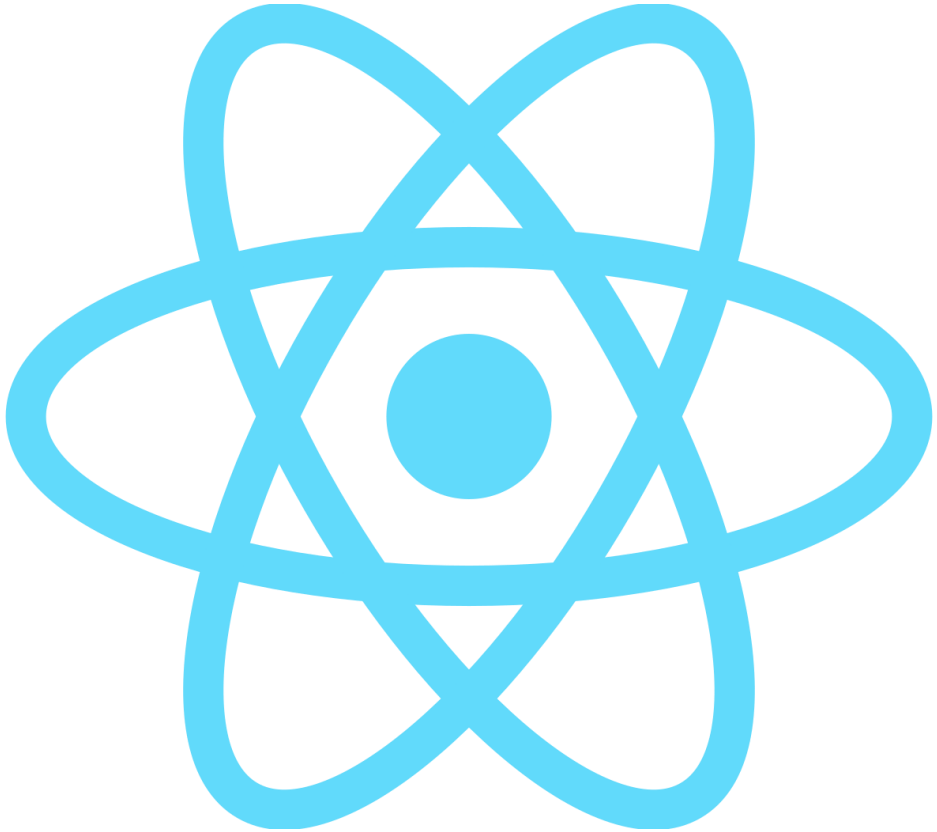
Για να το χρησιμοποιήσουμε μέσα ας πούμε σε κάποιο άλλο component θα χρησιμοποιήσουμε την παρακάτω σύνταξη : < UserInfo ></ UserInfo >

- Το κλείσιμο του tag, είναι απαραίτητο.
- Σε περίπτωση που θέλουμε να περάσουμε μία μεταβλητή μπορούμε να το κάνουμε με τον εξής τρόπο :

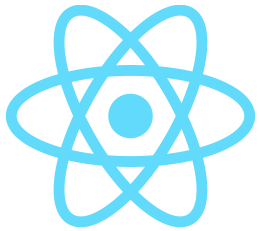
```
const  User = { name:"Giorgos",age="23" }
```

< UserInfo user={User}></ UserInfo >

Πλέον έχουμε πρόσβαση στον User,
μέσα στο component UserInfo, μέσω του props.user



Props



(1)

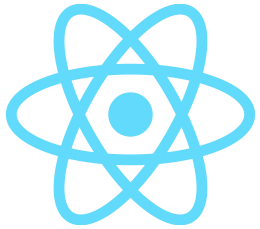
```jsx
export default function Example(props) {
    return (
        <div className="container">
            <div className="row justify-content-center">
                <div className="col-md-8">
                    <div className="card">
                        <div className="card-header">{props.Header}</div>

                        <div className="card-body">{props.Body}</div>
                    </div>
                </div>
            </div>
        </div>
    );
}
```

(2)

```jsx
export default function Example({Header,Body}) {
    return (
        <div className="container">
            <div className="row justify-content-center">
                <div className="col-md-8">
                    <div className="card">
                        <div className="card-header">{Header}</div>

                        <div className="card-body">{Body}</div>
                    </div>
                </div>
            </div>
        </div>
    );
}
```


Τα props, είναι ένα JavaScript αντικείμενο, μέσω του οποίου μπορούμε να περάσουμε μεταβλητές για να χρησιμοποιήσουμε μέσα στο component.

Όταν φτιάχνουμε το component, μπορούμε να περάσουμε σαν παράμετρο στο function απλά το props (1) (σαν ένα αντικείμενο) είτε μπορούμε να το κάνουμε destructure (2) (αν γνωρίζουμε από πρίν τι θα περιέχει).



Hooks

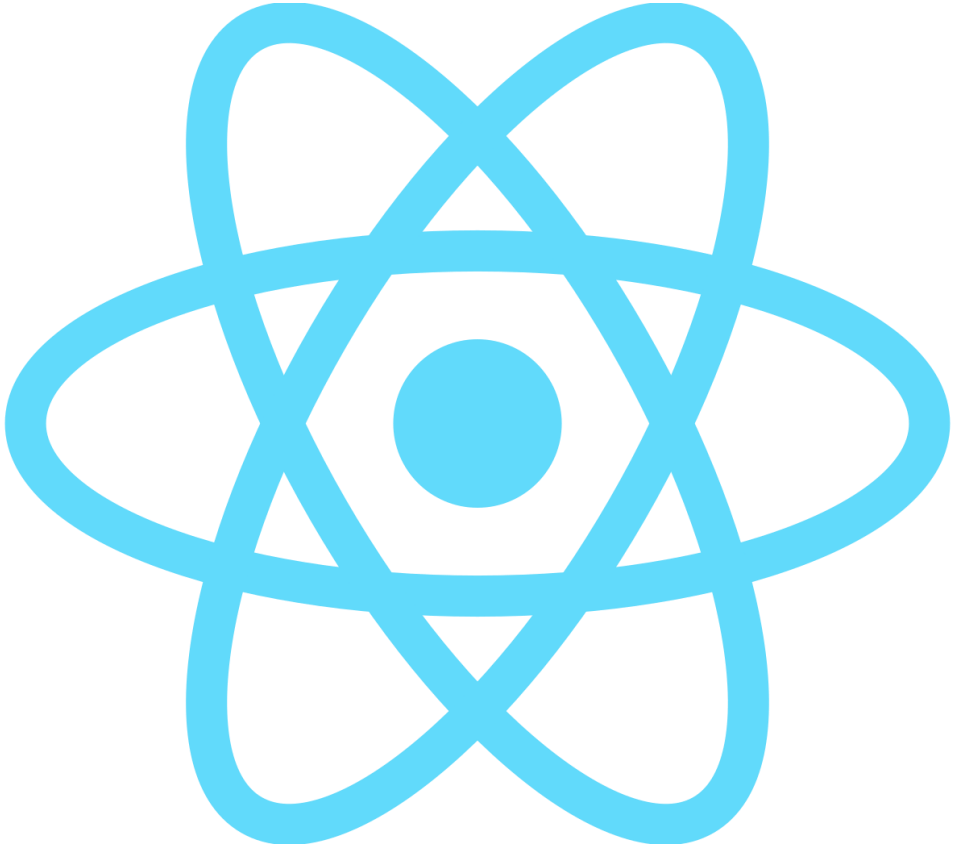
Τα Hooks είναι επίσης functions που μας δίνει η React (προστέθηκαν στην έκδοση 16.8). Ο σκοπός τους είναι η χρήση των δυνατοτήτων της React στα function components (δηλαδή δεν είναι πλέον μονόδρομος η χρήση Class – Components, επίσης τα hooks δεν λειτουργούν μέσα στα class-components).

Η React μας παρέχει out-of-the-box 15 Hooks (Αναφέρονται τα 13/15).

- useState
- useEffect
- **useContext**
- **useRef**
- **useMemo**
- **useCallback**
- **useId**
- **useLayoutEffect**
- **useTransition**
- **useImperativeHandle**
- **useReducer**
- **useDebugValue**
- **useDeferredValue**

Για την χρήση των hooks, πρέπει πρώτα να τα κάνουμε import στο component από την React

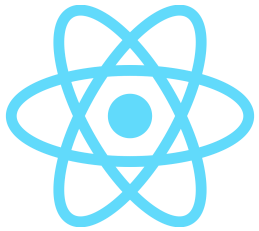
```
import React, { useState } from 'react';
```



State

Η React, χρησιμοποιεί την έννοια State για να διαχειρίζεται τα Components και είναι πολύ χρήσιμη όταν θέλουμε να φτιάξουμε δυναμικά UI's.

Το State στην React είναι ένα built-in JavaScript object το οποίο περιέχει πληροφορίες για το κάθε component, περιέχει μεταβλητές του component οι οποίες όταν αλλάξουν, θα πρέπει να αλλάξει και το UI (να γίνει ξανά render το component).



useState Hook

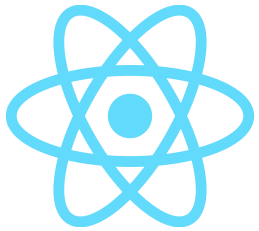
Με το hook useState, μπορούμε να χρησιμοποιήσουμε στα function-components την State.

Όπως βλέπουμε και στην εικόνα, για να φτιάξουμε έναν απλό μετρητή (πατάμε ένα κουμπί και αυξάνει ο αριθμός), το μόνο που χρειαζόμαστε από μεταβλητές είναι το count (ο τρέχων αριθμός).

```jsx
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Η μεταβλητή count δηλώνεται ώς κομμάτι του State του component με την χρήση του useState hook. Κάνοντας destructure το αντικείμενο που επιστρέφει το useState, μπορούμε να δούμε πώς περιέχει 2 πράγματα.

1. Την πραγματική μεταβλητή count.
2. Μία μέθοδο η οποία χρησιμοποιείται για να αλλάξει η τιμή της μεταβλητής count. (η αλλαγή τιμής πρέπει πάντα να γίνεται μέσω αυτής της function που επιστρέφεται από το hook.



useState Hook (2)

Το hook useState μπορεί να πάρει μία παράμετρο, η οποία θα είναι και η αρχική τιμή της μεταβλητής που θέλουμε να δηλώσουμε (στην προκειμένη περίπτωση το count). Μπορεί σαν αρχική τιμή να πάρει οποιουδήποτε primitive τύπου μεταβλητή.

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Όταν πατάμε το button, μπορούμε να δούμε πώς καλείται μία function (arrow function) και το count, μέσω της setCount παίρνει την τιμή count + 1.

Από την στιγμή που μία μεταβλητή που ανήκει στο state του component έχει πάρει άλλη τιμή (πάντα μέσω της αντίστοιχης set...) η React, αυτόματα θα κάνει re-render το component και ο αριθμός που θα βλέπουμε θα είναι πλέον η καινούργια τιμή του count.

useEffect Hook

Το Hook useEffect, χρησιμοποιείται σε περίπτωση που θέλουμε εκτελεστούν εντολές, είτε μόνο την πρώτη φορά, είτε κάθε φορά που γίνεται render του component, είτε μόνο όταν αλλάζει κάποια μεταβλητή στο component (στην ουσία να μην εκτελεστούν αυτές οι εντολές μέσα στο hook όταν δεν έχει αλλάξει το dependency του).

Το hook useEffect, παίρνει υποχρεωτικά τουλάχιστον μία παράμετρο, πρέπει να είναι ένα callback.Προαιρετικά μπορεί να πάρει και μία δεύτερη παράμετρο, έναν πίνακα, ο οποίος θα περιέχει τα dependencies του hook.

Τα dependencies του useEffect, είναι οι μεταβλητές που όταν αλλάζουν τιμή, μόνο τότε θα εκτελούνται οι εντολές μέσα στο hook.

πχ

- useEffect (()=>{εντολές}), οι εντολές μέσα στο hook, σε αυτήν την περίπτωση θα εκτελούνται σε κάθε render.
- useEffect (()=>{εντολές},[]), οι εντολές μέσα στο hook, σε αυτήν την περίπτωση θα εκτελούνται μόνο στο πρώτο render του component.
- useEffect (()=>{εντολές},[count]), οι εντολές μέσα στο hook, σε αυτήν την περίπτωση θα εκτελούνται μόνο όταν αλλάζει η τιμή του count.

Event Handling

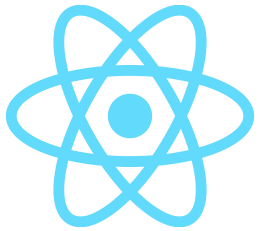
Όταν πατάμε ένα κουμπί, όταν στέλνουμε ένα form στον server γίνεται trigger ένα event. Σε plain html, ο τρόπος που θα διαχειριζόμασταν το event onClick πχ ενός button, θα ήταν να περάσουμε σε string, το όνομα της function που έχουμε φτιάξει για να το διαχειριστεί. Στην React περνάμε την μέθοδο και όχι αναφορά στο όνομα της μεθόδου ώς string.

Plain HTML

```
<button onclick="activateLasers()"> Activate Lasers </button>
```

React & JSX

```
<button onClick={activateLasers}> Activate Lasers </button>
```



Conditional Rendering

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

Υπάρχουν περιπτώσεις όπου θέλουμε ανάλογα με την τιμή μίας μεταβλητής, είτε είναι μία απλή Boolean true/false, είτε ένας αριθμός, είτε ένα string, να εμφανίσουμε διαφορετικά πράγματα.

Με την React αυτό είναι αρκετά εύκολο, καθώς όπως είπαμε οι εκφράσεις της JSX μπορούν να χρησιμοποιηθούν και μέσα σε if, loops κτλπ.

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

Lists

Έστω ότι έχουμε έναν πίνακα μέσα στον οποίο έχουμε αποθηκευμένα JavaScript Objects που αναφέρονται σε προϊόντα. Πχ

const Products = [Product1, Product2, Product3, Product4];

Και το Component <Product></Product>.

Θέλουμε για κάθε προϊόν στον πίνακα Products, να εμφανίσουμε ένα <Product product={x}></Product> (όπου Χ το προϊόν που θέλουμε να περάσουμε).

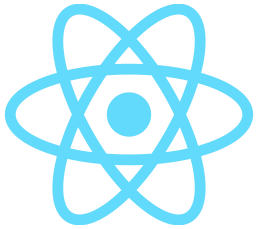
Με την map(), που λειτουργεί στην ουσία σαν μία foreach, μπορούμε να εκτελέσουμε εντολές για κάθε αντικείμενο που περιέχεται στον πίνακα πάνω στον οποίο την καλούμε.

Οπότε const ProductList = Products.map((Product) =>

{ return <Product product={Product}></Product>; });

Και έπειτα να τα εμφανίσουμε μέσα σε μία ul.

{ProductList}



Keys

Όταν δημιουργούμε λίστες με την React, είναι απαραίτητη ή χρήση ενός ακόμα attribute στα components που συμπεριλαμβάνονται στην λίστα. Το λεγόμενο Key.

Το Key είναι στην ουσία ένα string attribute, το οποίο χρησιμοποιεί η React εσωτερικά για να αναγνωρίσει ποιά αντικείμενα στο DOM έχουν αλλάξει, αφαιρεθεί ή προστεθεί σε κάθε render που κάνει.

Είναι πολύ σημαντικό να μην ξεχνάμε το Key attribute σε λίστες (Εμφανίζεται και αντίστοιχο error).

Το Key :

- Πρέπει να είναι μοναδικό ανάμεσα στα αντικείμενα της ίδιας λίστας (όχι globally unique).
- Πρέπει να είναι κάτι σταθερό και όχι μεταβλητό.
- Πολλές φορές χρησιμοποιούμε το id (αν υπάρχει) του αντικειμένου που περνάμε στην λίστα.

Σας ευχαριστώ που με παρακολουθήσατε.