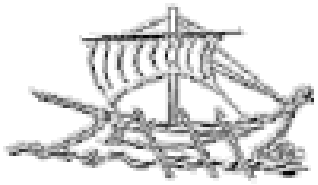


ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Τ.Ε.

Πτυχιακή εργασία

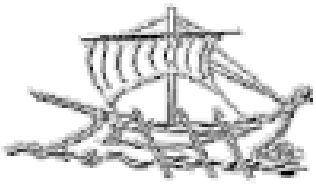
ΠΛΑΤΦΟΡΜΑ ΗΛΕΚΤΡΟΝΙΚΟΥ ΕΜΠΟΡΙΟΥ ΜΕ ΔΥΝΑΤΟΤΗΤΑ ΦΙΛΟΞΕΝΙΑΣ ΠΟΛΛΑΠΛΩΝ ΚΑΤΑΣΤΗΜΑΤΩΝ

Καλεπανάγος Διονύσιος
Γαϊτανάκης Εμμανουήλ

Επιβλέπων Καθηγητής

Γεώργιος Πρεζεράκος

Αθήνα, Σεπτέμβριος 2017



PIRAEUS UNIVERSITY OF APPLIED SCIENCES

SCHOOL OF ENGINEERING

DEPARTMENT of COMPUTER SYSTEM ENGINEERING

THESIS

E-commerce platform that has the ability to host multiple shops

Kalepanagos Dionysios

Gaitanakis Emmanouil

Supervisor Professor

Georgios Prezerakos

Athens, September 2017

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΝΕΥΜΑΤΙΚΑ ΔΙΚΑΙΩΜΑΤΑ

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Ηλεκτρονικών Υπολογιστικών Συστημάτων του Ανώτατου Εκπαιδευτικού Ιδρύματος Πειραιά δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό της ανάπτυξη μιας διαδικτυακής εφαρμογής. Την προσπάθειά μας αυτή υποστήριξε ο επιβλέπων καθηγητής μας, Γεώργιος Πρεζεράκος, τον οποίο θα θέλαμε να ευχαριστήσουμε.

ΠΕΡΙΛΗΨΗ

Η πτυχιακή αυτή αφορά μια πλατφόρμα ηλεκτρονικού εμπορείου η οποία θα έχει δυνατότητα να φιλοξενεί πολλαπλά μαγαζιά. Σε αυτήν την εφαρμογή θα υπάρχουν 3 ειδών χρήστες: Ο Διαχειριστής, ο Πωλητής και ο αγοραστής.

Ο διαχειριστής θα έχει πρόσβαση εκτός της κεντρικής σελίδας, αλλά και μια σελίδα διαχείρισης, από την οποία μπορεί να βλέπει πληροφορίες σχετικά με την εφαρμογή όπως τα προϊόντα της πλατφόρμας, τους εγγεγραμμένους χρήστες και διάφορα σχεδιαγράμματα για την καλύτερη απεικόνιση της λειτουργίας της πλατφόρμας. Επίσης θα μπορεί να εγγράφει νέους χρήστες και να αλλάζει το ρόλο τους, αν χρειαστεί. Τέλος θα μπορεί να κάνει εγκρίνει τα προϊόντα των χρηστών, ώστε να φαίνονται στη σελίδα.

Οι πωλητές, θα μπορούν να ανεβάζουν τα προϊόντα τους στην σελίδα, θα μπορούν να τα επεξεργάζονται και θα οργανώνονται σε φακέλους δυναμικά. Θα έχουν πρόσβαση στις πληροφορίες τους και θα μπορούν να τις τροποποιήσουν, αν το θελήσουν.

Οι απλοί χρήστες θα μπορούν να επεξεργάζονται τα στοιχεία τους, αφού φτιάξουν λογαριασμό. Θα μπορούν να ψάχνουν για διάφορα προϊόντα μέσω της μηχανής αναζήτησης, η οποία θα έχει φίλτρα για την ευκολότερη εύρεση προϊόντων και θα υπάρχει δυνατότητα καλαθιού για την αγορά τους.

ABSTRACT

This thesis concerns an e-commerce platform that has the ability to host multiple shops. In this application there will be 3 types of users: An Administrator, a Vendor and a User.

The administrator will have access to the home page and also to the management page, from which he can view information about the information such as the products of the platform, registered users and διάφορα various plots and diagrams for better visualization of the operating platform. Also he can create new users and change their role, if needed. Finally, he can validate user's products to appear on the page.

Sellers will be able to upload their products on the page, they can modify them and it will be organized into folders dynamically. They will have access to their information and can modify them, if they want.

Simple users can edit their information, upon creating an account. They can search for products through a search engine, which will have filters to make it easier to find products and there will be a cart functionality so they can purchase an interesting product.

ΠΟΙΟ ΠΡΟΒΛΗΜΑ ΠΑΜΕ ΝΑ ΛΥΣΟΥΜΕ

Στην πτυχιακή μας εργασία θέσαμε ως στόχο να λύσουμε το πρόβλημα που έχει κάθε ιδιοκτήτης καταστήματος στην αγορά: Έναν προσωπικό χώρο στην πλατφόρμα μας, στον οποίο ο κάθε μαγαζάτορας θα μπορεί να προωθήσει σε μεγαλύτερη εμβέλεια τα προϊόντα του απ' ότι το φυσικό του κατάστημα, αλλά και να τα πουλήσει ευκολότερα, μέσω διαδικτύου. Αυτό εάν το επιχειρούσε με μία προσωπική ιστοσελίδα, θα ήταν σίγουρα πολύ πιο δαπανηρό για ένα μικρό κατάστημα. Επίσης η διαχείρισή του πιθανότατα να φανεί δύσκολη σε έναν χρήστη με ελάχιστες γνώσεις πάνω στους ηλεκτρονικούς υπολογιστές, αντίθετα με την πλατφόρμα μας, η οποία είναι προσιτή και εύκολη στην χρήση από όλους.

ΠΑΡΟΜΟΙΕΣ ΠΛΑΤΦΟΡΜΕΣ ΣΤΟ ΔΙΑΔΙΚΤΥΟ

Για να γίνει αυτό το εγχείρημα, πρώτα κάναμε μία έρευνα για να δούμε τι λειτουργίες πρέπει να έχει η πλατφόρμα, πως πρέπει να στηθεί, τις ανάγκες που θα προκύψουν κλπ. Έτσι η πλατφόρμα βασίστηκε στα πρότυπα των μεγαλύτερων και πιο γνωστών e-shop που κυριαρχούν στον τομέα τους, όπως το **Amazon.** Το Amazon είναι μία από τις μεγαλύτερες εταιρίες στον κόσμο στην πώληση αγαθών και υπηρεσιών μέσω του διαδικτύου και μια από τις πρώτες που βασίστηκε το ηλεκτρονικό εμπόριο. Το amazon σου προσφέρει τη δυνατότητα να αναρτήσεις στην πλατφόρμα έναν αριθμό προϊόντων ανάλογα με το πακέτο που θα επιλέξεις, το οποίο βασίζεται στο πλήθος προϊόντων που έχεις σκοπό να πουλήσεις. Η επικοινωνία με τον αγοραστή καθώς και την αποστολή των προϊόντων την αναλαμβάνεται το Amazon. [1]

Μεγαλύτερη ομοιότητα με το Amazon καθώς και με την πλατφόρμα μας, έχει το Shopify. Το Shopify προσφέρει την δυνατότητα στον χρήστη να δημιουργήσει το δικό του ηλεκτρονικό κατάστημα στην πλατφόρμα, να πουλήσει, να οργανώσει καθώς και να παρακολουθεί τις παραγγελίες που λαμβάνει. Το μόνο που χρειάζεσαι για να δημιουργήσεις το δικό σου ηλεκτρονικό κατάστημα είναι μία πιστωτική κάρτα. [2]

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

ORM:	Object Relational Mapping
MVC:	Model View Controller
PHP	Hypertext Preprocessor
IRC	Internet Relay Chat
SQL	Structured Query Language
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
GUI	Graphic User Interface
CSTF	Cross Site Request Forgery
AJAX	Asynchronous JavaScript and XML
XML	eXtensible Markup Language
CVC	Card Verification Code
CDN	Content Delivery Network
ID	Identification
JSON	JavaScript Object Notation

ΑΠΟΔΟΣΗ ΟΡΩΝ

Framework	Πλαίσιο
User Interface	Περιβάλλον χρήστη
Caching	Προσωρινή μνήμη
Database	Βάση δεδομένων
Backup	Αντίγραφο
Sessions	Σεζόν
Shopping Cart	Καλάθι αγορών
Model	Μοντέλο
Full Stack	Πλήρη στοίβα
Public key	Δημόσιο κλειδί
Secret key	Μυστικό κλειδί
Token	Διακριτικό
Administrator	Διαχειριστής
Vendor	Πωλητής
User	Χρήστης
Search engine	Μηχανή Αναζήτησης
e-shop	Ηλεκτρονικό κατάστημα
Server	Διακομιστής
Authentication	Αυθεντικοποίηση
Routing	Δρομολόγηση
Developers	Προγραμματιστές
Gatekeeper	Φύλακας
View	Προβολή
Activity diagram	Διάγραμμα δραστηριοτήτων

Response	Απάντηση
Request	Αίτημα
Argument	Επιχείρημα
Currency	Συνάλλαγμα
Documentation	Εγχειρίδιο
Description	Περιγραφή
Add to Cart	Προσθήκη στο καλάθι
Checkout	Ταμείο
Payment details	Λεπτομέρειες πληρωμής
Web	Ιστός
Table	Τραπέζι
Roles	Ρόλοι
Name	Όνομα
Access	Πρόσβαση
Seeder	Σπορέας
Migration	Μετανάστευση
Hashing	Κατατεμαχισμός
Orders	Παραγγελίες
Stock	Απόθεμα
Thumbnail	Συνοπτικό
Subcategories	Υποκατηγορίες
Image	Εικόνες
Column	Στήλη
Hover	Αιώρηση
Input	Εισαγωγή

Regular Expression	Κανονική έκφραση
Middleware	Ενδιάμεσο λογισμικό
Brackets	Παρενθέσεις
Redirect	Ανακατεύθυνση
Trim	Κόβω
Blur	Θολώνω
Opacity	Αδιαφάνεια
Greyscale	Ασπρόμαυρο
Resize	Αλλαγή μεγέθους
Markup	Μαρκάρισμα
Path	Μονοπάτι

1 ΛΕΙΤΟΥΡΓΙΚΕΣ ΑΠΑΙΤΗΣΕΙΣ ΣΥΣΤΗΜΑΤΟΣ

Η ανάλυση απαιτήσεων είναι μια διαδικασία κατάρτισης μια λίστας, όπου αναφέρονται οι προδιαγραφές που πληροί η εφαρμογή μας.

Οι προδιαγραφές που προσδιορίζονται μπορεί να είναι τεχνολογικές, επιχειρηματικές, λειτουργικές και να σχετίζονται με την μορφή και το κόστος. Η λίστα που καταρτίζεται, χρησιμεύει τόσο σε αυτούς που θα εγκαταστήσουν ή θα αναπτύξουν το λογισμικό, όσο και σε εκείνους που θα τη χρησιμοποιήσουν και εφαρμόζεται σε μεγάλα και μικρά έργα πληροφορικής.

Συχνά προγραμματιστές, αναλυτές, πελάτες και δυνητικοί χρήστες υποτιμούν την ανάλυση απαιτήσεων και δεν της αποδίδουν τη δέουσα σημασία. Αυτό έχει ως αποτέλεσμα τη δημιουργία έργων που χρήζουν βελτιώσεων ή αλλαγών, γιατί δεν ανταποκρίνονται στους στόχους και τις επιδιώξεις που είχαν αρχικά τεθεί. Ωστόσο, οι βελτιώσεις και αλλαγές στα έργα πληροφορικής είναι ιδιαίτερα δαπανηρές, δύσκολες και χρονοβόρες, ενώ η πραγματοποίησή τους μπορεί να απαιτήσει εκ θεμελίων αναδημιουργία. Για το λόγο αυτό, η ανάλυση απαιτήσεων θεωρείται το συστατικό για επιτυχημένη υλοποίηση εφαρμογών.

1.1 ΟΦΕΛΕΙΕΣ ΑΝΑΛΥΣΗΣ ΑΠΑΙΤΗΣΕΩΝ

Οι σημαντικότερες ωφέλειες που απορρέουν από τη χρήση της ανάλυσης των απαιτήσεων είναι οργανωτικές, λειτουργικές και οικονομικές, με αυτήν ακριβώς τη σειρά, όχι βάσει σπουδαιότητας αλλά χρονικής ακολουθίας.

Η ανάλυση απαιτήσεων συντελεί στην καλή οργάνωση και εκτέλεση του έργου, που με τη σειρά τους εξασφαλίζουν τη λειτουργικότητά του για όλες τις εμπλεκόμενες πλευρές. Στο τέλος, τα οφέλη αυτά έχουν άμεσο αντίκρισμα στη μείωση του κόστους, τόσο για την επιχείρηση που υλοποιεί το έργο όσο και για τον πελάτη που θα το χρησιμοποιήσει.

Η ανάλυση απαιτήσεων μειώνει το κόστος υλοποίησης του έργου, καθώς εξασφαλίζει ότι θα γίνει βάσει χρονοδιαγράμματος. Αυτό έχει μεγάλη αξία γιατί αν δεν έχει προηγηθεί ανάλυση απαιτήσεων παρατηρούνται φαινόμενα όπως:

1. Το πρόγραμμα αποδεικνύεται ότι έχει αρκετά ελαττώματα (bugs), τα οποία πρέπει να διορθωθούν με "μπαλώματα" και τη συγγραφή καινούργιου κώδικα.
2. Η εφαρμογή που δημιουργήθηκε αποκλειστικά για ένα συγκεκριμένο πελάτη αποδεικνύεται ελλιπής στην πράξη και χρήζει βελτίωσης.
3. Ο πελάτης συνειδητοποίησε -την τελευταία στιγμή- ότι θέλει το ηλεκτρονικό του κατάστημα να περιέχει και "κάτι ακόμα". [3]

1.2 ΔΥΝΑΤΟΤΗΤΕΣ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ

- Εγγραφή χρηστών στο σύστημα
- Χρήστες με διαφορετικά δικαιώματα
- Μηχανή αναζήτησης προϊόντων
- Σχολιασμό προϊόντων
- Ποσότητα στην αποθήκη
- Ανάρτηση προϊόντων από πωλητές
- Επεξεργασία αναρτημένων προϊόντων από πωλητές
- Παρακολούθηση παραγγελιών

1.3 ΧΡΟΝΟΣ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΕΡΓΟΥ

Ο χρόνος υλοποίησης του έργου εξαρτάται από το εργατικό δυναμικό που θα το υποστηρίξει καθώς και τις προσωπικές τους γνώσεις. Το εργατικό δυναμικό στην δικιά μας πλατφόρμα είναι δύο άτομα, οπότε ο εκτιμώμενος χρόνος ολοκλήρωσης με συστηματική ενασχόληση είναι περίπου στους δύο με τρεις μήνες, γιατί είναι ειδικά σχεδιασμένο με τον κατάλληλο κώδικα για να γίνει η εμπλούτισή του σε περιεχόμενο από τους πωλητές που θα δημοσιοποιούν τα προϊόντα τους.

1.4 ΣΕΙΡΑ ΟΛΟΚΛΗΡΩΣΗΣ ΕΡΓΟΥ

Η σειρά ολοκλήρωσης του έργου μας είναι ένα από τα πιο σημαντικά κομμάτια που μας απασχόλησε στην πτυχιακή μας για να είναι έτοιμο σε λογικό χρονικό περιθώριο.

Τα βήματα που ακολουθήσαμε έτσι ώστε να δημιουργηθεί η διαδικτυακή εφαρμογή ήταν αρχικά με ελάχιστα γραφικά και εφέ, διότι δώσαμε προτεραιότητα στο να γίνει πρώτα λειτουργική. Οπότε, ξεκινήσαμε αρχικά με τον σχεδιασμό της βάσης δεδομένων, να ξέρουμε πόσους πίνακες θα έχουμε και ποιοι και αν θα συνδέονται μεταξύ τους. Έπειτα στην συγγραφή κώδικα για κάθε μια από τις δυνατότητες που είχαμε αποφασίσει πως θα έχει η πλατφόρμα.

1.5 ΑΣΦΑΛΕΙΑ ΔΕΔΟΜΕΝΩΝ

Ένα από τα μεγαλύτερα ζητήματα στον κόσμο της πληροφορίας είναι η ασφάλεια των δεδομένων για αυτό πρέπει να ακολουθήσουμε κάποια πρωτόκολλα ασφαλείας.

- Έλεγχος στα δεδομένα που αποθηκεύουμε.
- Database Backups
- Φυσική ασφάλεια

1.5.1 **Έλεγχος στα δεδομένα που αποθηκεύουμε**

Ο τρόπος που θα δημιουργήσουμε την πλατφόρμα μας έχει έτοιμο σύστημα ασφάλειας το οποίο μας διασφαλίζει ότι τα δεδομένα που ζητάμε, περνάνε ένα φιλτράρισμα ώστε να σταλθούν στον Server μας και δεν επιτρέπει σε ανεπιθύμητα στοιχεία από κακόβουλους χρήστες να αποθηκευτούν στην βάση δεδομένων μας.

1.5.2 **CSRF επιθέσεις**

Μία από τις πιο συχνές επιθέσεις που γίνονται από κακόβουλους χρήστες είναι η επίθεση CSRF. Με την μέθοδο αυτή ο κακόβουλος χρήστης εξαπατά το πρόγραμμα περιήγησης για την αποστολή ανεπιθύμητου αιτήματος HTTP στο οποίο πέφτει θύμα ο χρήστης.

Το Laravel δημιουργεί αυτόματα ένα "διακριτικό" CSRF για κάθε ενεργό χρήστη που διαχειρίζεται η εφαρμογή. Αυτό το διακριτικό χρησιμοποιείται για να επαληθεύσει ότι ο πιστοποιημένος χρήστης είναι αυτός που κάνει τα αιτήματα στην εφαρμογή.

1.5.3 **Database Backups**

Ανά τακτά χρονικά διαστήματα υπάρχει επιλογή να γίνονται Backup στην βάση δεδομένων. Με την λειτουργία αυτή μπορούμε να αξιοποιήσουμε τους πόρους του συστήματος πιο αποδοτικά καθώς σε οποιαδήποτε χρονική στιγμή μπορούμε να ανακτήσουμε τις πληροφορίες που είχαμε στην βάση δεδομένων.

1.5.4 **Φυσική ασφάλεια**

Διαλέγουμε πάντα τα αρχεία μας να είναι σε ένα Server αξιόπιστο που μας εγγυάται ότι σε κάθε φυσική καταστροφή που μπορεί να γίνει στον Server θα έχουν φροντίσει να μην επηρεαστούν τα δικά μας δεδομένα.

1.6 **ΔΥΝΑΤΟΤΗΤΕΣ ΑΝΑΒΑΘΜΙΣΗΣ**

Κάθε εφαρμογή πρέπει να είναι αναβαθμισμένη στην τελευταία έκδοση γιατί μας παρέχει με προστασία από τις συνεχώς εξελισσόμενες απειλές.

Άτομα με κακόβουλες προθέσεις βρίσκουν συνεχώς νέους και εφευρετικούς τρόπους για να επιτεθούν στις εφαρμογές για να πάρουν πληροφορίες από το σύστημα ή ακόμα και να το καταστρέψουν.

Δεν υπάρχει ενδεδειγμένη λύση καθώς οι απειλές εξελίσσονται, με τις αναβαθμίσεις του συστήματος όμως προλαβαίνουμε να τις αντιμετωπίσουμε. Το Laravel όπως και κάθε άλλο Framework φροντίζει να μας ενημερώνει όποτε έχουμε να εγκαταστήσουμε μια αναβάθμιση του συστήματος. [4]

2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΛΑΤΦΟΡΜΑΣ

Η ανάλυση της αρχιτεκτονικής της κάθε εφαρμογής είναι μια διαδικασία την οποία την κάνει ο κάθε προγραμματιστής, ούτως ώστε να έχει ένα πλάνο για το πώς θα στηθεί η εφαρμογή, τι εργαλεία θα χρησιμοποιήσει, ποιες τεχνολογίες κλπ. Είναι από τις πιο σημαντικές δουλειές του προγραμματιστή, διότι δίχως μια σχετική ανάλυση, υπάρχει περίπτωση να εμφανιστούν προβλήματα και η πρόοδος του προγράμματος να καθυστερήσει. Παρακάτω θα αναλύσουμε επί μέρους, τα κομμάτια αυτά:

2.1 PHP - LARAVEL

Το Laravel είναι ένα σχετικά νέο PHP framework (κυκλοφόρησε το 2011), όπου σύμφωνα με την πρόσφατη online έρευνα του Sitepoint, είναι το πιο δημοφιλές Framework ανάπτυξης μεταξύ των προγραμματιστών. Το Laravel διαθέτει ένα τεράστιο οικοσύστημα με μια άμεση πλατφόρμα φιλοξενίας και ανάπτυξης και το επίσημο website του, προσφέρει πολλά σεμινάρια προβολής με τίτλο Laracasts.

Το Laravel επιχειρεί να βγάλει τον πόνο από την ανάπτυξη διευκολύνοντας τα κοινά καθήκοντα που χρησιμοποιούνται στην πλειοψηφία των έργων ιστού όπως το authentication, routing, sessions, και caching.

Σκοπός του Laravel είναι να καταστήσει την διαδικασία ανάπτυξης μιας εφαρμογής ευχάριστη για τον προγραμματιστή χωρίς να θυσιάζει τη λειτουργικότητα της εφαρμογής. Οι ευτυχείς προγραμματιστές κάνουν τον καλύτερο κώδικα.

2.2 ΑΛΛΑ FRAMEWORKS

Τα Frameworks είναι οικοσυστήματα από μόνα τους, η χρήση τους είναι για να επιταχύνεις την ποιοτική παραγωγή κώδικα χωρίς ζητήματα ασφάλειας καθώς και να εμπλουτίσεις και να συντηρήσεις εύκολα τον ήδη υπάρχων κώδικα.

Για την δημιουργία της πτυχιακής χρησιμοποιήθηκε το Laravel: ένα ανοιχτού κώδικα PHP framework το οποίο δημιουργήθηκε από τον Taylor Otwell και προορίζεται για την ανάπτυξη web εφαρμογών ακολουθώντας την αρχιτεκτονική μοντέλου-προβολή-ελεγκτή (MVC). Παρακάτω, μπορούμε να δούμε μερικά από αυτά:

Symfony

Το Symfony είναι ένα σύνολο από στοιχεία PHP που εύκολα ξαναχρησιμοποιούνται. Αυτά επιτρέπουν στον προγραμματιστή να δημιουργεί κλιμακούμενες εφαρμογές υψηλής απόδοσης. Με 30 στοιχεία από τα οποία μπορεί να επιλέξει, ο προγραμματιστής έχει την πλήρη ελευθερία να πειραματίζεται και να εργάζεται σε ένα περιβάλλον ταχείας ανάπτυξης εφαρμογών. Τα API του Symfony επιτρέπουν επίσης την εύκολη ενσωμάτωση με εφαρμογές τρίτων και μπορούν να χρησιμοποιηθούν με δημοφιλή Front-End Frameworks, όπως το AngularJS.

Πολλά δημοφιλή έργα, συμπεριλαμβανομένων των Drupal και phpBB, χρησιμοποιούν επίσης το Symfony Framework. Στην πραγματικότητα, το Laravel, το πιο δημοφιλές Framework της PHP, κατασκευάζεται από την Symfony. [5]

Code Igniter

Το CodeIgniter αρχικά κυκλοφόρησε το 2006 και είναι ένα ελαφρύ PHP Framework. Αυτό έχει μια πολύ απλή διαδικασία εγκατάστασης που απαιτεί μόνο μια ελάχιστη διαμόρφωση. Είναι επίσης μια ιδανική επιλογή αν θέλετε να αποφύγετε τη σύγκρουση εκδόσεων της PHP, καθώς λειτουργεί ωραία σε όλες σχεδόν τις κοινές και αποκλειστικές πλατφόρμες φιλοξενίας.

Yii 2

Το Yii 2 είναι ενσωματωμένο με jQuery και διαθέτει ένα σύνολο δυνατοτήτων AJAX και εφαρμόζει έναν εύκολο στη χρήση μηχανισμό σχεδίασης, έτσι ώστε να μπορεί να αποτελέσει μια εξαιρετική επιλογή για κάποιον που προέρχεται από ένα περιβάλλον Front-End. Έχει επίσης μια ισχυρή γεννήτρια κωδικών τάξης που ονομάζεται Gii που διευκολύνει τον αντικειμενοστραφή προγραμματισμό και τα γρήγορα πρωτότυπα και παρέχει μια διαδικτυακή διεπαφή που μας επιτρέπει να δημιουργούμε διαδραστικά τον κώδικα που χρειαζόμαστε.

Phalcon

Το Phalcon είναι ένα άλλο μοντέρνο Framework, το οποίο είναι κατασκευασμένο για ταχύτητα. Είναι μια επέκταση PHP γραμμένη σε C και αναμφισβήτητα το ταχύτερο PHP Framework που είναι διαθέσιμο από σήμερα. Παρόλο που είναι μια επέκταση γραμμένη σε C, είναι πλήρες MVC Framework και προσφέρει τα περισσότερα από τα

σύγχρονα χαρακτηριστικά όπως δρομολόγηση, ελεγκτές, πρότυπα προβολής, γλώσσα ερωτήματος, Caching και ORM. [6]

2.2.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΟΥ FRAMEWORK

Φορητότητα

Ο τρόπος επιλογής της βάσης δεδομένων και συστήματος για Caching επιτρέπει την εγκατάσταση σε πολλές διαφορετικές εγκαταστάσεις server με διαφορετικές παραμέτρους και χαρακτηριστικά. Επίσης αν η εφαρμογή είναι ανοικτού κώδικα, με την δυνατότητα της φορητότητας είναι πιθανότερο περισσότεροι χρήστες να την χρησιμοποιήσουν.

Ταχύτερη ανάπτυξη

Τα Frameworks έχουν μεγάλες συλλογές από βιβλιοθήκες και συμβάλουν στην γρήγορη υλοποίηση μιας εργασίας. Για παράδειγμα, δεν χρειάζεται να γράψεις κάποια κλάση για να συνδεθείς ή να διαχειριστείς τις βάσεις δεδομένων ή να γράψεις κλάσεις για την διαχείριση των χρηστών.

Ασφάλεια της εφαρμογής

Τα χαρακτηριστικά ασφαλείας όπως είναι η πιστοποίηση χρηστών και η διαχείριση των δικαιωμάτων τους, διαχειρίζονται από το Framework. Οι εγγραφές στη βάση δεδομένων είναι ασφαλείς, χωρίς SQL Injections.

Υποστήριξη από την κοινότητα

Τα Frameworks υποστηρίζονται από κοινότητες και κανάλια IRC. Σε περίπτωση που αντιμετωπίσεις κάποιο πρόβλημα με το Framework, μπορείς να απευθυνθείς σε μια κοινότητα χρηστών και να λάβεις απαντήσεις, όπως επίσης να δεις τα προβλήματα που αντιμετώπισαν άλλοι Developers.

Επεκτάσεις και μονάδες

Κάποια από τα μέλη που συμμετέχουν στις κοινότητες υποστήριξης δημοσιεύουν δωρεάν επεκτάσεις για το framework και μονάδες που μπορείς να κατεβάσεις και να χρησιμοποιήσεις. Τέτοιες επεκτάσεις μπορεί να σας παρέχουν διασύνδεση με άλλες εφαρμογές μέσω κάποιου API.

2.2.2 ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΟΥ FRAMEWORK

Μαθαίνεις το Framework, όχι τη γλώσσα προγραμματισμού

Εάν χρησιμοποιείς ένα Framework και γνωρίζεις ελάχιστα για τη γλώσσα πίσω από αυτό, θα μάθεις πως δουλεύει το Framework και όχι την ίδια τη γλώσσα. Ο τρόπος που γράφουμε κώδικα jQuery είναι διαφορετικός από τον τρόπο που γράφουμε Javascript. Δηλαδή, αν γνωρίζουμε jQuery, δεν σημαίνει ότι κατανοούμε την Javascript. Γι' αυτό είναι σημαντικό να είμαστε γνώστες των προγραμματιστικών γλωσσών για την σωστή αξιοποίηση του Framework.

Περιορισμοί

Η βασική συμπεριφορά του Framework δεν μπορεί να τροποποιηθεί, υποδεικνύοντας ότι όταν χρησιμοποιείτε ένα Framework, πρέπει να σέβεσαι τους περιορισμούς και να εργαστείς με τον τρόπο που απαιτείται.

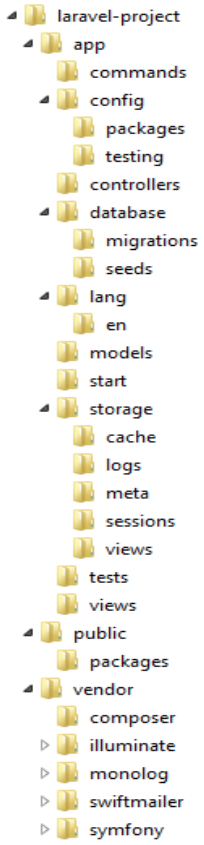
Ο κώδικας είναι δημόσιος

Δεδομένου ότι το Framework είναι άμεσα διαθέσιμο σε όλους, προσφέρεται επίσης σε άτομα με κακές προθέσεις. Μπορεί να μελετηθεί και οι ατέλειες του μπορούν να χρησιμοποιηθούν εναντίον μας. [7]

2.3 ΔΟΜΗ ΦΑΚΕΛΩΝ LARAVEL

Το Laravel αναφέρεται ως Full Stack Framework, διότι χειρίζεται τα πάντα, όπως το web που εξυπηρετεί στη διαχείριση βάσεων δεδομένων και το HTML. Ένα ολοκληρωμένο περιβάλλον ανάπτυξης ιστού μπορεί να προσφέρει μια καλύτερη εμπειρία στον προγραμματιστή.

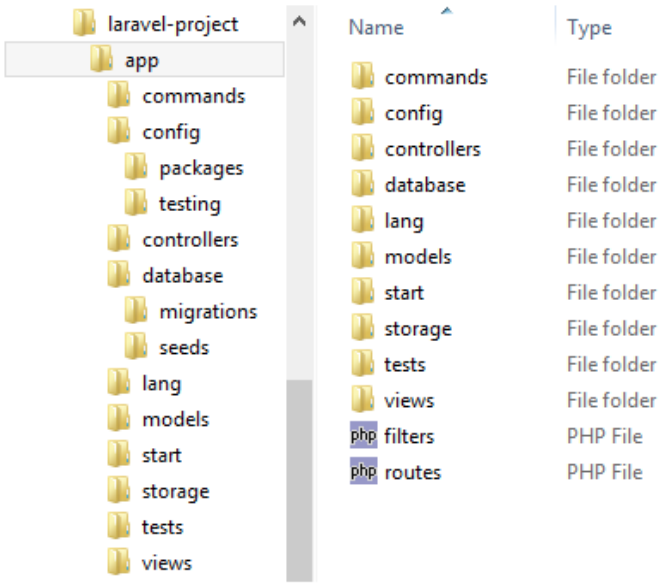
Ένα από τα ενδιαφέροντα χαρακτηριστικά του Laravel είναι ότι επιβάλλει κάποιους αρκετά σοβαρούς περιορισμούς στον τρόπο δομής των εφαρμογών σας στο διαδίκτυο. Παραδόξως, αυτοί οι περιορισμοί καθιστούν ευκολότερη τη δημιουργία εφαρμογών.



Εικόνα 1: Δομή φακέλων Laravel

Γονικοί φάκελοι	Σκοπός
/app/	Περιέχει τους ελεγκτές, τα μοντέλα, τις προβολές και τα στοιχεία ενεργητικού για την αίτησή σας. Εδώ υπάρχει το μεγαλύτερο μέρος του κώδικα μας.
/public/	Ο μόνος φάκελος που βλέπεις στον κόσμο όπως είναι. Αυτός είναι ο κατάλογος στον οποίο τοποθετούμε τον διακομιστή ιστού. Περιέχει το αρχείο index.php του bootstrap το οποίο ξεκινάει τον πυρήνα του Laravel. Ο δημόσιος κατάλογος μπορεί επίσης να χρησιμοποιηθεί για τη συγκράτηση όλων των στατικών στοιχείων που είναι προσβάσιμα από το κοινό, όπως CSS, αρχεία Javascript, εικόνες και άλλα αρχεία.
/vendor/	Ένα μέρος για όλους τους third-party codes. Σε μια τυπική εφαρμογή Laravel, αυτό περιλαμβάνει τον πηγαίο κώδικα του Laravel και plugins που περιέχουν πρόσθετες λειτουργίες.

Παραπάνω αναφερθήκαμε στην βασική δομή του Laravel. Τώρα θα αναλύσουμε την εσωτερική δομή του φάκελου /app/.



Εικόνα 2: Εσωτερική δομή /App/

Γονικοί φάκελοι	Σκοπός
/app/config	Περιέχει αρχεία ρυθμίσεων για την αλλαγή διαφόρων πτυχών του Framework. Τα περισσότερα από τα αρχεία ρυθμίσεων επιστρέφουν associative PHP arrays με τις επιλογές.
/app/config/auth.php	Διαμόρφωση για διάφορες ρυθμίσεις επιπέδου εφαρμογής, δηλαδή ζώνη ώρας, τοπική ρύθμιση, λειτουργία εντοπισμού σφαλμάτων και μοναδικό κλειδί κρυπτογράφησης.
/app/config/cache.php	Αν η εφαρμογή χρησιμοποιεί την προσωρινή αποθήκευση για να επιταχύνει το χρόνο απόκρισης, ρυθμίζουμε τις δυνατότητες τις από εδώ.
/app/config/database.php	Περιέχει σχετικές πληροφορίες διαμόρφωσης για τη βάση δεδομένων, δηλαδή προεπιλεγμένη μηχανή βάσης δεδομένων και πληροφορίες σύνδεσης.
/app/config/session.php	Διαμόρφωση που ελέγχει τον τρόπο διαχείρισης των session χρήστη από το Laravel, δηλαδή τον οδηγό για την λειτουργία, τη διάρκεια ζωής της περιόδου λειτουργίας των session.
/app/controllers	Περιέχει τις κατηγορίες ελεγκτών που χρησιμοποιούνται για την παροχή βασικής λογική εφαρμογής, την αλληλεπίδραση με τα μοντέλα δεδομένων και τη φόρτωση των αρχείων προβολής για την εφαρμογή σας.
/app/database/migrations/	Ο φάκελος migrations περιέχει κλάσεις PHP που επιτρέπουν στο Laravel να ενημερώνει το Σχήμα της τρέχουσας βάσης δεδομένων σας διατηρώντας συγχρόνως όλες τις εκδόσεις της βάσης δεδομένων
/app/database/seeds/	Ο φάκελος Seeds περιέχει αρχεία PHP που επιτρέπουν στον Artisan να συμπληρώνει πίνακες βάσεων δεδομένων με δεδομένα αναφοράς.
/app/models/	Ο φάκελος models περιέχει κλάσεις που αντιπροσωπεύουν τις πληροφορίες (δεδομένα) της εφαρμογής και τους κανόνες χειρισμού αυτών των δεδομένων.
/app/views/	Ο κατάλογος προβολών περιέχει τα αρχεία HTML που χρησιμοποιούνται από ελεγκτές ή δρομολογητές.

/app/routes.php	Είναι το αρχείο δρομολόγησης της εφαρμογής, το οποίο περιέχει κανόνες δρομολόγησης που λέει στο Laravel τον τρόπο σύνδεσης των εισερχόμενων αιτημάτων.

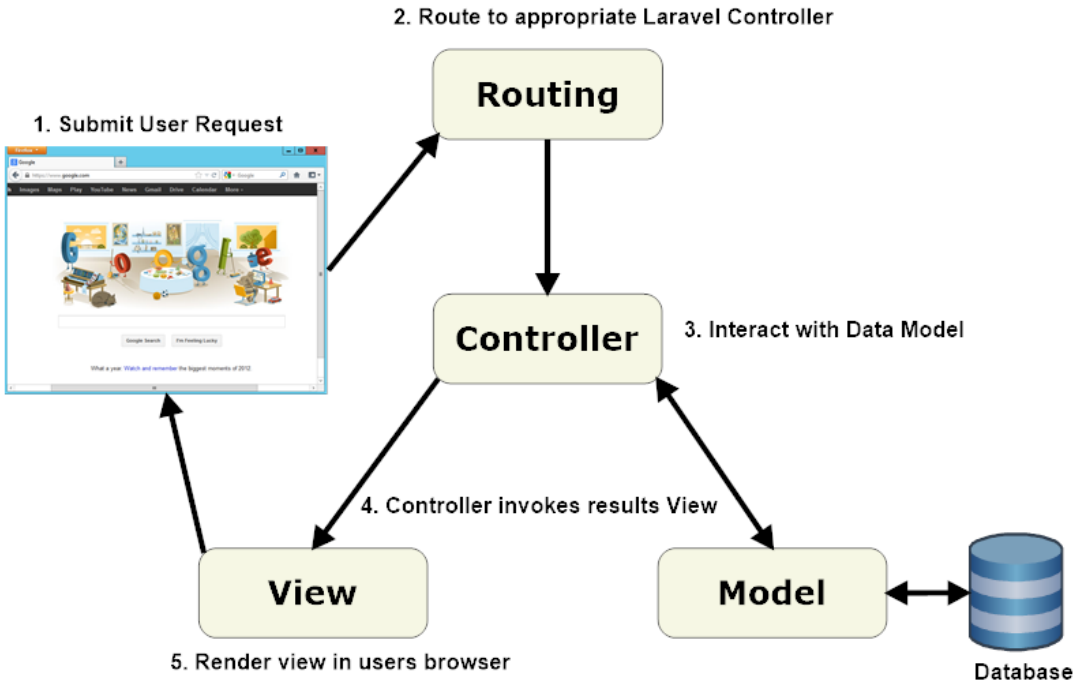
2.4 ΑΡΧΙΤΕΚΤΟΝΙΚΗ MVC

Το MVC (Model-View-Controller) είναι ένα αρχιτεκτονικό μοντέλο λογισμικού για την ανάπτυξη ιστοσελίδων. Είναι επίσης ένα από τα πιο συχνά χρησιμοποιούμενα βιομηχανικά πρότυπα για την δημιουργία κλιμακωτών και επεκτάσιμων έργων. Η MVC αρχιτεκτονική είναι πολύ διάσημη καθώς απομονώνει την λογική της εφαρμογής από το επίπεδο της διεπαφής του χρήστη. Σκοπός του είναι να χωρίζει μια εφαρμογή σε τρία βασικά λογικά στοιχεία: το μοντέλο, την προβολή και τον ελεγκτή. Κάθε ένα από αυτά είναι κατασκευασμένα για να χειρίζονται συγκεκριμένες πτυχές ανάπτυξης μιας εφαρμογής. Αυτή η λογική βοηθάει τον προγραμματιστή στην σωστή οργάνωση του κώδικα και κάνει πιο εύκολη την συντήρησή του. [8]

Το **μοντέλο (Model)** είναι οντότητες με τη μορφή δεδομένων στην ιστοσελίδα και ο τομέας στον οποίο είναι χτισμένο το λογισμικό μας. Τα μοντέλα βασίζονται σε πραγματικά στοιχεία όπως ένα πρόσωπο, ένας τραπεζικός λογαριασμός ή ένα προϊόν. Για παράδειγμα, σε ένα ηλεκτρονικό κατάστημα, τα προϊόντα (προϊόντα), οι Προσφορές και ο Χρήστης είναι τα μοντέλα. Αυτά συνδέονται με πίνακες βάσεων δεδομένων και χειρίζονται τις σχέσεις και τις λειτουργίες. Ένα προϊόν μπορεί να έχει κάποιες προσφορές και εκπτώσεις. Τα μοντέλα είναι περισσότερα από αποθηκευμένα δεδομένα. Το μοντέλο προϊόντος πρέπει να έχει ορισμένους περιορισμούς (αριθμός στοιχείων, κόστος, προσφορές υπό όρους). Ένα πρότυπο εκπληρώνει όλες τις προσδοκίες μιας οντότητας.

Η **προβολή (View)** είναι η οπτική αναπαράσταση ενός μοντέλου και αφορά το περιεχόμενο (σελίδες HTML) που προβάλλεται στον χρήστη. Για παράδειγμα, όταν επισκέπτεστε ένα ηλεκτρονικό κατάστημα, σας παρέχονται λίστες με στοιχεία (προϊόντα), προσφορές, καλαθιού αγορών κλπ. Όλο αυτό το περιεχόμενο είναι δεδομένα (HTML) για εμφάνιση στο πρόγραμμα περιήγησης. Αυτό τα δεδομένα είναι γνωστά ως προβολή στο MVC.

Ο **ελεγκτής (Controller)** είναι η γέφυρα μεταξύ μοντέλου και προβολής. Διαχειρίζεται το αίτημα του χρήστη με λογική. Για παράδειγμα, σε ένα ηλεκτρονικό κατάστημα, αναζήτηση και φιλτράρισμα προϊόντων, ενημέρωση του καλαθιού, χειρισμός του ελέγχου ταυτότητας χρήστη και ολοκλήρωση της πληρωμής είναι οι μερικοί ελεγκτές. Στο MVC, το περιεχόμενο HTML και η βάση δεδομένων δεν είναι άμεσα προσβάσιμα για τον εξωτερικό κόσμο. Αυτά εξυπηρετούνται μέσω ελεγκτών. [9]



Εικόνα 3: Αρχιτεκτονική MVC

2.5 AUTHENTICATION

Το Laravel κάνει την εφαρμογή της ταυτοποίησης πολύ απλή και εύκολη στην υλοποίησή της. Στην πραγματικότητα, σχεδόν όλα έχουν ρυθμιστεί για εμάς από την αρχή που κάνουμε εγκατάσταση το framework. Αυτό δεν μας σταματάει στο να κάνουμε τις δικές μας τροποποιήσεις στο αρχείο ρυθμίσεων ελέγχου ταυτότητας. Αυτό βρίσκεται στο config/auth.php.

2.6 ROUTING

Ένα από τα πιο αναμφισβήτητα χαρακτηριστικά του Laravel είναι η δρομολόγηση που μας παρέχει. Με λίγες μόνο γραμμές κώδικα, μπορούμε εύκολα και γρήγορα να φτιάξουμε τον «δρόμο», ο οποίος στην ουσία συνδέει τις διάφορες συναρτήσεις με τα εκάστοτε Views. Παρακάτω μπορούμε να δούμε ένα παράδειγμα:

```
Route::get('/', function() {
    return View::make('home');
});
```

Με τον παραπάνω κώδικα έχουμε πετύχει την δρομολόγηση στην αρχική σελίδα μιας εφαρμογής.

2.6 SESSIONS

Είναι μια περίοδος σύνδεσης που μπορεί να οριστεί και ως μέσο αποθήκευσης πληροφοριών από πλευράς διακομιστή. Τα Sessions είναι επιθυμητό να διατηρούνται σε όλη την αλληλεπίδραση του χρήστη με τον ιστότοπο.

Αντί να αποθηκεύει μεγάλες και συνεχώς μεταβαλλόμενες πληροφορίες μέσω των Cookies στο πρόγραμμα περιήγησης του χρήστη, αποθηκεύεται μόνο ένα μοναδικό αναγνωριστικό στην πλευρά του πελάτη που ονομάζεται "ID Session". Αυτό το αναγνωριστικό περιόδου σύνδεσης διαβιβάζεται στον Web Server κάθε φορά που το πρόγραμμα περιήγησης κάνει ένα αίτημα HTTP. Η εφαρμογή ζεύγει αυτή την περίοδο σύνδεσης με την εσωτερική βάση δεδομένων της και ανακτά τις αποθηκευμένες μεταβλητές για χρήση. [10]

2.7 CACHING

Είναι μια μέθοδος την οποία χρησιμοποιούμε για να μην επιβαρύνουμε το Server μας. Σε αυτήν αποθηκεύουμε στατικά δεδομένα, δηλαδή δεδομένα τα οποία δεν χρειάζεται να αναπαραχθούν, επεξεργαστούν και να υποστούν κάποια αλλαγή. Το αποτέλεσμα είναι ο server να μην δέχεται αιτήματα για το συγκεκριμένο περιεχόμενο. Τα δεδομένα αυτά είναι αποθηκευμένα σε μια προσωρινή μνήμη. Στο Laravel είναι έτοιμες όλες οι ρυθμίσεις για το Caching.

2.8 BLADE TEMPLATE

Το Blade είναι ένα απλό αλλά ισχυρό template engine που παρέχει το Laravel. Σε αντίθεση με άλλα δημοφιλή template engines, το Blade δεν μας εμποδίζει να χρησιμοποιήσουμε απλό κώδικα PHP μέσα στα Views. Στην πραγματικότητα, όλα τα Blade Views μεταγλωττίζονται σε απλό κώδικα PHP και αποθηκεύονται προσωρινά μέχρι να τροποποιηθούν, πράγμα που σημαίνει ότι το Blade προσθέτει ουσιαστικά μηδενική επιβάρυνση. Τα αρχεία προβολής με Blade μηχανισμό χρησιμοποιούν την επέκταση αρχείου .blade.php και αποθηκεύονται στο resources/views. [11]

Μία από τις πιο σημαντικές λειτουργίες του μηχανισμού Blade είναι ότι μας δίνει την δυνατότητα να δημιουργήσουμε μακέτες, μια μεγάλη βοήθεια να μην ξανά γράφουμε ίδιο κώδικα για άλλες σελίδες και ευκολία στην μαζική τροποποίηση ενός μέρους κώδικα.

2.9 OBJECT RELATIONAL MAPPING

Το ORM είναι μια τεχνική που μας επιτρέπει να αναζητήσουμε και να επεξεργαστούμε δεδομένα από μια βάση δεδομένων χρησιμοποιώντας ένα αντικειμενοστραφή παράδειγμα. Όταν μιλάμε για το ORM, οι περισσότεροι άνθρωποι αναφέρονται σε μια βιβλιοθήκη που υλοποιεί αυτήν την τεχνική, έτσι βγήκε και το όνομα αυτό.

Μία βιβλιοθήκη ORM είναι μια εντελώς συνηθισμένη βιβλιοθήκη γραμμένη στη γλώσσα επιλογής μας που ενσωματώνει τον κώδικα που απαιτείται για τον χειρισμό των δεδομένων, έτσι ώστε πλέον να μην χρησιμοποιούμε πια την SQL. Μπορούμε να αλληλοεπιδράσουμε άμεσα με ένα αντικείμενο στην ίδια γλώσσα. Συγκεκριμένα το Laravel έχει το Eloquent ORM. [12]

3 ΑΝΑΛΥΣΗ ΛΕΙΤΟΥΡΓΙΩΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Σε αυτή την ενότητα ερμηνεύονται συνολικά τα κύρια κομμάτια που υλοποιήσαμε στην πλατφόρμα ηλεκτρονικού εμπορίου. Παρακάτω θα αναλύσουμε τα εξής:

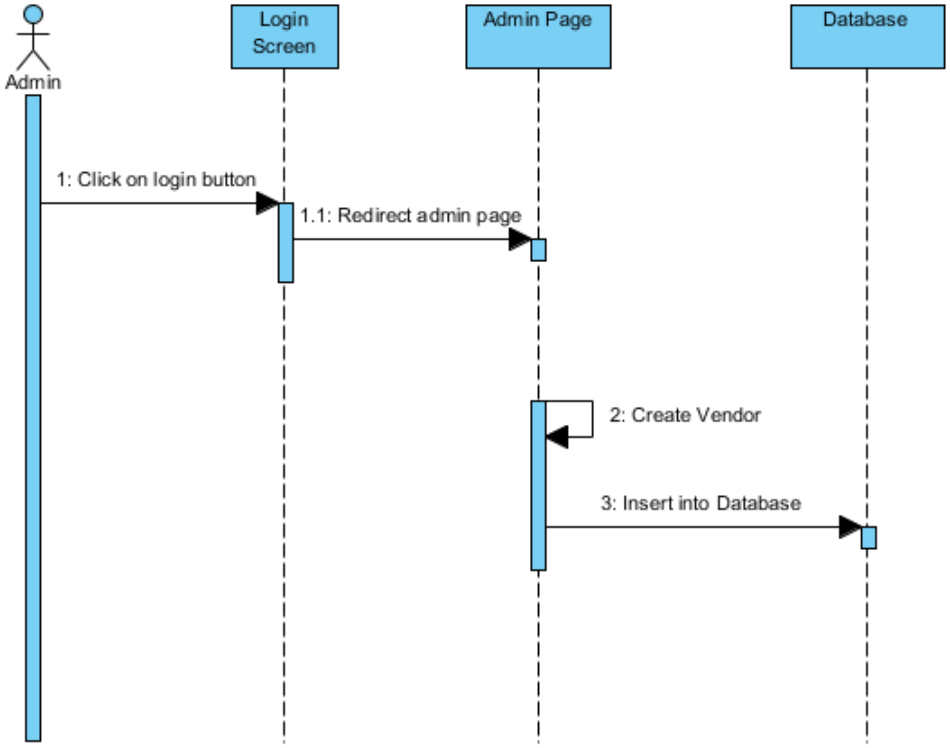
- Κατηγορίες χρηστών
- Λειτουργία του Shopping Cart
- Σύνδεση με API για την πραγματοποίηση αγορών
- Βάση δεδομένων για την εφαρμογή
- Μηχανή αναζήτησης
- Προστασία σελίδων από χρήστες

3.1 **Κατηγορίες Χρηστών**

Στο πλατφόρμα μας έχουμε τρείς κατηγορίες χρηστών: τον Διαχειριστή (Administrator), τον πωλητή (Vendor) και τον απλό χρήστη ή αλλιώς, αγοραστή (User).

3.1.1 **Διαχειριστής**

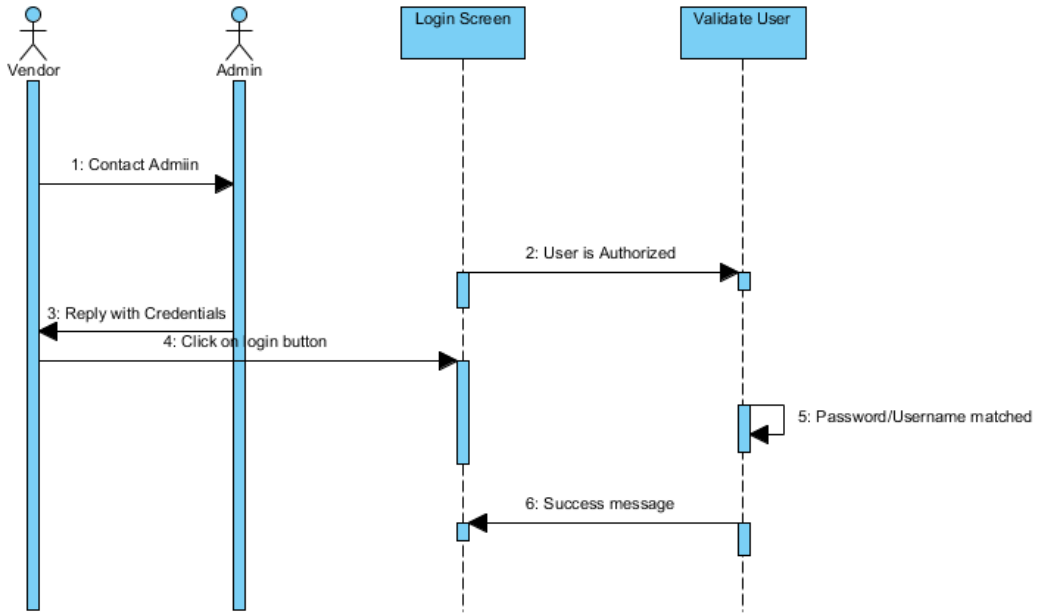
Ο Διαχειριστής είναι υπεύθυνος για την δημιουργία λογαριασμού κάθε πωλητή μετά από επικοινωνία, καθώς και τον έλεγχο των προϊόντων που δημοσιοποιούν οι πωλητές.



Διάγραμμα 3: Ακολουθία Δημιουργίας Vendor

3.1.2 **Πωλητής**

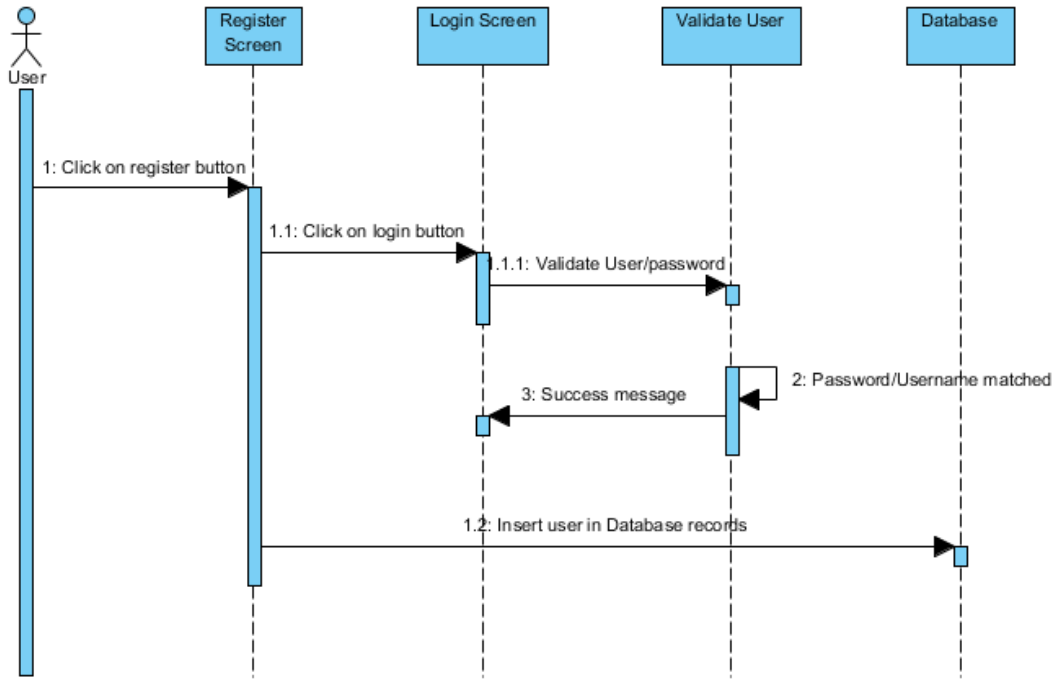
Εφόσον ο διαχειριστής αναβαθμίσει τον λογαριασμό του πωλητή, αυτός εισέρχεται στον λογαριασμό και συμπληρώσει τα προσωπικά του στοιχεία. Μετά έχει το δικαίωμα να δημοσιοποιήσει τα προϊόντα του μαζί με τις φωτογραφίες τους που θέλει να πουλήσει. Έχει την δυνατότητα να ενημερώσει ή ακόμα και να διαγράψει προϊόντα που βρίσκονται προς πώληση. Τέλος, είναι υπεύθυνος για την αποστολή των προϊόντων στον αγοραστή.



Διάγραμμα 4: Ακολουθία εισόδου Vendor στην πλατφόρμα

3.1.3 **Χρήστης**

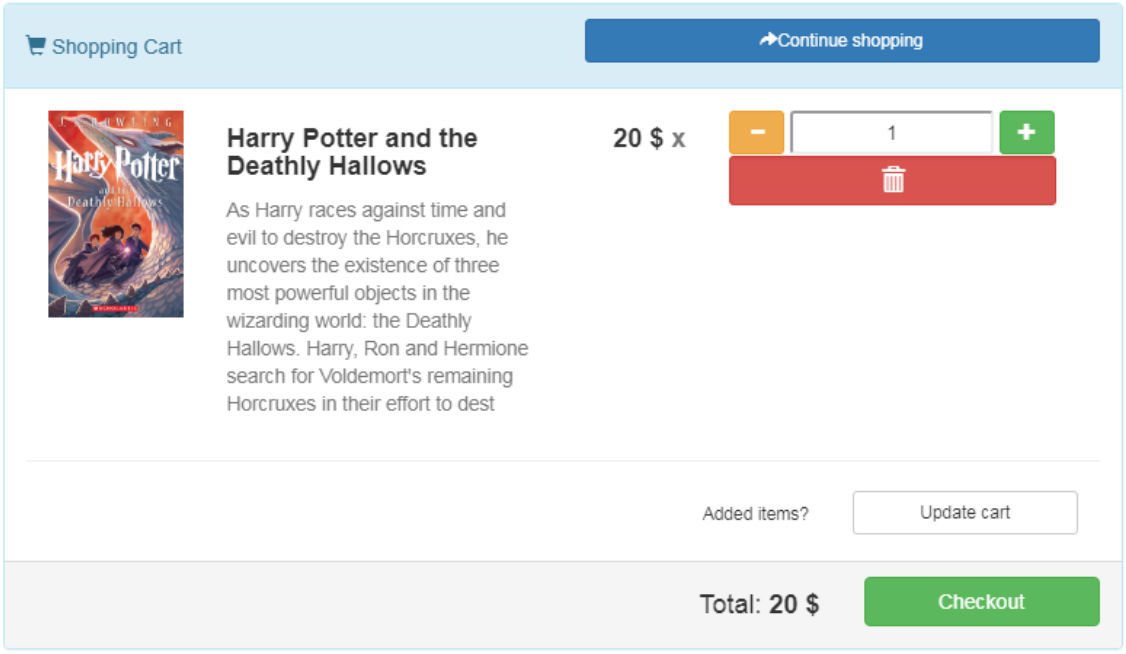
Ο χρήστης δημιουργεί λογαριασμό από μόνος του για να αγοράσει κάποιο προϊόν από το κατάστημα. Έπειτα κάνει τις ανάλογες αναζητήσεις για τα προϊόντα που τον ενδιαφέρουν.



Διάγραμμα 3: Ακολουθία Εγγραφής / Εισόδου χρήστη

3.2 ΛΕΙΤΟΥΡΓΙΑ SHOPPING CART

Για την λειτουργία κάθε ηλεκτρονικού καταστήματος είναι απαραίτητο η δημιουργία ενός εικονικού καλαθιού στο οποίο ο χρήστης προσθέτει τα προϊόντα που θέλει να αγοράσει, επιτρέπει στους καταναλωτές να δουν τι έχουν προσθέσει στο καλάθι καθώς και να τροποποιήσουν τις επιλογές τους.

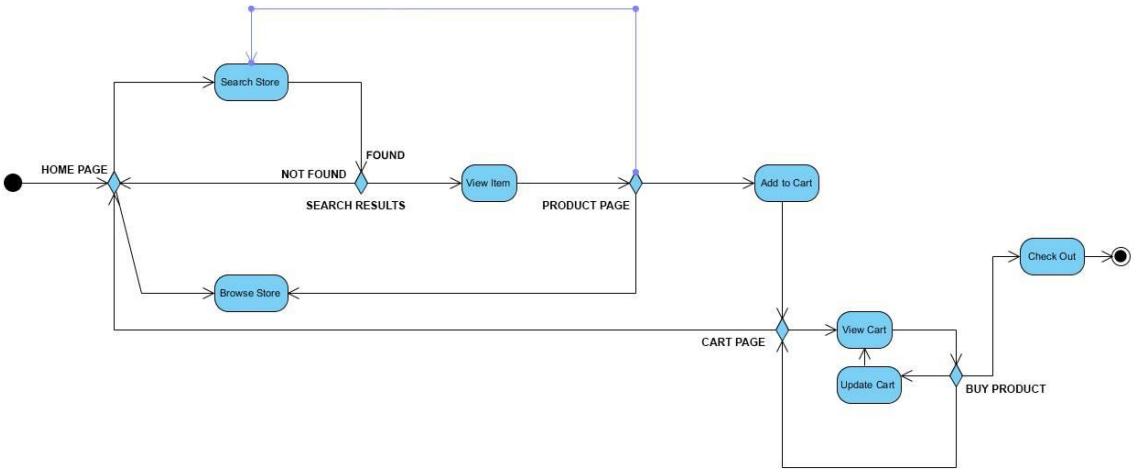


Εικόνα 4: Καλάθι αγορών

Στο δικό μας καλάθι δίνεται η δυνατότητα στον χρήστη να αυξομειώσει την ποσότητα καθώς και να διαγράψει την αγορά που θέλει να πραγματοποιήσει. Παρακάτω ακολουθεί η ανάλυση του κώδικα των λειτουργιών τοποθέτηση προϊόντων στο καλάθι, αύξηση, μείωση και διαγραφή των προϊόντων.

3.2.1 **Activity diagram shopping cart**

Παρακάτω παρουσιάζουμε με ένα διάγραμμα ενεργειών, την λειτουργία του καλαθιού.



Διάγραμμα 4: Activity diagram shopping cart

3.2.2 **ΤΟΠΟΘΕΤΗΣΗ ΠΡΟΙΟΝΤΩΝ ΣΤΟ ΚΑΛΑΘΙ**

Για την τοποθέτηση ενός προϊόντος στο καλάθι, έχουμε γράψει μια συνάρτηση, η οποία ο ρόλος της είναι να περιμένει το αίτημα από τον χρήστη για την προσθήκη του προϊόντος. Όταν συμβεί αυτό, αποθηκεύουμε στην μεταβλητή product το μοναδικό κλειδί που χαρακτηρίζει κάθε προϊόν id, το οποίο αντιστοιχεί σε ένα αριθμό. Αμέσως μετά κάνουμε έλεγχο, αν υπάρχει στα Sessions κάποιο καλάθι με προϊόντα. Αν υπάρχει, τα προσθέτουμε και αυτά μαζί με το καινούριο που επέλεξε ο χρήστης. Επόμενο βήμα είναι να αποθηκεύσουμε το ενημερωμένο καλάθι στα Sessions.

Παρακάτω μπορούμε να δούμε το κομμάτι κώδικα, που είναι υπεύθυνο για αυτήν την λειτουργία:

```
public function AddToCart(Request $request, $id){
    $product = Product::find($id);
    $emptyCart = Session::has('cart') ? Session::get('cart') : null;
    $cart = new Cart($emptyCart);
    $cart->addProducts($product, $product->id);

    $request->session()->put('cart', $cart);
    return redirect()->back();
}
```

3.2.3 **ΑΥΞΗΣΗ - ΜΕΙΩΣΗ ΠΟΙΟΝΤΩΝ**

Στο παρακάτω κώδικα για την αύξηση και μείωση των προϊόντων τοποθετούμε στην μεταβλητή emptyCart το αποθηκευμένο καλάθι που υπάρχει στα Sessions. Έπειτα δημιουργούμε μια μεταβλητή Cart και αποθηκεύουμε τα δεδομένα που περιείχε το παλιό καλάθι, με τις συνάρτησεις IncreaseProduct και DecreaseProduct για αύξηση και μείωση των προϊόντων αντίστοιχα. Μετά πραγματοποιούμε την επιθυμητή λειτουργία. Σε περίπτωση που στην μείωση του προϊόντος, το προϊόν φτάσει να έχει μηδενική τιμή, φροντίζουμε τα το αφαιρέσουμε από το καλάθι. Με την ολοκλήρωση της διαδικασίας προσθέτουμε πάλι το καινούριο μας καλάθι στα Sessions.

```
public function IncreaseProduct($id){
    $emptyCart = Session::has('cart') ? Session::get('cart') : null;
    $cart = new Cart($emptyCart);
    $cart->IncreaseProduct($id);

    if(count($cart->products) > 0)
       Session::put('cart', $cart);
    return redirect()->route('shoppingCart');
  }

public function DecreaseProduct($id){
    $emptyCart = Session::has('cart') ? Session::get('cart') : null;
    $cart = new Cart($emptyCart);
    $cart->DecreaseProduct($id);

    if(count($cart->products) > 0)
       Session::put('cart', $cart);
```

```
        else
            Session::forget('cart');
        return redirect()->route('shoppingCart');
    }
```

3.2.4 **ΔΙΑΓΡΑΦΗ ΠΟΪΟΝΤΩΝ**

Για την διαγραφή των προϊόντων χρησιμοποιούμε την συνάρτηση RemoveProduct. Έπειτα ακολουθεί έλεγχος αν ήταν το τελευταίο προϊόν και σε περίπτωση που ήταν, διαγράφουμε το καλάθι με την συνάρτηση forget. Διαφορετικά ακολουθεί η αποθήκευση του στα Sessions.

```
    public function RemoveProduct($id){
        $emptyCart = Session::has('cart') ? Session::get('cart') : null;
        $cart = new Cart($emptyCart);
        $cart->RemoveProduct($id);

        if(count($cart->products) > 0)
            Session::put('cart', $cart);
        else
            Session::forget('cart');
        return redirect()->route('shoppingCart');
    }
```

3.3 ΣΥΝΔΕΣΗ ΜΕ API ΓΙΑ ΤΗΝ ΠΡΑΓΜΑΤΟΠΟΙΗΣΗ ΑΓΟΡΩΝ

Το περιβάλλον εργασίας εφαρμογών (API) είναι ένα σύνολο ρουτινών, πρωτόκολλων και εργαλείων για την κατασκευή εφαρμογών λογισμικού. Ένα API καθορίζει τον τρόπο αλληλεπίδρασης των στοιχείων του λογισμικού. Επιπλέον, τα API χρησιμοποιούνται κατά τον προγραμματισμό των γραφικών στοιχείων διεπαφής χρήστη (GUI). Ένα καλό API διευκολύνει την ανάπτυξη ενός προγράμματος παρέχοντας όλα τα δομικά στοιχεία. Στην πτυχιακή αυτή χρησιμοποιήσαμε για τις αγορές των καταναλωτών το API του STRIPE. [13]

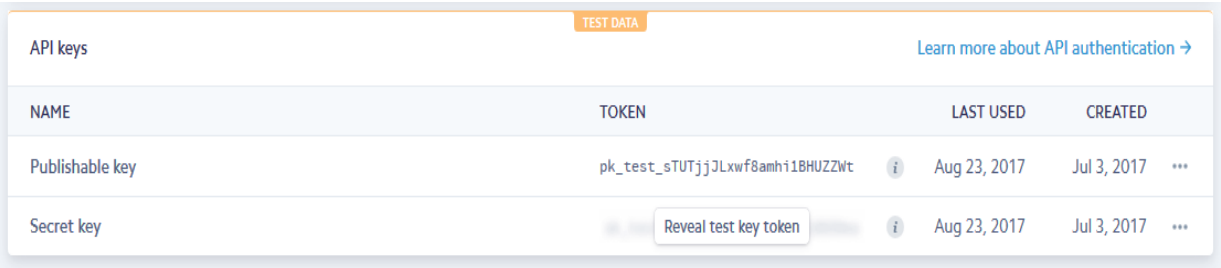
3.3.1 STRIPE

Το Stripe μας παρέχει ότι χρειαζόμαστε για να συνδέσουμε την πλατφόρμα μας με ένα API που με τις κατάλληλες ρουτίνες διασφαλίζει στον χρήστη ότι όλα τα στοιχεία του κατά την συναλλαγή και μετά την εκπλήρωση της θα είναι ασφαλή.

3.3.2 STRIPE Λογαριασμός

Για να γίνουν οι αγοραπωλησίες μέσω του Stripe, οι μαγαζάτορες καλούνται να δημιουργήσουν λογαριασμό στο Stripe. Έτσι από τη διαχείριση του siteθα μπορούν να βλέπουν χρήσιμες πληροφορίες για τις αγορές, όπως το πότε έγιναν, από ποιον έγιναν, πόσο ήταν το ποσό κλπ.

Επίσης, δίνονται 2 σετ από κλειδιά. Ένα σετ για δοκιμές (test) και ένα για πραγματικά δεδομένα (live). Το κάθε ένα από αυτά τα σετ, περιέχουν ένα μυστικό κλειδί (secret key) και ένα δημόσιο κλειδί (public key). Στην παρούσα κατάσταση, εφόσον η εφαρμογή δεν είχε σκοπό να χρησιμοποιηθεί με αληθινά δεδομένα, χρησιμοποιήσαμε μόνο τα δοκιμαστικά κλειδιά. Παρακάτω θα δούμε τα API keys του Stripe.



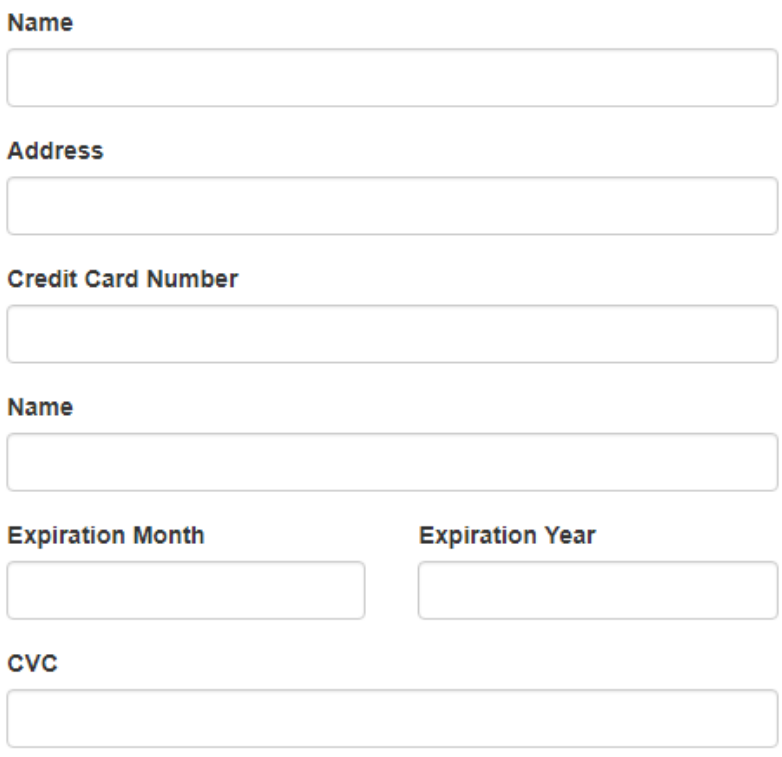
Εικόνα 5: API keys Stripe

3.3.3 STRIPE COLLECTING DATA PROCESS FLOW

Για να επιτευχθούν οι αγορές, συμβουλευτήκαμε το αναλυτικό documentation του Stripe. Υπάρχει μια συγκεκριμένη διαδικασία όπου το Stripe χρεώνει έναν πελάτη, όπου μπορούμε να βρούμε με ευκολία ανά βήματα.

Όσον αφορά τη φόρμα που θα συμπληρώνει ο πελάτης, μας νοιάζουν 4 πράγματα: ο αριθμός της πιστωτικής κάρτας, τον χρόνο και μήνα λήξης της και τέλος τον κωδικό ασφαλείας CVC, όπου έχουν όλες οι κάρτες στο πίσω μέρος τους. Όταν πάρουμε αυτά τα στοιχεία από τον πελάτη, τα στέλνουμε στο Stripe και αυτό κάνει την απαραίτητη επιβεβαίωση.

Οπότε για αρχή πρέπει να πάρουμε αυτά τα στοιχεία από τον χρήστη. Παρακάτω θα δούμε την φόρμα που καλείται να συμπληρώσει ο πελάτης, έτσι ώστε να συλλέξουμε τα στοιχεία του.



Εικόνα 6: Φόρμα Παραγγελίας

Εκτός από τα στοιχεία που προαναφέραμε, ζητάμε επίσης όνομα και διεύθυνση του πελάτη. Για την φόρμα αυτήν, χρησιμοποιούμε 2 JavaScript αρχεία: Το 1ο είναι υπεύθυνο να συλλέξει τα παραπάνω στοιχεία, καθώς χειρίζεται αυτήν την φόρμα, ενώ το 2ο χρειάζεται για την επικοινωνία με το Stripe. Για το 2ο αρχείο δεν έχουμε, παρά να συμπεριλάβουμε το CDN του Stripe στον κώδικά μας.

Αν και η φόρμα γίνεται με POST μέθοδο, δεν την στέλνουμε κατευθείαν στον server μας, δηλαδή στην Laravel εφαρμογή μας.

Όπως είπαμε, χρησιμοποιούμε JavaScript για να συλλέξουμε αυτά τα στοιχεία για να τα στείλουμε στο Stripe. Έπειτα, η πλατφόρμα θα χειριστεί το response του Stripe, αφού πρώτα επικοινωνήσουμε με το Stripe service, καθώς αυτό είναι υπεύθυνο για την επικύρωση των δεδομένων. Έτσι, εφόσον γίνει η επιβεβαίωση των δεδομένων, με την σειρά του το Stripe, μας στέλνει ένα Token το οποίο θα χρησιμοποιήσουμε για να κάνουμε την χρέωση. Παρακάτω θα αναλύσουμε τα κύρια κομμάτια κώδικα που περιέχει το JavaScript αρχείο, το οποίο χειρίζεται την φόρμα.

Εδώ θέτουμε το public key που μας παρέχει το Stripe API. Το χρειαζόμαστε για να αναγνωρίσει τη πλατφόρμα μας και να ξεκινήσει την επικοινωνία με το Stripe.

```
Stripe.setPublishableKey('public_key');
```

Έπειτα πρέπει να συλλέξουμε τις πληροφορίες της κάρτας.

```
Stripe.card.createToken({
    number: $('#card-number').val(),
    cvc: $('#card-cvc').val(),
    exp_month: $('#card-expiry-month').val(),
    exp_year: $('#card-expiry-year').val(),
    name: $('#card-name').val()
}, stripeResponseHandler);
```

Στο σημείο αυτό παρατηρούμε πως, εκτός το ότι συλλέγουμε τις πληροφορίες από τον χρήστη, επίσης φτιάχνει το Token που προαναφέραμε. Το 2ο argument που περνάμε στην φόρμα είναι το stripeResponseHandler, το οποίο μέσω μιας συνάρτησης, ελέγχει για τυχόν λάθη στις πληροφορίες της πιστωτικής κάρτας που μπορεί να πληκτρολογήσει ο πελάτης.

Ας δούμε την συνάρτηση του stripeResponseHandler.

```
function stripeResponseHandler(status, response) {
   if (response.error) {
      $('charge-error').removeClass('hidden');
      $('charge-error').text(response.error.message);
      $form.find('button').prop('disabled', false);
   } else {    //token was created!
      var token = response.id; //get the token ID
      $form.append($('<input type="hidden" name="stripeToken" />').val(token));
      $form.get(0).submit();  // Submit the form:
   }
}
```

Βλέπουμε στο 1ο κομμάτι της συνάρτησης πως αν όντως προκύψουν λάθη, τότε θα τα αναδείξει στον χρήστη. Ειδάλλως στο 2ο κομμάτι, θα δημιουργήσει ένα Token μιας χρήσης, το οποίο θα το προσθέσει στην φόρμα πληρωμής σε ένα κρυφό πεδίο. [14]

3.3.4 **STRIPE CHARGE PROCESS FLOW**

Εφόσον το Stripe μας επιστρέψει το Token, είμαστε έτοιμοι να κάνουμε την χρέωση στον πελάτη. Η χρέωση θα πρέπει να γίνει από την δικιά μας πλευρά, δηλαδή θα γίνει με PHP κώδικα. Η διαδικασία συνεχίζεται ως εξής:

Στην αρχή, θέτουμε το secret key που μας παρέχει το API του Stripe.

```
Stripe::setApiKey(secret_key);
```

Εδώ πραγματοποιείται η χρέωση

```
charge = Charge::create(array(
        "amount" => $cart->totalPrice * 100,
        "currency" => "usd",
        "source" => $request->input('stripeToken'),
        "description" => "Test Charge..."
));
```

Στο πεδίο του “amount”, παρατηρούμε πως πολλαπλασιάζουμε την μεταβλητή μας με το 100. Αυτό γίνεται διότι το Stripe χρησιμοποιεί τα σεντς ως προεπιλεγμένη μονάδα μέτρησης. Οπότε για να φαίνονται σωστά οι πωλήσεις, κάνουμε αυτήν την ενέργεια.

Στο πεδίο του “currency” επιλέγουμε το συνάλλαγμα. Στο documentation του Stripe, έχει ως προεπιλογή το USD, το οποίο είναι τα δολάρια. Θα μπορούσαμε να το κάνουμε σε EU το οποίο είναι το ευρώ, αλλά εφόσον η εφαρμογή δημιουργήθηκε καθαρά για εκπαιδευτικούς σκοπούς, το αφήσαμε ως έχει.

Στο πεδίο του “source” το Stripe περιμένει να συλλέξει το Token που δημιουργήσαμε πιο πριν. Το Token μαζί με το secret key, θα τα στείλουμε στο Stripe, όχι μόνο για επιβεβαίωση των δεδομένων, αλλά και να κάνει την αγορά.

Τέλος στο πεδίο του “description”, μπορούμε να θέσουμε διάφορες πληροφορίες όπως πχ το ID του πελάτη.

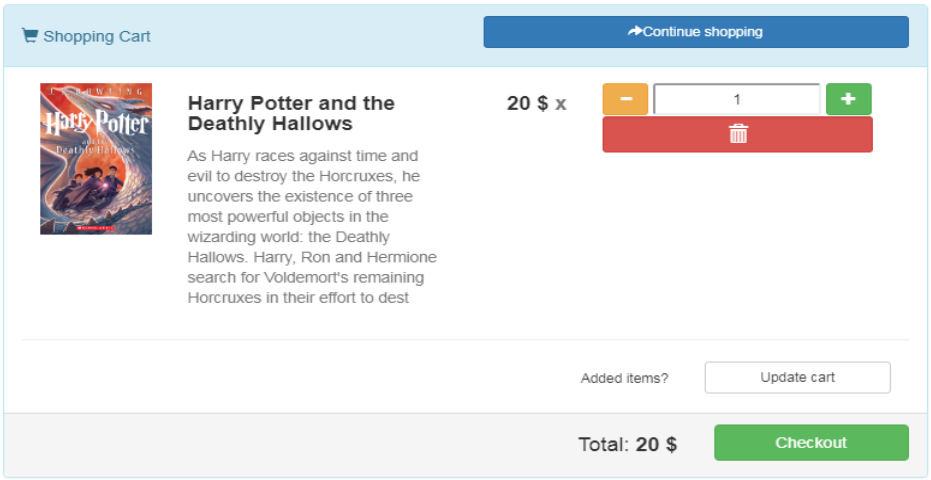
3.3.5 **Παράδειγμα μιας πώλησης με βήματα**

Εφόσον αναλύσαμε το διαδικασία πληρωμής/αγοράς με το Stripe, τώρα με μια σειρά από φωτογραφίες θα δείξουμε το τα βήματα και τις ενέργειες που γίνονται ώστε να αγοραστεί ένα προϊόν.

Καθώς ψάχναμε για βιβλία στην πλατφόρμα μας, βρήκαμε το συγκεκριμένο τίτλο που μας ενδιέφερε.

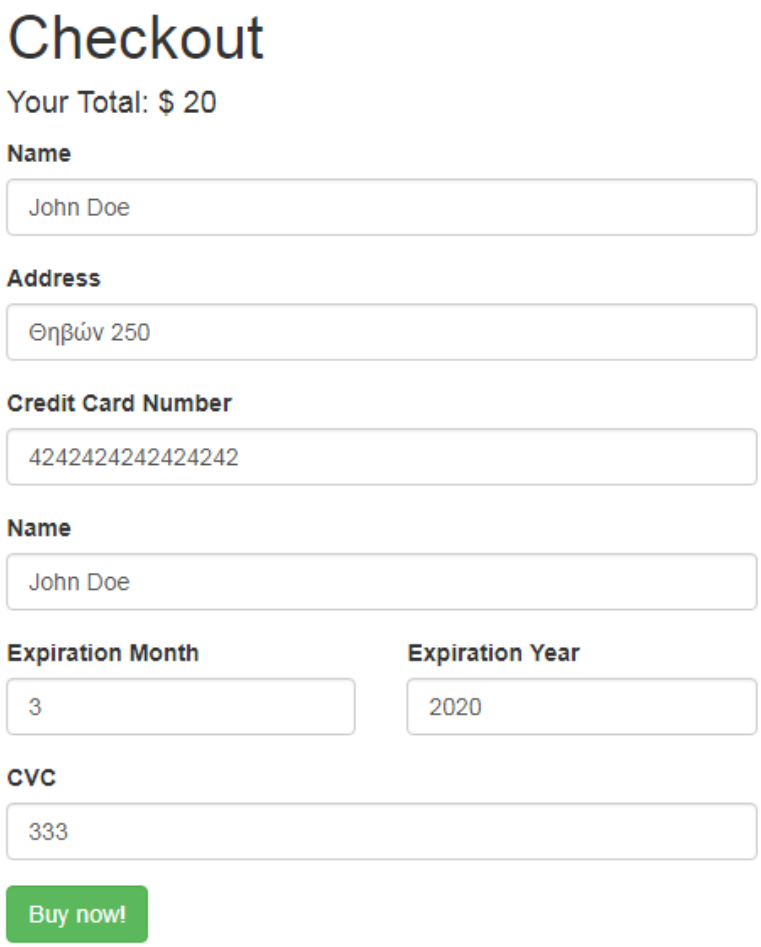
Εικόνα 7: Προϊόν καταστήματος

Εδώ ο πελάτης μπορεί να δει διάφορες πληροφορίες για το συγκεκριμένο προϊόν. Αφού θέλει να το αγοράσει, πατάει στο κουμπί Add to Cart. Αμέσως θα δούμε πως το προϊόν, βρίσκεται πλέον στο καλάθι.



Εικόνα 8: Καλάθι αγορών

Καθώς θα πατήσουμε στο κουμπί Checkout, θα οδηγηθούμε στην φόρμα στην οποία θα συμπληρώσουμε τα στοιχεία μας.



Εικόνα 9: Φόρμα συμπλήρωσης στοιχείων πιστωτικής κάρτας

Όλα τα στοιχεία είναι ψεύτικα και συμπληρώθηκαν μόνο για επίδειξη της εφαρμογής. Το μόνο που πρέπει να προσέξουμε, είναι στα στοιχεία τα οποία χειρίζεται το JavaScript αρχείο, δηλαδή τα στοιχεία της πιστωτικής κάρτας.

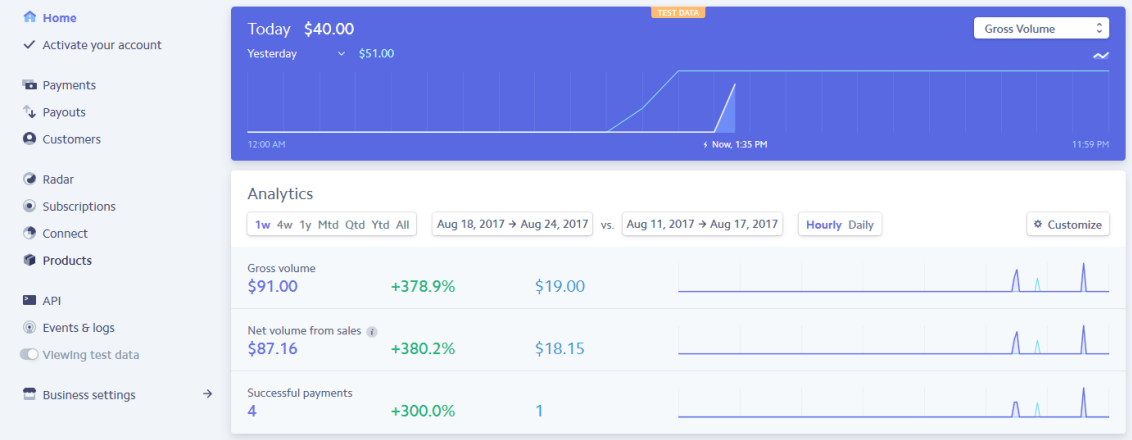
Εφόσον χρησιμοποιούμε ψεύτικα στοιχεία, το Stripe μας παρέχει διάφορες δοκιμαστικές κάρτες οι οποίες θα λειτουργούν κανονικά ως αληθινές. Μία από αυτές είναι η 4242424242424242 [15] και προφανώς θα λειτουργεί μόνο για το δοκιμαστικό περιβάλλον. Αν βάλουμε κάτι διαφορετικό, θα μας εμφανίσει σφάλμα. Επίσης, όσον αφορά την ημερομηνία λήξης, δεν έχουμε κάποιον περιορισμό, απλά τα στοιχεία θα πρέπει να είναι λογικά. Δηλαδή δεν μπορούμε να γράψουμε πως λήγει το 2016 ενώ εμείς βρισκόμαστε το 2017! Το ίδιο ισχύει και για το CVC.

Εάν κάνουμε inspect element στην φόρμα, μπορούμε να παρατηρήσουμε το Token που έχει δημιουργηθεί.

Εικόνα 10: Κώδικας Φόρμας

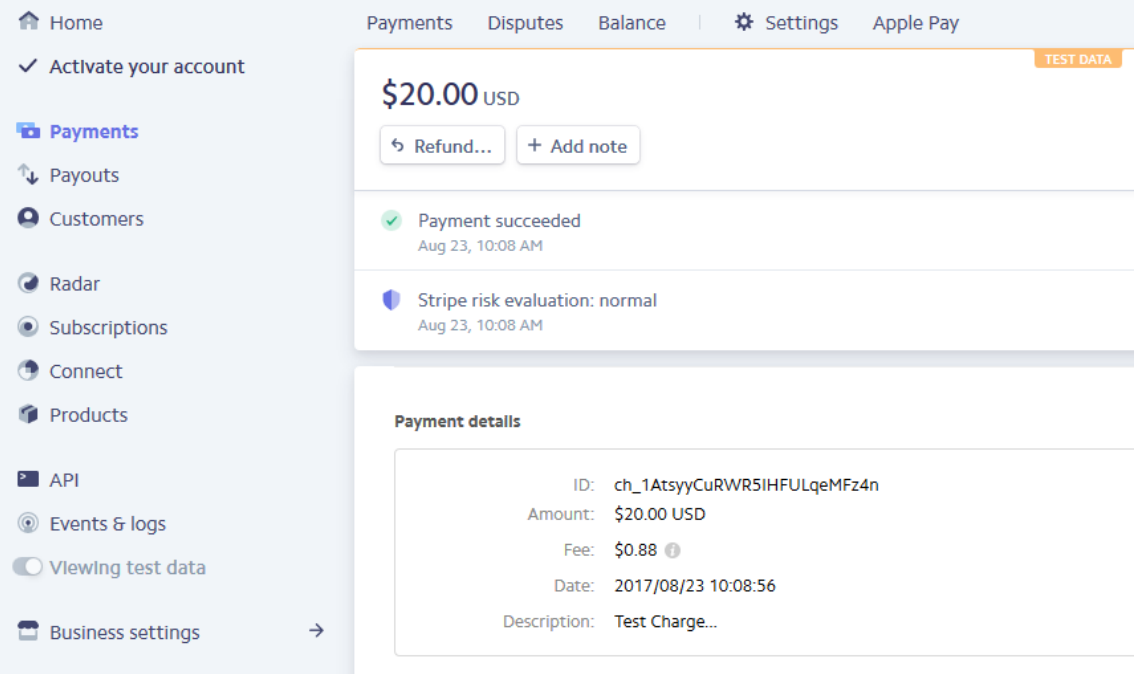
Αυτό είναι το Token στο οποίο αναφερόμασταν, όπου μαζί με το secret key, πραγματοποιείται η αγορά. Το επόμενο βήμα, εφόσον είναι επιτυχής η αγορά μας, μας ανακατευθύνει στην αρχική σελίδα με ένα επιτυχές μήνυμα.

Ας δούμε όμως ο πωλητής τι βλέπει στο Stripe λογαριασμό του.



Εικόνα 11: Stripe Dashboard

Εδώ ο πωλητής μπορεί να δει διάφορες κινήσεις με στατιστικά που έχουν γίνει στον λογαριασμό του.



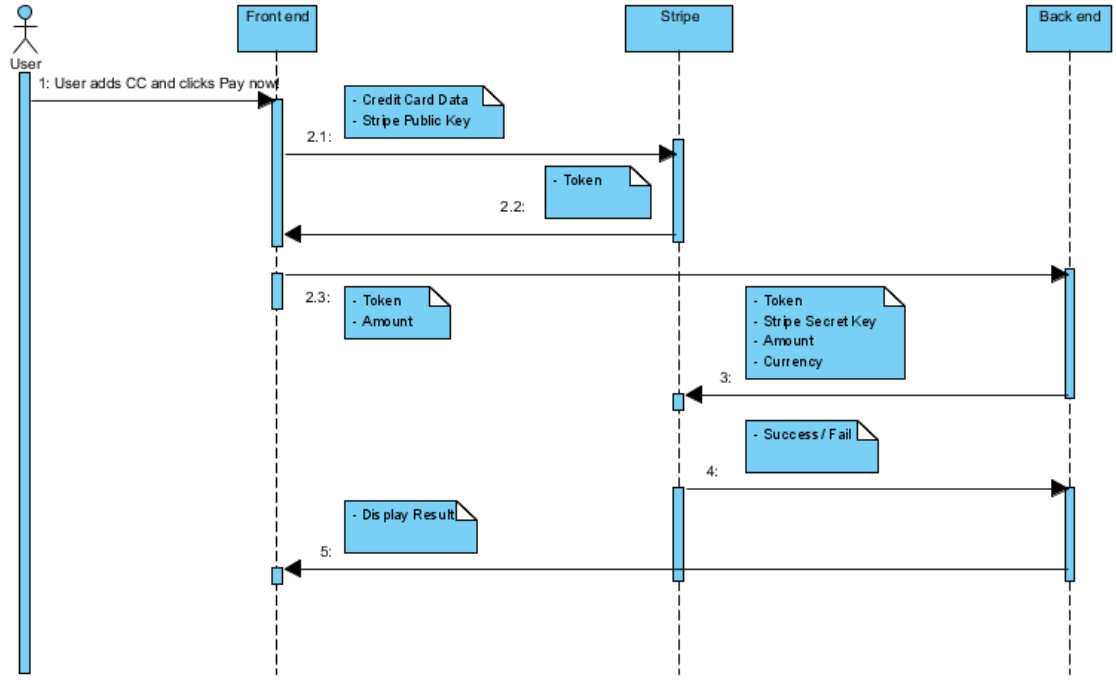
Εικόνα 12: Stripe Dashboard/Payments

Ενώ εδώ μπορεί να δει την αγορά που έκανε ο πελάτης.

Μπορούμε να διακρίνουμε το πεδίο που ονομάζεται Payment details, στο οποίο αναφέρονται λεπτομερώς βασικές πληροφορίες για την συγκεκριμένη αγοραπωλησία. Διακρίνουμε το Fee: $0.88 όπου είναι η προμήθεια που κρατάει το Stripe ως υπηρεσία.

Το ID από την άλλη είναι ένα αναγνωριστικό πεδίο, το οποίο χρησιμοποιείται κυρίως για ταυτοποίηση της αγοράς. Η πλατφόρμα αποθηκεύει αυτό το ID στην βάση της, στον πίνακα Orders.

Παρακάτω μπορούμε να δούμε αναλυτικά την ροή του Stripe με ένα διάγραμμα:

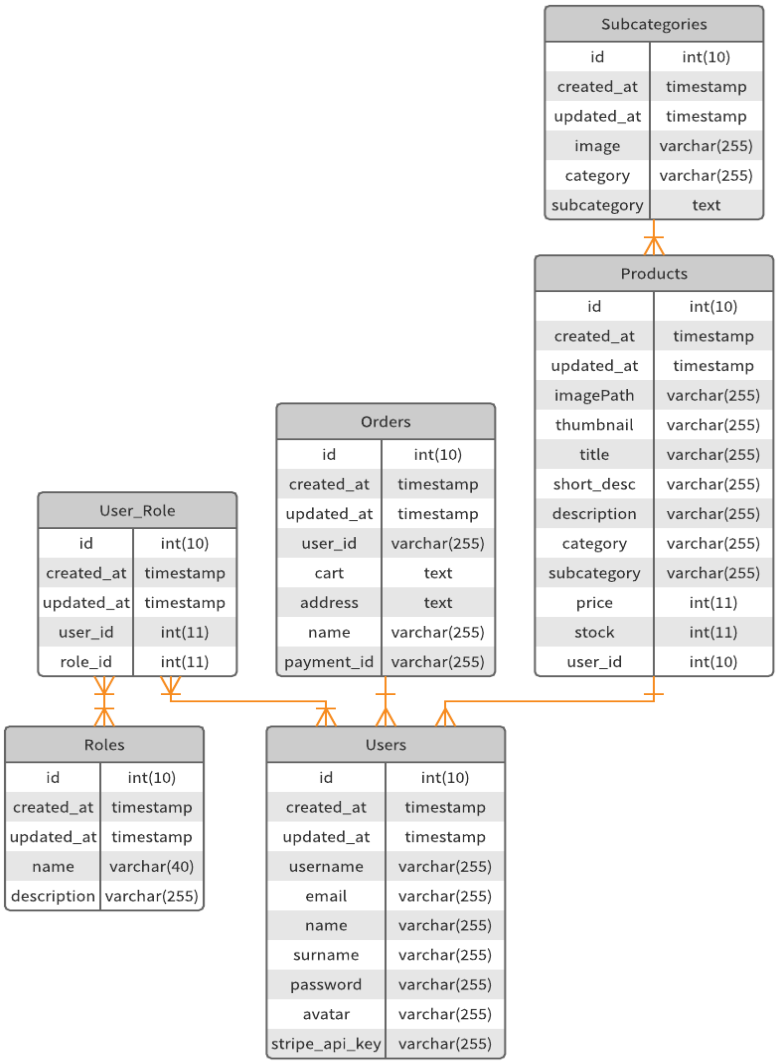


Διάγραμμα 5: Sequence diagram Stripe flow payment process

3.4 ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΤΗΝ ΕΦΑΡΜΟΓΗ

Κάθε μία web εφαρμογή που έχει αλληλεπιδράσεις με έναν server, αποθηκεύει τα δεδομένα της σε μία βάση. Ο σχεδιασμός της βάσης είναι το σημαντικότερο κομμάτι στον σχεδιασμό της εφαρμογής διότι έτσι επιτυγχάνουμε μια σωστή και ομαλή λειτουργία, όλων των λειτουργιών που θέλουμε να έχει η εφαρμογή μας.

Παρακάτω θα δούμε το πώς έχει σχεδιαστεί η βάση δεδομένων αυτής της πλατφόρμας και όπου κριθεί απαραίτητο, θα γίνει η ανάλογη ανάλυση των πινάκων:



Εικόνα 13: Ανάλυση βάσης δεδομένων

Αρχικά θα ξεκινήσουμε από τους χρήστες, μιας που είναι ένα table που μας παρέχει από την αρχή το Laravel και θεωρείται το πιο βασικό στην εφαρμογή μας.

Ο πίνακας των χρηστών περιέχει τις πιο σημαντικές πληροφορίες που θα έπρεπε να έχει ένας άνθρωπος, όπως όνομα, επώνυμο, ένα e-mail που κυρίως χρειάζεται για την εγγραφή του στην πλατφόρμα, καθώς και ένα username. Επίσης έχει ένα πεδίο avatar, στο οποίο θα αποθηκεύεται η φωτογραφία που θα επιλέξει να έχει ως avatar στο προφίλ του, καθώς και το stripe key στο οποίο θα αποθηκεύεται το κλειδί με το οποίο θα πραγματοποιεί πωλήσεις, αν και εφόσον αυτός ο χρήστης είναι πωλητής.

Για να διαβαθμίσουμε τους χρήστες βάσει ρόλων, φτιάξαμε τον πίνακα Roles. Αυτός έχει 2 πεδία, πέρα από τα στάνταρντ που δίνει το Laravel: name και description. Στο name έχουμε τον τίτλο του ρόλου και στο description έχουμε την περιγραφή του. Για παράδειγμα, name: Admin και description: Administrator user which has access to all contents of site and manages the administration page. Κάναμε μια σχέση πολλά προς πολλά για την ένωση αυτών των 2 πινάκων με τη χρήση ενός ενδιάμεσου πίνακα (pivot table) με όνομα User_Role. Αυτό το κάναμε για να έχουμε καλύτερο έλεγχο στις τιμές των πινάκων και να μπορούμε ανά πάσα στιγμή να έχουμε πρόσβαση από τον ένα πίνακα, στον άλλον με ευκολία. Επίσης το κάναμε για το ενδεχόμενο που κάποιος χρήστης έχει παραπάνω από έναν ρόλο. Οπότε για παράδειγμα στην εφαρμογή μπορεί να έχουμε πάνω από έναν πωλητή, ενώ ένας πωλητής να έχει επίσης ρόλο απλού χρήστη. Εδώ εξηγείται η χρήση της σχέσης πολλά προς πολλά.

Εδώ θα πρέπει να αναφέρουμε πως το Laravel μας δίνει την δυνατότητα να χρησιμοποιήσουμε seeders. Οι seeders είναι μέθοδοι του Laravel, που όπως λέει και το όνομά του, μας δίνει τη δυνατότητα να γεμίσουμε τους πίνακες της βάσης μας με έτοιμες τιμές/προϊόντα. Κάθε φορά που θα κάνουμε κάποια αλλαγή σε έναν seeder, μπορούμε με μία απλή εντολή να γεμίσει ξανά τον συγκεκριμένο πίνακα. Φυσικά αν κάνουμε αλλαγές στον ίδιο τον πίνακα, δεν έχουμε παρά να τρέξουμε ξανά το συγκεκριμένο migration.

Οπότε για τις απαραίτητες δοκιμές, έχουμε φτιάξει σε seeder 3-4 χρήστες και 3 ρόλους. Οι ρόλοι είναι Admin, Vendor και User και οι χρήστες έχουν διάφορους ρόλους. Οπότε κάθε φορά που θέλαμε να δοκιμάσουμε κάτι ή κάτι πήγαινε στραβά, απλά τρέχαμε ξανά τα migration και τους seeders και η εφαρμογή ήταν πάλι έτοιμη!

Παρακάτω θα δούμε ένα παράδειγμα ενός χρήστη και ενός ρόλου:

```
$user = new User();
$user->username = 'admin';
$user->email = 'admin@gmail.com';
$user->password = bcrypt('admin');
$user->save();
$user->roles()->attach(Role::where('name', 'Admin')->first());
```

Καθ’ αυτόν τον τρόπο, φτιάχνουμε έναν χρήστη στον seeder. Να αναφέρουμε πως το bcrypt είναι μια συνάρτηση η οποία κάνει hashing στον κωδικό που θα δώσει ο χρήστης, ή εμείς στο seeder. Το hashing έχει μονόδρομη κωδικοποίηση και δεν μπορεί βρει κάποιος με reverse-engineering τον αρχικό κωδικό. Τέλος στην τελευταία γραμμή κώδικα, βλέπουμε το πώς ορίζουμε τον ρόλο στον χρήστη.

Παρακάτω θα δούμε ένα παράδειγμα ρόλου:

```
Role::create([
    'id'    => 1,
    'name'    => 'Admin',
    'description'    => 'Admin User.'
]);
```

Έπειτα έχουμε τον Orders πίνακα. Εδώ αποθηκεύονται οι απαραίτητες πληροφορίες, αφού έχει γίνει μία αγορά. Περιέχει το όνομα, τη διεύθυνση καθώς και το user_id του χρήστη ο οποίος πραγματοποίησε την αγορά. Για λόγους απλότητας, δεν φτιάξαμε ξεχωριστό πίνακα που να κρατάει το καλάθι αγορών.

Αντί γι' αυτό, το βάλαμε να υπάρχει στο session. Έτσι όταν το καλάθι έμπαινε στο session και συνεχιζόταν η αγορά, το καλάθι γινόταν JSON και περνούσε όλο σε μορφή string στη στήλη με όνομα cart στον πίνακα orders. Τέλος ο πίνακας Orders έχει και ένα payment_id το οποίο εξυπηρετεί το stripe να ελέγξει την εγκυρότητα του κάθε χρήστη σε σχέση με την πληρωμή του.

Τώρα θα εξετάσουμε τον πίνακα Products. Ο πίνακας αυτός περιέχει τίτλο προϊόντος, μικρή και μεγάλη περιγραφή, κατηγορία και υποκατηγορία, φωτογραφία και thumbnail φωτογραφίας, τιμή, διαθεσιμότητα προϊόντος (stock). Τέλος έχει ως ξένο κλειδί ένα user_id το οποίο δείχνει στο id του user, ούτως ώστε να γίνει η ανάλογη συσχέτιση, σε περίπτωση που ο user είναι πωλητής.

Τώρα, όσον αφορά τις κατηγορίες και τις υποκατηγορίες, μία κίνηση θα ήταν να κάναμε δύο ενδιάμεσους πίνακες και να κάναμε συσχετίσεις ανάμεσα σε πίνακα product και πίνακα category και ανάμεσα σε category και sub-category. Έτσι ναι μεν, θα είχαμε περισσότερες επιλογές στην διαχείριση των δεδομένων, από την άλλη δε, θα αυξανόταν χωρίς λόγο η πολυπλοκότητα της βάσης.

Οπότε κινηθήκαμε με τον εξής τρόπο: Δημιουργήσαμε άλλον έναν πίνακα με όνομα Subcategories. Αυτός ο πίνακας είναι εξαιρετικά απλός, διότι περιέχει μόνο τα εξής πεδία: ένα πεδίο για μία εικόνα, που θα εξηγήσουμε σε λίγο τον λόγο ύπαρξής της, ένα πεδίο για την κατηγορία και άλλο ένα για την υποκατηγορία.

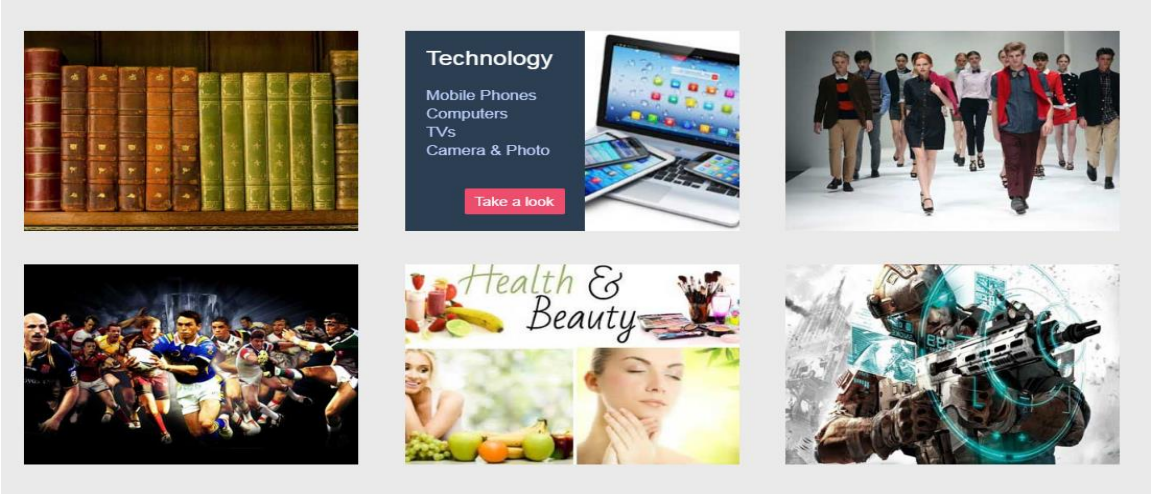
Όπως φτιάξαμε seeders για κάποιους χρήστες και για τους ρόλους, έτσι πράξαμε και για τις κατηγορίες και τις υποκατηγορίες. Έτσι λοιπόν, φτιάξαμε σε μορφή πίνακα, τη δομή κάθε κατηγορίας και περάσαμε σε μορφή JSON όλες τις υποκατηγορίες της εκάστοτε κατηγορίας.

Παρακάτω θα δούμε ένα παράδειγμα:

```
SubCategories::create(array(
        'id' => '1',
        'category' => 'Books',
        'image' => 'books.jpg',
        'subcategory' =>'{
          "0": {"name": "Drama","img": "pictures/Subcategories/Books/drama.jpg"},
          "1": {"name": "Horror","img": "pictures/Subcategories/Books/horror.jpg"},
          "2": {"name": "Romance","img": "pictures/Subcategories/Books/romance.jpg"},
           "3":{"name":"Sciencefiction","img":"pictures/Subcategories/Books/sciencefictio
n.jpg",
          "4": {"name": "Fantasy","img": "pictures/Subcategories/Books/fantasy.jpg"},
          "5":{"name":"Biography","img": "pictures/Subcategories/Books/biography.jpg"},
          "6":{"name":"Adventure","img": "pictures/Subcategories/Books/adventure.jpg"},
"7":{"name":"Educational","img":"pictures/Subcategories/Books/educational.jpg"}
        }'
      ));
```

Συνολικά έχουμε 6 κατηγορίες, οπότε κάτι αντίστοιχο επαναλήφθηκε 5 φορές ακόμα. Στο column category, βάλαμε το όνομα της κατηγορίας. Το image που προαναφέραμε, είναι το εξώφυλλο της κατηγορίας. Έπειτα έχουμε το column subcategory, το οποίο το περάσαμε υπό μορφή JSON. Αυτό έγινε γιατί ήταν πιο γρήγορη και εύκολη λύση από το να φτιάξουμε 2 ενδιάμεσους πίνακες. Επίσης τα queries που θα έπρεπε να κάνουμε για να φέρνει τα επιθυμητά αποτελέσματα με τον άλλον τρόπο, θα μείωνε την απόδοση του site. Οπότε μέσα στην υποκατηγορία έχουμε 2 πεδία: το name, το οποίο είναι το όνομα της υποκατηγορίας, και το img το οποίο είναι path με τη φωτογραφία της κάθε υποκατηγορίας.

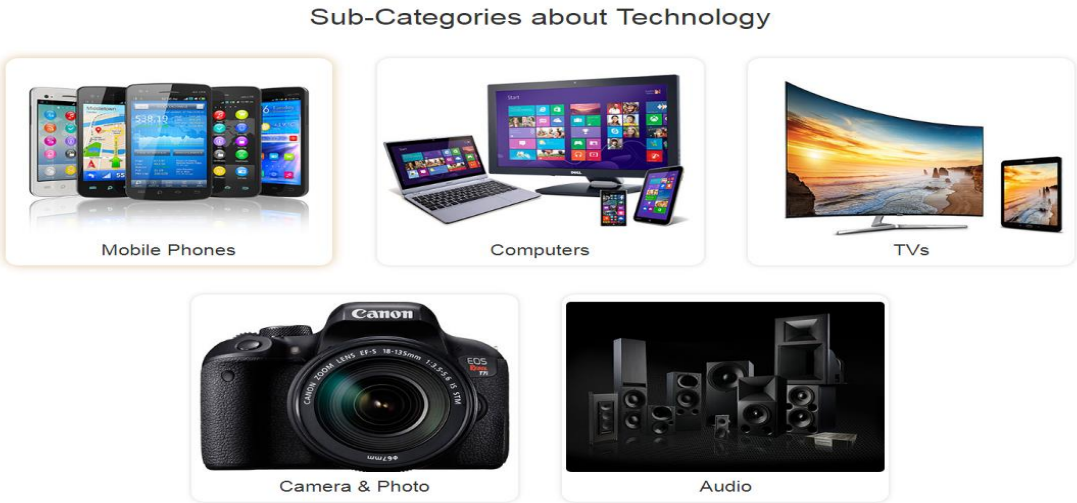
Παρακάτω θα δούμε μια φωτογραφία από τις κατηγορίες της πλατφόρμας:



Εικόνα 14: Κατηγορίες πλατφόρμας

Όπως προαναφέραμε οι κατηγορίες είναι 6, εξού και οι 6 φωτογραφίες. Στο στιγμιότυπο, έχουμε κάνει hover με το ποντίκι, πάνω στη 2η κατηγορία. Έτσι ένα μικρό κουτάκι παρουσιάζεται και μας λέει το όνομα της κατηγορίας μαζί με τις 4 από τις υποκατηγορίες που έχει. Αν πατήσουμε σε κάποια υποκατηγορία, θα μας εμφανίσει σε νέα σελίδα, τα προϊόντα που έχει αυτή η υποκατηγορία. Διαφορετικά, μπορούμε να πατήσουμε στο Take a look και να μας πάει σε άλλη σελίδα, οποία θα μας αποκαλύψει και άλλες, αν υπάρχουν, κατηγορίες.

Ας δούμε μέσα στην τεχνολογία τι υπάρχει με την παρακάτω φωτογραφία:



Εικόνα 15: Υποκατηγορίες πλατφόρμας

Όπως και πριν, έτσι και εδώ όταν κάνουμε hover με το ποντίκι σε μία υποκατηγορία, εμφανίζεται ένα κίτρινο σχεδιάγραμμα. Αυτό γίνεται για την καλύτερη εμπειρία του χρήστη και για να καταλαβαίνει ποια κατηγορία, είναι έτοιμος να επιλέξει.

3.5 ΜΗΧΑΝΗ ΑΝΑΖΗΤΗΣΗΣ

3.5.1 Η απλή προσέγγιση

Όπως κάθε σωστό ηλεκτρονικό κατάστημα, έτσι και η δική μας πλατφόρμα, χρειαζόταν μια μηχανή αναζήτησης, ούτως ώστε οι χρήστες να ψάχνουν τα προϊόντα της αρεσκείας τους, εύκολα και γρήγορα.

Για επιτευχθεί αυτό, στον controller των προϊόντων, φτιάξαμε μια νέα συνάρτηση, η οποία έκανε ακριβώς αυτό: ένα απλό database query το οποίο κατεύθυνε τον χρήστη στην κατάλληλη σελίδα και ανάλογα τα αποτελέσματα, του έδινε και το αντίστοιχο μήνυμα ή/και τα λήμματα.

Παρακάτω θα δούμε το ένα λόγω query που μας επιστρέφει τα αποτελέσματα:

```
$products = Product::where('title', 'LIKE', "%{$query}%")->orWhere('description', 'LIKE', "%{$query}%")->get();
```

Το συγκεκριμένο query κοιτάει τον πίνακα Product και όπου υπάρχει τίτλος ή περιγραφή, ίδια ή παρόμοια με αυτό που πληκτρολόγησε ο χρήστης, φέρνει τα αντίστοιχα αποτελέσματα.

Το συγκεκριμένο query έγινε με ρήτρα “LIKE”. Το LIKE στην SQL επιστρέφει ίδια ή παρόμοια αποτελέσματα, χρησιμοποιώντας wildcard operators. Τα wildcard operators στην προκειμένη περίπτωση είναι τα % τα οποία χρησιμοποιούνται μπροστά και πίσω από την μεταβλητή η οποία θα επιστρέψει κάποιο αποτέλεσμα. Πχ "%{$query}%.

Για να γίνει πιο σωστό το αποτέλεσμα και για την καλύτερη εμπειρία του χρήστη, το query πριν δώσει το αποτέλεσμα, περνάει πρώτα από φιλτράρισμα. Οπότε για να αποφύγουμε αφελής πληκτρολογήσεις από χρήστες στη φόρμα αναζήτησης, όπως σύμβολα, σημεία στίξης, κενά κλπ, χρησιμοποιούμε regular expressions. Παρακάτω θα δούμε τον υπεύθυνο κώδικα:

```
$query = preg_replace('/[\.\,\!\(\)]/', '', ltrim($request->input('query')));
```

Η συνάρτηση preg_replace εφαρμόζει το regular expression που έχουμε δώσει. Στην προκειμένη περίπτωση το regular expression που χρησιμοποιήσαμε είναι το /[\.\,\!\(\)]/. Εδώ ταιριάζει αποτελέσματα, όπως τελεία, κόμμα, θαυμαστικό, άνοιγμα παρένθεσης και κλείσιμο παρένθεσης. Έπειτα με την ltrim, αν τυχόν βρήκε κάποια από τα παραπάνω στη φόρμα αναζήτησης, τα κόβει, ούτως ώστε το query να είναι καθαρό.

3.5.2 **Μηχανή αναζήτησης με φίλτρα**

Αφού φτιάξαμε μια βασική δομή της μηχανής αναζήτησης, αποφασίσαμε να κάνουμε μία, με πιο περίπλοκη λογική, έτσι ώστε ο χρήστης να έχει την δυνατότητα να επιλέξει διάφορα φίλτρα, όπως όριο τιμής από το μικρότερο ως το μεγαλύτερο ή ακόμα και μια κατηγορία ή/και την εκάστοτε υποκατηγορία της.

Για αρχή κάναμε έναν απλό έλεγχο, αν όλα τα inputs που δίνει ο χρήστης, περιέχουν κάτι. Αν δεν περιέχουν, στην ουσία τον κάνουμε ανακατεύθυνση στην αρχική σελίδα. Αν έστω και ένα περιέχει κάτι, δημιουργούμε έναν πίνακα το οποίο είναι αποθηκευμένο στην μεταβλητή $conditions. Σε αυτόν τον πίνακα ελέγχουμε ένα-ένα τα inputs, αν έχουν κάποια τιμή. Αν έχουν, τότε αποθηκεύουμε στη μεταβλητή $conditions σε μορφή πίνακα, το query.

Παρακάτω θα δούμε ένα παράδειγμα:

```
if ($title != "") {
    $conditions[] = array('title', 'LIKE', "%{$title}%");
}
```

Παρόμοιοι έλεγχοι γίνονται και για τα υπόλοιπα inputs. Εν τέλει, κάνουμε ένα μαζικό query, χρησιμοποιώντας την μεταβλητή $conditions:

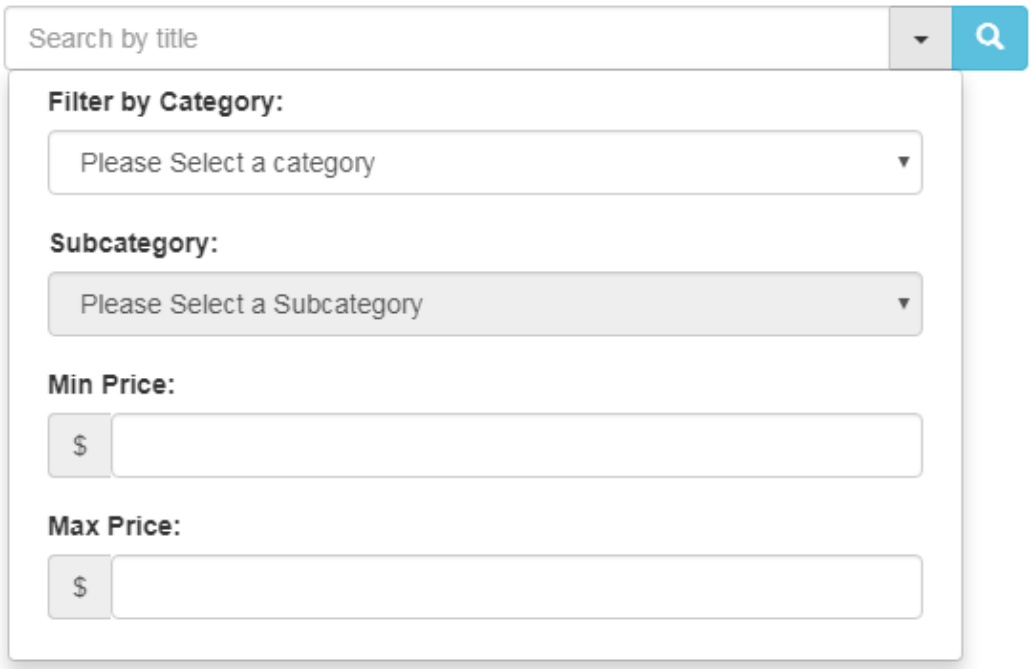
```
$products = Product::where($conditions)->get();
```

Τέλος, όπως κάναμε αρχικά τα φιλτραρίσματα του query, στην 1η έκδοση της μηχανής αναζήτησης, έτσι προβήκαμε και για τα υπόλοιπα inputs. Για τα inputs, τα οποία περιείχαν αριθμούς, πχ για τις τιμές, χρησιμοποιήσαμε διαφορετικό regular expression, όπως θα δούμε στο παρακάτω παράδειγμα:

```
$priceFrom = preg_replace('/[^.0-9]/', '', ltrim($request->input('priceFrom')));
```

Το συγκεκριμένο regular expression ταιριάζει έναν χαρακτήρα ο οποίος θα είναι μόνο αριθμός. Οπότε δεν μπορεί να δεχτεί γράμματα ή σύμβολα.

Ας δούμε πως φαίνεται πλέον η μηχανή αναζήτησης



Εικόνα 16: Μηχανή αναζήτησης

3.6 ΠΡΟΣΤΑΣΙΑ ΣΕΛΙΔΩΝ ΑΠΟ ΤΟΥΣ ΧΡΗΣΤΕΣ

Καθώς γράφαμε τις θεμελιώδης συναρτήσεις που χρειαζόταν ένα ηλεκτρονικό κατάστημα, όπως το να βάζεις προϊόντα μέσα στο καλάθι, να τα αφαιρείς κ.λπ., καταλάβαμε πως θα έπρεπε να καθιερώσουμε μία σχετική ασφάλεια, ανάμεσα στις σελίδες. Πιο συγκεκριμένα, θέλουμε να υπάρχει μία ασφάλεια στην σελίδα διαχείρισης του e-shop.

Για τον λόγο αυτό φτιάξαμε ένα νέο middleware. Το middleware, όπως λέει και το όνομά του, λειτουργεί ως ενδιάμεσος ανάμεσα σε ένα request και σε ένα response. Με λίγα λόγια, κάθε είδους ασφάλεια που παρέχει το Laravel και γενικότερα τα frameworks, βρίσκονται στα διάφορα middlewares. Ένα βασικό middleware που έχει το Laravel είναι η ασφάλεια που δίνει στις σελίδες, όταν ο χρήστης έχει εισέλθει στον λογαριασμό του. Δηλαδή μπορούμε να ορίσουμε ένα συγκεκριμένο view να επιτρέπεται μόνο σε χρήστες που έχουν κάνει log in, ενώ τους επισκέπτες να μην τους αφήνει. Αυτό μας βόλεψε, μέχρι που διαβαθμίσαμε τους χρήστες με ρόλους: user, vendor, administrator. Όπως φαίνεται, και οι 3 ρόλοι, συγκαταλέγονται στους χρήστες, οπότε κάθε ένας, που θα έβαζε τα στοιχεία του και θα έμπαινε στον λογαριασμό του, με λίγο ψάξιμο, θα έβρισκε και την σελίδα διαχείρισης και θα μπορούσε να πειράξει διάφορα πράγματα, κάτι που δεν θέλαμε.

Πριν όμως δούμε το middleware, θα πρέπει πρώτα να κάνουμε κάποιους ελέγχους, στο μοντέλο του χρήστη. Αυτοί οι έλεγχοι, θα κρίνουν για το αν ο χρήστης έχει κάποιο ρόλο και θα επιστρέφει τα ανάλογα αποτελέσματα. Παρακάτω θα δούμε τον υπεύθυνο κώδικα:

```
public function hasAnyRole($roles){
    if (is_array($roles)){
        foreach ($roles as $role) {
            if ($this->hasRole($role)) {
                return true;
            }
        }
```

```
        } else if ($this->hasRole($roles)) {
            return true;
        }
        return false;
    }

    public function hasRole($role){
        if ($this->roles()->where('name', $role)->first()){
            return true;
        }
        return false;
    }
```

Η συνάρτηση hasAnyRoles ελέγχει αν ο authenticated χρήστης έχει κάποιο ρόλο, ούτως ώστε να του δώσει την κατάλληλη πρόσβαση.

Στο 1ο if, κάνει foreach σε όλους τους ρόλους και ελέγχει αν ο χρήστης έχει κάποιους από αυτούς με χρήση μιας άλλης συνάρτησης, της hasRole. Αν έχει, τότε επιστρέφει true. Το foreach στην προκειμένη περίπτωση χρησιμεύει αν δίναμε σε κάποιο χρήστη 2 ρόλους ταυτόχρονα. Πχ ο διαχειριστής της πλατφόρμας, θα μπορούσε κάλλιστα να ήταν και ένας απλός χρήστης και να μπορεί να αγοράζει προϊόντα.

Αλλιώς ελέγχει αν ο χρήστης έχει κάποιο ρόλο, πάλι με χρήση της συνάρτησης hasRole. Αν έχει, τότε επιστρέφει true. Ειδάλλως, επιστρέφει false.

Τώρα όσον αφορά την συνάρτηση hasRole, είναι αυτή που κάνει την πιο σημαντική δουλειά. Αυτή ελέγχει αν ο χρήστης έχει έναν συγκεκριμένο ρόλο. Πιο συγκεκριμένα, μπαίνει μέσα στους ρόλους από τον χρήστη, βάσει τις συσχετίσεις που έχουμε κάνει στην βάση δεδομένων, και βλέπει αν εμφανίζεται ο ρόλος που τσεκάρουμε, στους ρόλους που έχει ο χρήστης. Αν εμφανίζεται, επιστρέφει true, ειδάλλως επιστρέφει false.

Εφόσον κάναμε τους απαραίτητους ελέγχους για τους ρόλους, στο μοντέλο του χρήστη, επανερχόμαστε στο νέο middleware. Αυτό το middleware το ονομάσαμε CheckRole και έχει έναν απλό σκοπό: να κοιτάει τα requests που κάνουν οι χρήστες και ανάλογα τον ρόλο τους, να κάνει τις αντίστοιχες ενέργειες.

Παρακάτω θα δούμε το τμήμα κώδικα που κάνει ακριβώς αυτό:

```
if ($request->user()=== null){
        return redirect()->route('product.index');
 }
$actions = $request->route()->getAction();
$roles = isset($actions['roles']) ? $actions['roles'] : null;

if ($request->user()->hasAnyRole($roles) || !$roles) {
        return $next($request);
}
        return redirect()->route('product.index');
```

Στο 1ο If, ελέγχει αν έχουμε κάποιον χρήστη στο request, δηλαδή αν είναι επισκέπτης. Σε περίπτωση που είναι, τον κάνει ανακατεύθυνση στην αρχική σελίδα, με το κατάλληλο μήνυμα.

Σε αυτό το σημείο, πρέπει να ανακτήσουμε κάποιες ενέργειες. Αυτό το κάνουμε έχοντας πρόσβαση στο route και έπειτα χρησιμοποιώντας την μέθοδο getAction(). Τώρα αυτή η μέθοδος μπαίνει στα Route και παίρνει τις τιμές κλειδιά.

Για παράδειγμα, ας δούμε πως ορίζεται ένα route:

```
Route::post('/checkout', [
    'uses' => 'OrderController@postCheckout',
    'as' => 'checkout',
    'middleware' => 'auth'
]);
```

Το συγκεκριμένο Route χρησιμοποιεί POST HTTP μέθοδο, χρησιμοποιεί τον OrderController, την συνάρτηση postCheckout και έχει ως middleware το auth. Οι τιμές κλειδιά που μας ενδιαφέρουν είναι μέσα στις αγκύλες. Πρέπει να ανακτήσουμε αυτές τις ενέργειες για να δούμε τι ρόλο έχουμε δώσει σε κάθε σελίδα.

Το ίδιο Route βέβαια, με το καινούριο middleware θα τροποποιηθεί ως εξής:

```
Route::post('/checkout', [
    'uses' => 'OrderController@postCheckout',
    'as' => 'checkout',
    'middleware' => 'roles',
    'roles' => ['User']
]);
```

Εδώ πλέον δεν χρησιμοποιούμε το auth middleware, αλλά αυτό που φτιάξαμε εμείς. Από κάτω του ορίζουμε και τον ρόλο, Στην προκειμένη περίπτωση, πρόκειται για ρόλο User. Οπότε με την getAction() μέθοδο, ελέγχουμε αυτά τα πράγματα.

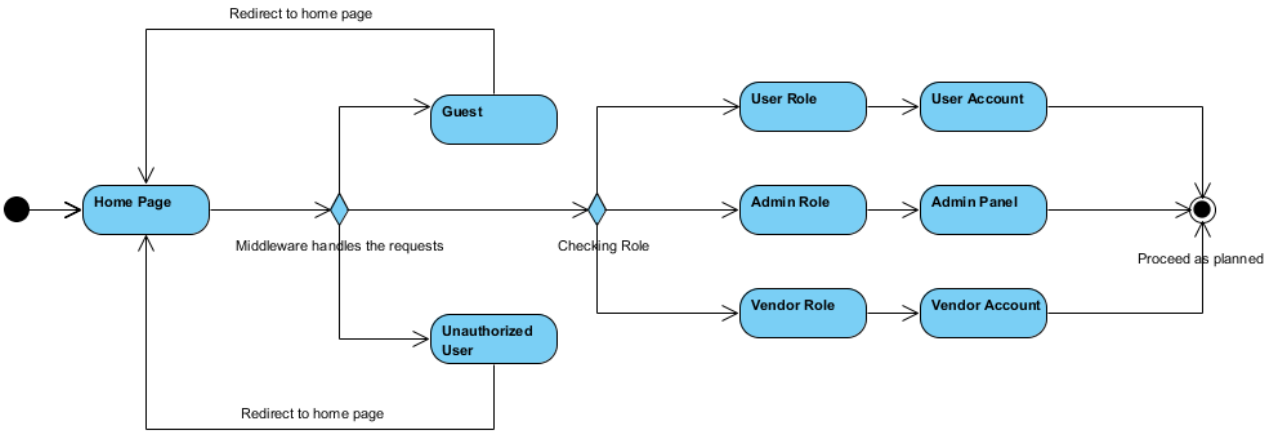
Έπειτα ελέγχουμε στην μεταβλητή roles αν τα actions έχουν όντως το roles κλειδί που αναφέραμε πριν, διότι θα υπάρχουν και routes που δεν θα έχουν αυτήν την έξτρα προστασία. Αν έχουν το roles κλειδί, τότε το αποθηκεύουμε. Ειδάλλως το θέτουμε ως null.

Έπειτα στο 2ο If κοιτάει αν ο User έχει κάποιο ρόλο και ποιος είναι αυτός, καλώντας την συνάρτηση hasAnyRoles, που είδαμε λεπτομερώς πιο πάνω. Αν ο χρήστης, έχει κάποιον ρόλο ή δεν υπάρχει κάποιος ρόλος ως προαπαιτούμενο για κάποια σελίδα, τότε τον προχωράει στο επόμενο request και σταματάει τους περαιτέρω ελέγχους.

Ειδάλλως, για τους μη εξουσιοδοτημένους χρήστες, τους ανακατευθύνει στην αρχική σελίδα.

Πχ αν ένα view χρειάζεται ρόλο διαχειριστή και ο χρήστης που θέλει να εισέλθει, έχει τον ρόλο διαχειριστή, τότε τον περνάει στο επόμενο request και κατά συνέπεια να δει την σελίδα διαχείρισης. Αλλιώς τον στέλνει στην αρχική σελίδα.

Παρακάτω μπορούμε να δούμε το σχεδιάγραμμα ροής του middleware:



Διάγραμμα 6: Activity diagram custom Middleware process

3.7 ΑΝΤΑΠΟΚΡΙΣΗ ΣΕ ΠΟΛΛΑΠΛΕΣ ΟΘΟΝΕΣ

Κάθε διαδικτυακή εφαρμογή πρέπει να μεριμνήσει στο να ανταποκρίνεται σωστά σε πολλαπλές οθόνες. Πιο συγκεκριμένα, θα πρέπει να πιάσει ένα ευρύτερο φάσμα χρηστών, όπου άλλοι χρησιμοποιούν κινητά για να περιηγηθούν στην πλατφόρμα, άλλοι tablet και άλλοι με τον υπολογιστή τους, έχοντας αρκετά μεγαλύτερες σε διάσταση, οθόνες.

Για να το επιτύχουμε αυτό, χρησιμοποιήσαμε το CSS Framework της Twitter, το Bootstrap. Το Bootstrap λοιπόν κάνει ακριβώς αυτό το πράγμα: μας λύνει τα χέρια από τα αναρίθμητα media queries που θα χρειαζόμασταν για να επιτύχουμε σωστή ανταπόκριση στις διάφορες οθόνες της αγοράς. Φυσικά το Bootstrap από μόνο του δεν λύνει το πρόβλημα, καθώς κάποια ιδιαίτερα κομμάτια κώδικα ήθελαν media queries, αλλά μας μείωσε το χρόνο στον οποίο αφιερώσαμε για να κάνουμε ανταποκρίσιμη την πλατφόρμα. Τα media queries είναι μία τεχνική στις εντολές του CSS, στα οποία μας επιτρέπει να γράψουμε για τα ίδια elements, διαφορετικούς κανόνες και για συγκεκριμένη ανάλυση, δηλαδή ύψος και πλάτος οθόνης.

Παρακάτω θα δούμε ένα παράδειγμα.

```
@media screen and (min-width: 768px) {
   #adv-search {
      width: 500px;
      margin: 0 auto;
   }
}
```

Με @media ξεκινάνε τα media queries και έχουν brackets όπως οι συναρτήσεις. Επίσης σε αυτά μπορούμε να θέσουμε σε τι διαστάσεις θα ισχύουν οι κανόνες μας. Για παράδειγμα, όπως βλέπουμε, έχει τεθεί min-width: 768px. Αυτό σημαίνει πως για πλάτος 768pixel και πάνω θα ισχύει ο κανόνας που έχει στο παράδειγμα.

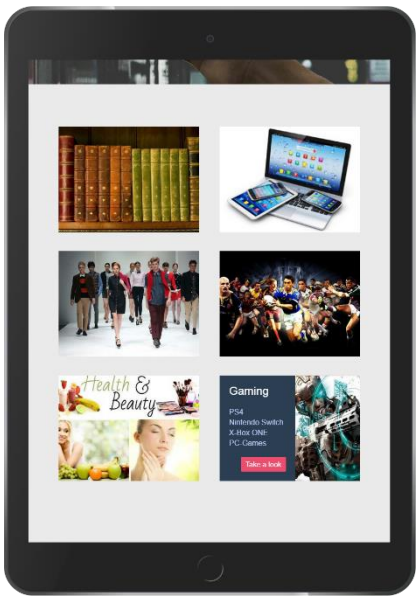
Από την άλλη, το Bootstrap, όταν χρησιμοποιούμε τις κλάσεις του, μας προσφέρει ένα αντίστοιχο αποτέλεσμα, χωρίς να χρειαστούμε να γράφουμε για την κάθε διάσταση ξεχωριστά.

Ας δούμε μερικά παραδείγματα το πώς ανταποκρίνεται η πλατφόρμα σε διάφορες διαστάσεις:

Το συγκεκριμένο μοντέλο είναι ένα iPhone 6 το οποίο έχει διαστάσεις 375x667.

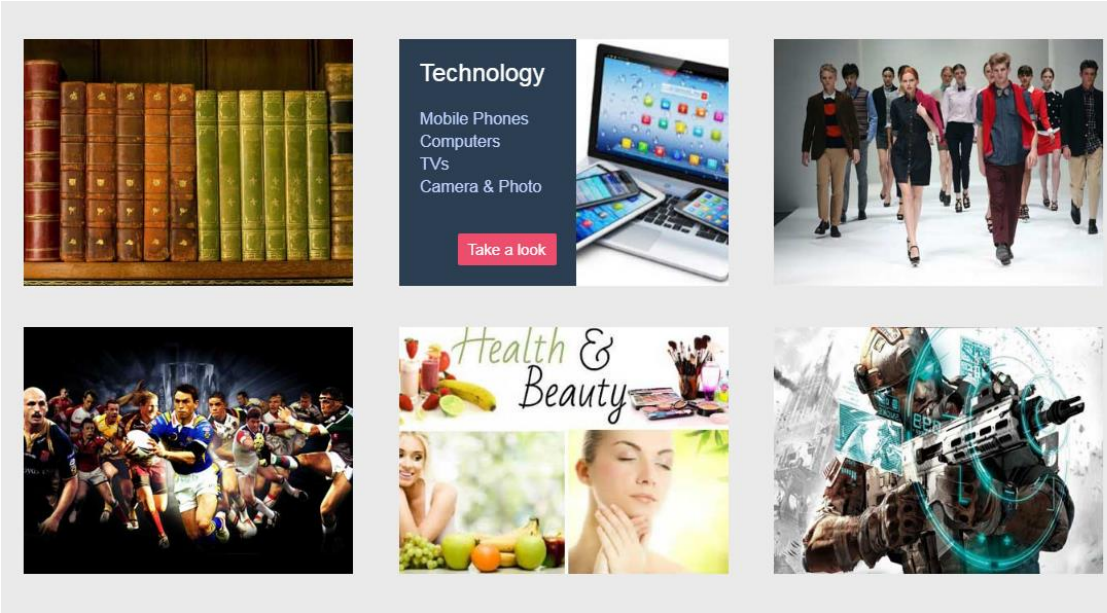
Το ύψος δεν έχει μεγάλη σημασία στην προκειμένη περίπτωση. Το πλάτος είναι αυτό που πάντα πρέπει να προσέχουμε.

Εικόνα 17: Πλατφόρμα σε διαστάσεις 375 x 667



Εδώ θα εξετάσουμε την ίδια περίπτωση αλλά σε μεγαλύτερη οθόνη. Η συσκευή αυτή είναι ένα iPad, η οποία έχει διαστάσεις 768x1024

Εικόνα 18: Πλατφόρμα σε διαστάσεις 768x1024



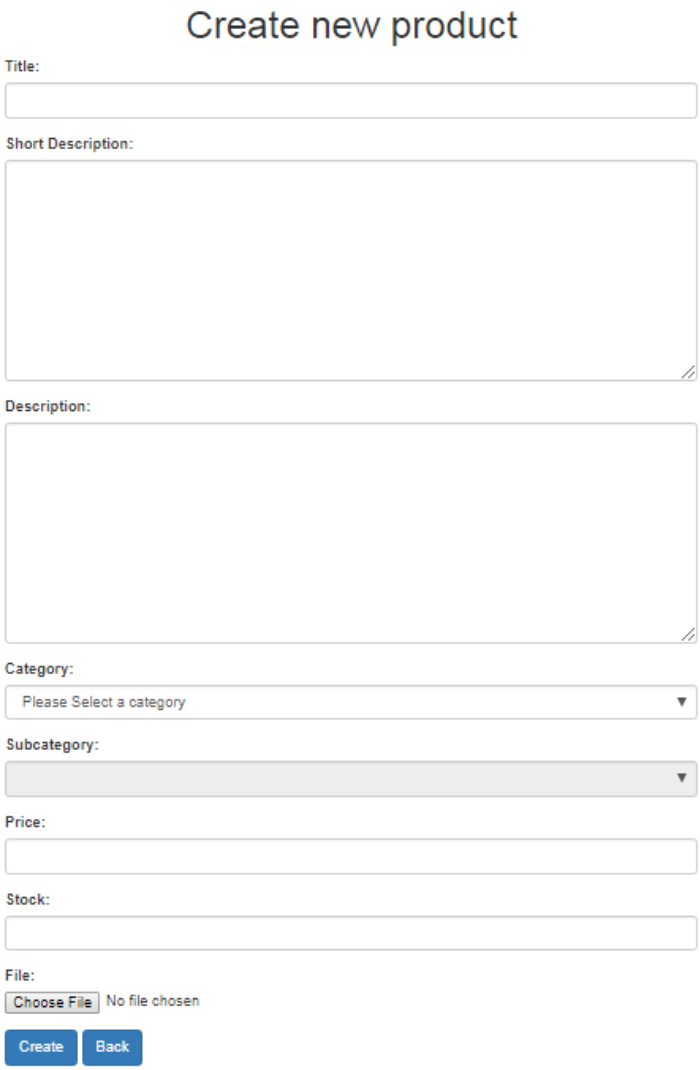
Εικόνα 19: Πλατφόρμα σε διαστάσεις 1440 x 900

Τέλος θα εξετάσουμε την οθόνη στην οποία φτιάχτηκε εξ' αρχής η πλατφόρμα. Πρόκειται για μία οθόνη υπολογιστή με ανάλυση 1440 x 900.

Όπως παρατηρούμε, υπάρχουν μικροδιαφορές από ανάλυση σε ανάλυση, αλλά εν τέλει, σε όλες τις διαστάσεις, η πλατφόρμα ανταποκρίνεται εξαιρετικά!

3.8 ΑΠΟΘΗΚΕΥΣΗ ΦΩΤΟΓΡΑΦΙΩΝ ΣΤΟ LARAVEL

Μια κύρια δυνατότητα που έχουν οι πωλητές, αφού φτιάξουν τον λογαριασμό τους, είναι να δημιουργούν και να επεξεργάζονται τα προϊόντα τους. Στα προϊόντα αυτά, έχουν πολλές επιλογές και πεδία να συμπληρώσουν όπως τίτλος, κατηγορία, υποκατηγορία κλπ.



Εικόνα 20: Φόρμα δημιουργίας προϊόντος

Έχει επίσης ένα πεδίο για να ανεβάσει μία φωτογραφία του προϊόντος. Το Laravel μπορεί να διαχειριστεί από μόνο του τις φωτογραφίες, αλλά δεν μπορεί να τις επεξεργαστεί. Έτσι περάσαμε στην εφαρμογή μας ένα πακέτο που κάνει ακριβώς αυτό: Το εν λόγω πακέτο λέγεται intervention και μπορεί να επεξεργαστεί με ευκολία τις

εικόνες και να προσθέσει διάφορα εφέ όπως trim, blur, opacity, greyscale κλπ. Εμείς κυρίως το χρησιμοποιήσαμε για σμίκρυνση των διαστάσεων της εικόνας για να φτιάξουμε ένα thumbnail, για την σωστή απεικόνισή της στη πλατφόρμα.

Αυτό το κάναμε, ούτως ώστε ο πωλητής να μην υποχρεώνεται να κάνει resize μόνος του την φωτογραφία, αλλά να την ανεβάζει σε οποιαδήποτε ανάλυση και η πλατφόρμα θα την διαχειριστεί καταλλήλως. Αφ' ετέρου, το κάναμε για να μην «σπάσει» την πλατφόρμα, όπως για παράδειγμα, μια τεράστια εικόνα θα μας χαλούσε δραματικά το markup και θα έπρεπε να κάνουμε πάλι αλλαγές στο CSS. [16]

Ας δούμε τον κώδικα ο οποίος είναι υπεύθυνος για τις εικόνες.

```
if($file = $request->hasFile('image')) {
        $file = $request->file('image');
        $extension = $file->getClientOriginalName();
        $username = Auth::user()->username;
        $thumb = Image::make($file->getRealPath())->resize(100, 100, function ($constraint) {
            $constraint->aspectRatio(); //maintain image ratio
        });
        $destinationPath = public_path('/uploads/products/' . $username);
        $file->move($destinationPath, $extension);
        $thumb->save($destinationPath.'/thumb_'.$extension);
        $product['imagePath'] = '/uploads/products/'. $username . '/' . $extension;
        $product['thumbnail'] = '/uploads/products/'. $username . '/thumb_' . $extension;
    }
```

Το κομμάτι κώδικα που μας νοιάζει είναι το συγκεκριμένο:

```
$thumb = Image::make($file->getRealPath())->resize(100, 100, function ($constraint) {
        $constraint->aspectRatio(); //maintain image ratio
});
```

Αυτό είναι υπεύθυνο στο να κάνει σμίκρυνση στην φωτογραφία αλλά παράλληλα να διατηρήσει σωστές αναλογίες.

Παρακάτω βλέπουμε το path στο οποίο αποθηκεύονται οι φωτογραφίες. Για να υπάρχει μια τάξη στα αρχεία, έχουμε ορίσει να δημιουργείται μέσα στο /uploads/products/ ένας φάκελος με το όνομα του χρήστη τον οποίο έφτιαξε το προϊόν. Έτσι όλες οι φωτογραφίες που θα ανεβάζει ο x πωλητής, θα είναι μέσα στον δικό του φάκελο.

```
$destinationPath = public_path('/uploads/products/' . $username);
```

Συνεχίζοντας, βλέπουμε το τελικό όνομα που παίρνει το κάθε αρχείο, μέσα στο φάκελο με το μεταβλητό όνομα:

```
$product['imagePath'] = '/uploads/products/'. $username . '/' . $extension;
```

Και αυτό για το thumbnail:

```
$product['thumbnail'] = '/uploads/products/'. $username . '/thumb_' . $extension;
```

Οπότε στον φάκελο πάντα θα υπάρχουν 2 αρχεία με ίδιο περιεχόμενο: το 1^{ο} θα είναι η αυθεντική φωτογραφία και το 2^{ο} θα είναι το thumbnail.

4 ΒΑΣΙΚΑ ΠΡΟΒΛΗΜΑΤΑ ΚΑΙ ΠΩΣ ΑΝΤΙΜΕΤΩΠΙΣΤΗΚΑΝ

Το βασικό πρόβλημα που αντιμετωπίσαμε ήταν το πώς θα στήσουμε την εφαρμογή να δέχεται ταυτόχρονα, πολλαπλά ηλεκτρονικά καταστήματα. Εφόσον επιλέξαμε το Stripe ως Payment Gateway, έπρεπε να βρούμε ένα τρόπο με αυτό. Ο πιο σωστός τρόπος να επιλυθεί αυτό το ζήτημα θα ήταν να χρησιμοποιήσουμε το Stripe Connect.

Το Stripe Connect είναι η λύση για κάθε επιχείρηση η οποία χρειάζεται να επεξεργαστεί πληρωμές και να πληρώσει έμμεσα, τα ηλεκτρονικά καταστήματα που φιλοξενεί. Όμως λόγω της ελλιπούς τεκμηρίωσης και πως το Stripe Connect δεν υποστηρίζεται προς το παρόν στην Ελλάδα, έπρεπε να βρούμε έναν άλλον τρόπο να διαχειριστούμε αυτό το πρόβλημα.

Την λύση δώσαμε, ακολουθώντας την απλή ροή του Stripe για ένα κατάστημα, με την μόνη διαφορά πως τα κλειδιά θα τα παίρνει δυναμικά από την βάση δεδομένων και στο καλάθι αγορών, ο πελάτης θα έχει την επιλογή ποιον καταστηματάρχη να πληρώσει πρώτα, σε περίπτωση που το καλάθι έχει προϊόντα από διάφορους πωλητές.

Ένα άλλο πρόβλημα που συναντήσαμε αργότερα ήταν στην δημιουργία των προϊόντων από την μεριά των πωλητών. Συγκεκριμένα, κάθε φορά που ένας πωλητής ανέβαζε ένα προϊόν, κάποιος άλλος πωλητής μπορούσε να δει το προϊόν του πρώτου στο δικό του προφίλ! Αυτό αποτελούσε πρόβλημα διότι εκτός του ότι δεν ήταν σωστό να βλέπει ο κάθε πωλητής όλη την λίστα των προϊόντων του καταστήματος, θα μπορούσε να τροποποιήσει αλλά ακόμα και να διαγράψει προϊόντα και ας μην ήταν τα δικά του! Έτσι λοιπόν, για να λύσουμε αυτό το πρόβλημα, κάναμε το εξής: στο πίνακα του Products, προσθέσαμε ένα ξένο κλειδί, με όνομα user_id, το οποίο θα «έδειχνε» στο μοναδικό κλειδί id στον πίνακα Users. Έπειτα, στην συνάρτηση η οποία καλεί τα προϊόντα, τροποποιήσαμε ως εξής την εντολή:

$products = Product::where(‘user_id’,auth()->user()->id)->paginate(5);

Το auth()->user()->id θα αντιστοιχεί στο id του χρήστη ο οποίος είναι συνδεδεμένος εκείνη τη στιγμή. Επίσης με χρήση του Eloquent ORM Model, χρησιμοποιήσαμε την εντολή where, η οποία αντιστοιχεί το ξένο κλειδί user_id του πίνακα Products, με το id του Users. Έτσι καταφέραμε κάθε πωλητής, να βλέπει και να επεξεργάζεται μόνο τα δικά του προϊόντα.

Επίσης ένα σημαντικό πρόβλημα που είχαμε, ήταν η ασφάλεια συγκεκριμένων σελίδων της εφαρμογής. Για παράδειγμα, θέλαμε να έχει πρόσβαση στην Administration σελίδα μόνο ένας λογαριασμός admin. Αν κάποιος είναι συνδεδεμένος ως User ή Vendor και έγραφε στο url για παράδειγμα /admin θα τον οδηγούσε στην Administration σελίδα δίχως πρόβλημα.

Πολλά άλλα Frameworks, έχουν αυτήν την ικανότητα από την αρχή, αλλά στο Laravel δεν υπήρχε μια τέτοια λύση.

Οπότε για αρχή δημιουργήσαμε ένα απλό σύστημα ρόλων. Όπως έχουμε εξηγήσει, υπάρχουν 3 διαφορετικοί ρόλοι, Users, Vendors, Admins. Με χρήση Eloquent ORM δημιουργήσαμε μία πολλά-προς-πολλά σχέση ανάμεσα σε Users και Roles και φτιάξαμε ένα Pivot πίνακα user_role, οποίος λειτουργεί ως ενδιάμεσος των 2 προηγούμενων και αντιστοιχεί κάθε χρήστη με τον ρόλο του.

Έπειτα, φτιάξαμε το δικό μας Middleware, μιας που αυτό που έχει το Laravel, απλά προστατεύει τις σελίδες από μη συνδεδεμένους χρήστες. Εμείς θέλαμε μια έξτρα προστασία, η οποία θα ελέγχει τι ρόλο έχει ο συνδεδεμένος χρήστης. Αν έχει ρόλο admin, τότε θα τον οδηγήσει στη σελίδα διαχείρισης και ούτω καθεξής. Έτσι αν κάποιος είναι συνδεδεμένος ως User ή Vendor και έγραφε στο URL για παράδειγμα /admin θα τον κάνει redirect στην αρχική σελίδα.

ΕΠΙΛΟΓΟΣ

Στην παρούσα πτυχιακή εργασία εργαστήκαμε πάνω σε μία πλατφόρμα η οποία έχει την δυνατότητα να περιλαμβάνει πολλαπλά ηλεκτρονικά καταστήματα.

Το 1ο κεφάλαιο αφιερώθηκε στην συνοπτική ανάλυση απαιτήσεων της εφαρμογής. Το 2ο κεφάλαιο περιείχε πληροφορίες σχετικά με τα εργαλεία που χρησιμοποιήσαμε, πως στήθηκε η εφαρμογή και ποια αρχιτεκτονική λογισμικού ακολουθήσαμε. Το 3ο κεφάλαιο, περικλείει αναλυτική περιγραφή τον τρόπο λειτουργίας της εφαρμογής, από την βάση δεδομένων μέχρι και την ροή μιας αγοραπωλησίας. Τέλος, το 4ο κεφάλαιο, έχει διάφορες δυσκολίες, όπου συναντήσαμε και πως επιλύθηκαν.

Η κατασκευή ενός ηλεκτρονικού καταστήματος είναι μία «υποχρεωτική» πλέον κίνηση που πρέπει να κάνει ένας μαγαζάτορας, για το μαγαζί του, έτσι ώστε να βελτιώσει τα κέρδη της επιχείρησής του. Με την κατασκευή αυτής της διαδικτυακής εφαρμογής ήρθαμε σε επαφή με πολλά ενδιαφέροντα γνωστικά αντικείμενα, όπως τις βασικές αρχές ενός ηλεκτρονικού καταστήματος, πως δουλεύει ένα Payment Gateway API, καθώς και την απαραίτητη ασφάλεια που πρέπει να υπάρχει, τόσο στην πλατφόρμα, όσο και στην αγορά προϊόντος.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. https://www.thebalance.com/amazon-com-company-research-2071316
2. https://www.shopify.com/faq
3. http://www.kepa.gov.cy/diktiothite/Portal/PortalDocuments.aspx?DocumentId=a735c138-a74e-483b-8720-31bc9aca7169
4. https://gr.norton.com/vital-security/article
5. https://opensource.com/business/16/6/which-php-framework-right-you
6. http://noeticforce.com/best-php-frameworks-for-modern-web-development
7. https://1stwebdesigner.com/web-frameworks/
8. http://laravelbook.com/laravel-architecture/
9. http://www.findalltogether.com/tutorial/laravel-framework-architecture/
10. http://www.lassosoft.com/Tutorial-Understanding-Cookies-and-Sessions
11. https://laravel.com/docs/5.4/blade
12. Laravel: My first Framework by Maksim Surguy
13. https://stripe.com/docs/
14. https://stripe.com/docs/stripe.js/v2
15. https://stripe.com/docs/testing#cards
16. http://image.intervention.io/