ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δημιουργία Διαδικτυακού Καταστήματος Χρησιμοποιώντας Το Framework Django

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τότσκας Βασίλειος (ΑΕΜ: 2480)

Μπογδαμπεΐδης Αλέξανδρος (ΑΕΜ: 2482)

Επιβλέπων Αναπληρωτης Καθηγητής: **Βέργαδος Ι. Δημήτριος**

Καστοριά, Μάρτιος 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δημιουργία Διαδικτυακού Καταστήματος Χρησιμοποιώντας Το Framework Django

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τότσκας Βασίλειος (ΑΕΜ: 2480)

Μπογδαμπεΐδης Αλέξανδρος (ΑΕΜ: 2480, 2482)

Επιβλέπων Αναπληρωτης Καθηγητής: **Βέργαδος Ι. Δημήτριος**

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης

.....	.....	.....
Ονοματεπώνυμο Μέλους	Ονοματεπώνυμο Μέλους	Ονοματεπώνυμο Μέλους
Ιδιότητα Μέλους	Ιδιότητα Μέλους	Ιδιότητα Μέλους

Καστοριά, Μάρτιος 2024

ΑΦΙΕΡΩΜΑΤΑ ΚΑΙ ΕΥΧΑΡΙΣΤΙΕΣ

Τότσκας Βασίλειος

Αφιερώνω την διπλωματική μου εργασία στους γονείς μου, Νικόλαο και Στυλιανή, στον αδερφό μου Γιάννη και στις γιαγιάδες μου.

Μπογδαμπεΐδης Αλέξανδρος

Αφιερώνω την διπλωματική μου εργασία σε όλη την οικογένεια μου,στους γονείς μου Ευθύμιο και Μαρούλα, και στα αδέρφια μου Γιώργο, Νικολέττα και Κοραλία.

Θα θέλαμε να ευχαριστήσουμε ιδιαίτερα τον επιβλέποντα Καθηγητή κ. Βέργαδο Ι. Δημήτριο καθώς και όλους τους καθηγητές και συμφοιτητές με τους οποίους συμπορευτήκαμε κατά τη διάρκεια της ακαδημαϊκής μας πορείας, διότι η συμβολή τους ήταν καθοριστική για να καταφέρουμε να φτάσουμε στο σημερινό μας επίπεδο.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη μιας διαδικτυακής πλατφόρμας για ηλεκτρονικό εμπόριο, η οποία ενσωματώνει ένα διαχειριστικό περιβάλλον (Admin Panel).

Η εφαρμογή αυτή αναπτύχθηκε με τη χρήση του Django Framework για την υλοποίηση του Backend και του Vue.js (Nuxt3 SSR) Framework για το Frontend, προσφέροντας μια σύγχρονη, δυναμική εμπειρία πλοήγησης.

Επιπλέον, η εργασία περιλαμβάνει την ανάπτυξη μιας εφαρμογής για τη βελτιστοποίηση εικόνων με τη χρήση του Nest.js Framework, στοχεύοντας στην αύξηση της απόδοσης και της ταχύτητας φόρτωσης των περιεχομένων.

Μέσα από την ανάπτυξη και την εφαρμογή αυτών των τεχνολογιών, κατέστη δυνατή η εμβάθυνση στις λειτουργίες και τις δυνατότητες που προσφέρουν τα εν λόγω Frameworks, καθώς και η βελτίωση της εμπειρίας χρήσης τόσο για τον καταναλωτή όσο και για το εργατικό προσωπικό.

Συνολικά, η πτυχιακή αυτή εργασία συμβάλλει στη δημιουργία μιας αποτελεσματικής και φιλικής προς τον χρήστη διαδικτυακής πλατφόρμας, με έμφαση στη γρήγορη και ποιοτική πρόσβαση και διαχείριση.

ΛΕΞΕΙΣ - ΚΛΕΙΔΙΑ

Frameworks, Django, Python, Typescript, HTML, CSS-SCSS, Vue, Nuxt, Database, Postgresql, Docker, Github, Redis, Ngix, Ci, Rest API, Backend, Frontend, Eshop, Ecommerce, Authentication, Nest.js, Vite, SSR

ABSTRACT

This thesis focuses on the development of a web platform for e-commerce, which incorporates an Admin Panel.

This application was developed using the Django Framework for the Backend and the Vue.js (Nuxt3 SSR) Framework for the Frontend, providing a modern, dynamic navigation experience.

In addition, the work includes the development of an application for image optimization using the Nest.js Framework, aiming to increase the performance and loading speed of the content.

Through the development and implementation of these technologies, it was possible to deepen the functionality and capabilities offered by these Frameworks, as well as improve the user experience for both the consumer and the workforce.

Overall, this thesis contributes to the creation of an efficient and user-friendly web platform, with a focus on fast and quality access and management.

KEYWORDS

Frameworks, Django, Python, Typescript, HTML, CSS-SCSS, Vue, Nuxt, Database, Postgresql, Docker, Github, Redis, Ngix, Ci, Rest API, Backend, Frontend, Eshop, Ecommerce, Authentication, Nest.js, Vite, SSR

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Ως μέρος της διατριβής μας, χρησιμοποιήθηκε το Framework δομής δεδομένων Django για να δημιουργηθεί ένα διαδικτυακό κατάστημα με στόχο τη βαθιά μάθηση και κατανόηση του πλαισίου και των δυνατοτήτων που προσφέρει.

Κατ' αρχάς, στο πρώτο κεφάλαιο παρουσιάζουμε μία εισαγωγή στις πλατφόρμες ανάπτυξης ηλεκτρονικών καταστημάτων, αναλύοντας τα είδη τους, τα Framework ιστού και προβάλλουμε κάποια επιμέρους παραδείγματα. Έπειτα, στο δεύτερο κεφάλαιο κάνουμε ανάλυση της διαδικτυακής εφαρμογής Grooveshop και των λειτουργιών της. Στο επόμενο κεφάλαιο, περιγράφουμε αναλυτικά την Αρχιτεκτονική της εφαρμογής μαζί με κάποια από τα εργαλεία που χρησιμοποιήσαμε. Το κεφάλαιο τέσσερα είναι αφιερωμένο στην εμπειρία και διεπαφή του χρήστη. Επίσης αναφέρουμε τη μεθοδολογία ανάπτυξης μέσω διαγραμμάτων ανάλυσης, ώστε να γίνει απολύτως κατανοητός ο τρόπος με τον οποίο δομήθηκε το συγκεκριμένο ηλεκτρονικό κατάστημα.

Σκοπός και Στόχοι

Στόχος του ηλεκτρονικού μας καταστήματος είναι να παρουσιάζει τα προϊόντα της αποθήκης με ευκολονόητο τρόπο, να διευκολύνει τη διαδικασία εύρεσης του συγκεκριμένου προϊόντος από τον πελάτη, να παρέχει στον πελάτη σχετικές πληροφορίες για το προϊόν, να τον καθοδηγεί στη διαδικασία παραγγελίας καθώς και να γνωρίζει το κόστος της παραγγελίας του. Αυτά τα στοιχεία εμφανίζονται στο ηλεκτρονικό κατάστημα και ενημερώνονται αυτόματα. Το άτομο που διαχειρίζεται την εφαρμογή είναι υπεύθυνο για την καταγραφή των στοιχείων που είναι απαραίτητα για τη λειτουργία της εφαρμογής. Αυτή η εφαρμογή παρέχει επίσης άλλες υπηρεσίες που αναφέρουμε και εξηγούμε παρακάτω.

Σκοπός της παρούσας Πτυχιακής εργασίας είναι η χρήση σύγχρονης τεχνολογίας προγραμματισμού για τη δημιουργία μιας εφαρμογής που διαχειρίζεται μια αποθήκη και το ηλεκτρονικό της κατάστημα. Ο στόχος επίσης είναι να μάθουμε τα πλαίσια Django, Nuxtjs, Vue.js, Nest.js και άλλες απαραίτητες τεχνολογίες μέσω αυτής της διαδικασίας.

ΚΕΦΑΛΑΙΟ 1. Πλατφόρμες Ανάπτυξης Ηλεκτρονικών Καταστημάτων

1.1 Τι είναι μια πλατφόρμα ανάπτυξης ηλεκτρονικού εμπορίου

Εν έτη 2024, πιστεύουμε ότι απαιτείται διαδικτυακή παρουσία για όλες τις επιχειρήσεις. Η παρουσία στο διαδίκτυο δεν σημαίνει τη δημιουργία λογαριασμών ή σελίδων σε κοινωνικά δίκτυα όπως το Instagram ή το Facebook. Φυσικά και τα social media αποτελούν σημαντικό μέρος της διαδικτυακής παρουσίας κάθε επιχείρησης, αλλά δεν αρκεί η αποκλειστική χρήση τους. Είναι αναγκαίο τα καταστήματα που πωλούν προϊόντα, ανεξαρτήτως τύπου, να παρέχουν στους πελάτες τη δυνατότητα αγοράς μέσω ηλεκτρονικού καταστήματος ή αλλιώς E-Shop.

Η επιλογή της κατάλληλης πλατφόρμας για τη δημιουργία του ηλεκτρονικού μας καταστήματος είναι πλέον η πιο σημαντική παράμετρος για την εμπορική επιτυχία κάθε καταστήματος στο World Wide Web. Μερικοί από τους παράγοντες που επηρεάζουν την επιλογή του «εργαλείου» για τη δημιουργία ενός ηλεκτρονικού καταστήματος είναι ο διαθέσιμος προϋπολογισμός, τα προϊόντα, αλλά αναπόφευκτα οι στόχοι που θέτει κάθε εταιρεία.

1.2 Τι εννοούμε με τον όρο Framework

Ένα πλαίσιο είναι σαν ένα σύνολο κομματιών παζλ που μπορούμε να χρησιμοποιήσουμε για να φτιάξουμε οτιδήποτε θέλουμε. Η πλατφόρμα παρέχει ένα σύνολο βασικών ενοτήτων προγραμματισμού, εργαλείων και βιβλιοθηκών που χρησιμοποιούνται για τη δημιουργία προϊόντων λογισμικού.

Τα Frameworks παρέχουν στους προγραμματιστές τα εργαλεία και τη λειτουργικότητα που χρειάζονται και καθορίζουν κανόνες για τη δημιουργία της αρχιτεκτονικής ιστότοπων, εφαρμογών, API και υπηρεσιών. Έτσι, ο βασικός πυρήνας του έργου μας εγκαθιδρύεται γρήγορα και μπορεί να επεκταθεί περαιτέρω σύμφωνα με συγκεκριμένες απαιτήσεις.

Ένα πλαίσιο πλατφόρμας μπορεί να περιλαμβάνει στοιχεία, βοηθητικά προγράμματα, βιβλιοθήκες κώδικα, γλώσσες δέσμης ενεργειών και άλλο λογισμικό που διευκολύνουν την ανάπτυξη και την ενσωμάτωση.

Τα Frameworks είναι προσαρμόσιμα πράγματα, που σημαίνει ότι μπορούμε να πάρουμε έτοιμα προς χρήση πρότυπα και στοιχεία και να τα προσαρμόσουμε με βάση τις ανάγκες μας.

1.2.1 Ποια η σημασία των Frontend και Backend Frameworks

Οι εφαρμογές Ιστού αποτελούνται από ένα τμήμα υποστήριξης ή διακομιστή και ένα τμήμα διεπαφής ή πελάτη. Υπάρχουν διαφορετικά Frontend και Backend Frameworks ανάλογα με τις ανάγκες του προγραμματιστή.

Ένα Frontend Framework είναι υπεύθυνο για τη διεπαφή χρήστη, το οπτικό μέρος του ιστότοπου ή της εφαρμογής με το οποίο αλληλεπιδρά ο τελικός χρήστης. Βασίζονται σε γλώσσες προγραμματισμού Frontend, όπως HTML, CSS και JavaScript, και ασχολούνται με το σχεδιασμό UI/UX, τα πρότυπα, τα αποσπάσματα κώδικα, τη βελτιστοποίηση SEO, τη διαχείριση αλληλεπίδρασης με τον χρήστη και άλλα.

Ένα Backend Framework είναι υπεύθυνο για τα κρυφά μέρη ενός ιστότοπου ή μιας εφαρμογής με τα οποία αλληλεπιδρούν οι προγραμματιστές. Καλύπτει λειτουργίες διακομιστή και βάσεων δεδομένων, λογική και αρχιτεκτονική λύσεων, πρωτόκολλα δρομολόγησης, ασφάλεια δεδομένων, επιλογές εξουσιοδότησης και πολλά άλλα. Αυτές οι πλατφόρμες βασίζονται σε γλώσσες προγραμματισμού Backend όπως Python, Java, Ruby, PHP και .NET.

1.2.2 Τα προτερήματα της χρήσης των Frameworks

Οι περισσότεροι προγραμματιστές επιλέγουν να χρησιμοποιούν Frameworks που ήδη υπάρχουν. Τα προκατασκευασμένα πλαίσια μπορούν να είναι ένας πολύ καλός τρόπος για να ξεκινήσουμε με τον προγραμματισμό. Συνήθως είναι πολύ καλοφτιαγμένα και έχουν όλα τα χαρακτηριστικά που χρειάζεται ένας προγραμματιστής, χωρίς να χρειάζεται να μάθει κωδικοποίηση.

Δημιουργήθηκαν Frameworks για να βοηθήσουν στην ταχεία ανάπτυξη, η οποία με τη σειρά της έχει ως αποτέλεσμα την εξοικονόμηση χρόνου και χρημάτων. Με τη χρήση Framework, οι προγραμματιστές μπορούν να χρησιμοποιήσουν προκατασκευασμένα εργαλεία και πρότυπα για να εξοικονομήσουν χρόνο, με αποτέλεσμα ένα έργο πιο εκλεπτυσμένο και λεπτομερές.

Η ανάπτυξη ιστού είναι μια σοβαρή επιχείρηση και η αξιοπιστία και η ασφάλεια είναι απαραίτητες προκειμένου να διατηρηθεί η ομαλή λειτουργία των πραγμάτων. Έτοιμα εξαρτήματα έχουν δημιουργηθεί και βελτιωθεί από μια κοινότητα εθελοντών που έχουν δοκιμαστεί σε όλα τα πιθανά σενάρια. Αυτό σημαίνει ότι είναι αξιόπιστα και λειτουργούν καλά στα πιο συνηθισμένα σενάρια. Χρησιμοποιώντας ένα υπάρχον Framework, αποφεύγουμε πολλά από τα κοινά σφάλματα και δημιουργούμε μια εξαιρετικά σταθερή, αξιόπιστη και ασφαλή λύση σε μικρότερο χρονικό διάστημα, προς ικανοποίηση πελατών και δημιουργών.

Οι προγραμματιστές των Frameworks συχνά εστιάζουν στην καλύτερη απόδοση, επειδή αυτό είναι ένα κρίσιμο τεχνικό χαρακτηριστικό. Οι λύσεις που βασίζονται σε Framework τείνουν να εκτελούνται πιο γρήγορα και να διασφαλίζουν μεγαλύτερη χωρητικότητα φόρτωσης, κάτι που είναι σημαντικό για τις διαδικτυακές λύσεις. Είναι εύκολο να επεκταθούν, πράγμα που σημαίνει ότι μπορούν να χειριστούν περισσότερους πελάτες γρήγορα.

1.3 Σύνοψη ορισμένων πλαισίων ανάπτυξης

1.3.1 Next.js

Πηγή: https://nextjs.org

Το Next.js είναι ένα πλαίσιο React που διευκολύνει τη δημιουργία ιστότοπων που φαίνονται υπέροχα στο frontend και επίσης λειτουργούν ομαλά στο backend. Το React είναι μια βιβλιοθήκη JavaScript που χρησιμοποιείται παραδοσιακά για τη δημιουργία εφαρμογών Ιστού που αποδίδονται στο πρόγραμμα περιήγησης του πελάτη με JavaScript. Frameworks όπως το Next.js παρακάμπτουν αυτά τα προβλήματα επιτρέποντας σε μέρος ή ολόκληρο τον ιστότοπο να αποδοθεί από την πλευρά του διακομιστή πριν αποσταλεί στον πελάτη. Το Next.js είναι ένα από τα πιο δημοφιλή πλαίσια για το React. Είναι μία από τις πολλές συνιστώμενες "αλυσίδες εργαλείων" που είναι διαθέσιμες κατά την εκκίνηση μιας νέας εφαρμογής, οι οποίες παρέχουν όλες ένα επίπεδο αφαίρεσης για να βοηθήσουν σε κοινές εργασίες.

Το Next.js απαιτεί Node.js και μπορεί να αρχικοποιηθεί χρησιμοποιώντας το Node Package Manager. Οι προγραμματιστές αντιμετωπίζουν προβλήματα με αυτήν τη στρατηγική, όπως η μη παροχή τροφοδοσίας σε χρήστες που δεν έχουν πρόσβαση σε JavaScript, πιθανά ζητήματα ασφάλειας, σημαντικά εκτεταμένοι χρόνοι φόρτωσης σελίδων και βλάβη στη συνολική βελτιστοποίηση μηχανών αναζήτησης του ιστότοπου.

1.3.2 Laravel

Πηγή: https://laravel.com

Το Laravel είναι ένα πλαίσιο διαδικτυακής εφαρμογής με εκφραστική και κομψή σύνταξη. Το Laravel είναι ένα πλαίσιο web που παρέχει μια δομή και ένα σημείο εκκίνησης για τη δημιουργία της εφαρμογής. Ο χρήστης μπορεί να επικεντρωθεί στη δημιουργία κάτι εκπληκτικού χρησιμοποιώντας αυτό το πλαίσιο.

Το Laravel προσπαθεί να προσφέρει μια καθηλωτική εμπειρία ανάπτυξης, παρέχοντας ταυτόχρονα ισχυρά χαρακτηριστικά, όπως έγχυση εξάρτησης, εκφραστική αφαίρεση βάσης δεδομένων, ουρές, προγραμματισμένες εργασίες, δοκιμές μονάδων ενοποίησης και πολλά άλλα.

1.3.3 Nest.js

Πηγή: https://nestjs.com

Το Nest είναι ένα Framework για τη δημιουργία εφαρμογών από την πλευρά του διακομιστή που είναι αποτελεσματικές και επεκτάσιμες. Το Nest είναι ένα προοδευτικό Framework κατασκευασμένο με JavaScript και υποστηρίζει πλήρως το TypeScript και συνδυάζει στοιχεία OOP (Αντικειμενοστρεφής προγραμματισμός), FP (Λειτουργικός Προγραμματισμός) και FRP (Λειτουργικός αντιδραστικός προγραμματισμός).

Το Nest χρησιμοποιεί ισχυρά πλαίσια διακομιστή HTTP, όπως το Express (η προεπιλογή) και προαιρετικά μπορεί να διαμορφωθεί ώστε να χρησιμοποιεί και το Fastify.

Το Nest παρέχει ένα επίπεδο αφαίρεσης πάνω από αυτά τα κοινά πλαίσια Node.js (Express/Fastify), αλλά επίσης εκθέτει τα API τους απευθείας στον προγραμματιστή. Αυτό δίνει στους προγραμματιστές την ελευθερία να χρησιμοποιούν τις μυριάδες ενότητες τρίτων που είναι διαθέσιμες για την υποκείμενη πλατφόρμα.

1.3.4 Fast API

Πηγή: https://fastapi.tiangolo.com

Το FastAPI είναι ένα σύγχρονο, γρήγορο (υψηλής απόδοσης) Web Framework για τη δημιουργία API στην Python v3.7+, που βασίζεται σε τυπικές υποδείξεις τύπου Python.

Κάποια από τα βασικά του χαρακτηριστικά είναι η πολύ υψηλή απόδοση στο ίδιο επίπεδο με το NodeJS και το Go (χάρη στο Starlette και το Pydantic). Ένα από τα πιο γρήγορα διαθέσιμα πλαίσια Python. Αυξάνει την ταχύτητα ανάπτυξης χαρακτηριστικών κατά

περίπου 200% έως 300%, προσφέρει περίπου 40% μείωση των ανθρώπινων σφαλμάτων (προγραμματιστών), καλή υποστήριξη Editor καθώς και μειωμένο χρόνο εντοπισμού σφαλμάτων. Επίσης χαρακτηρίζεται για την σχεδίαση του, ώστε να είναι εύκολο στη χρήση και εύκολο στην εκμάθηση αλλά και για τον λιγότερο χρόνο που απαιτείται για την ανάγνωση του documentation. Μειώνει τόσο την επανάληψη του ίδιου κώδικα όσο και τα σφάλματα που μπορεί να προκύψουν. Είναι βασισμένο σε ανοιχτά πρότυπα API (πλήρως συμβατό): OpenAPI (παλαιότερα γνωστό ως Swagger) και JSON Schema.Υποστηρίζει πλήρως τον ασύγχρονο προγραμματισμό και μπορεί να τρέξει σε διακομιστές παραγωγής Gunicorn και ASGI όπως οι Uvicorn και Hypercorn.

1.3.5 Svelte

Πηγή: https://svelte.dev

Το Svelte είναι ένα component Framework όπως το React και το Vue, αλλά με μια σημαντική διαφορά. Τα παραδοσιακά Frameworks επιτρέπουν την γραφή του δηλωτικού κώδικα που βασίζεται στην κατάσταση, αλλά έχουν μειονεκτήματα. Τα προγράμματα περιήγησης πρέπει να κάνουν επιπλέον δουλειά για να μεταφράσουν αυτές τις δηλωτικές δομές (declarative structures) σε χειρισμούς DOM χρησιμοποιώντας τεχνικές όπως η εικονική διαφοροποίηση DOM (virtual DOM diffing), η σπατάλη προϋπολογισμών πλαισίων και η φορολόγηση του συλλέκτη σκουπιδιών (garbage collector).

Αντίθετα, το Svelte μετατρέπει τα στοιχεία σε εξαιρετικά αποτελεσματικό επιτακτική κώδικα που εκτελείται κατά το χρόνο κατασκευής και ενημερώνει χειρουργικά το DOM. Ως αποτέλεσμα, μπορούμε να δημιουργήσουμε φιλόδοξες εφαρμογές με εξαιρετικά χαρακτηριστικά απόδοσης.

Η πρώτη έκδοση του Svelte ήταν μόνο για τον έλεγχο υποθέσεων. Αυτό σημαίνει ότι ένας αποκλειστικός μεταγλωττιστής μπορεί να δημιουργήσει ισχυρό κώδικα που παρέχει εξαιρετική εμπειρία χρήστη, το δεύτερο είναι μια μικρή αναβάθμιση.

Η έκδοση 3 είναι μια σημαντική αναμόρφωση. Τους τελευταίους 5-6 μήνες, έχει επικεντρωθεί στην παροχή μιας εξαιρετικής εμπειρίας προγραμματιστή. Τα εξαρτήματα μπορούν τώρα να δημιουργηθούν με πολύ λιγότερο boilerplate από ό,τι αλλού. Εάν ο χρήστης είναι εξοικειωμένος με άλλα Frameworks θα εκπλαγεί ευχάριστα.

1.4 Βιβλιοθήκες (Libraries) και Πακέτα (Packages)

Κατά την ανάπτυξη της ιστοσελίδας μας, χρησιμοποιήσαμε διάφορες βιβλιοθήκες τρίτων (3rd party libraries) για να ενισχύσουμε και να επεκτείνουμε τις δυνατότητες των Frameworks που χρησιμοποιήθηκαν (Django, Nuxt3 και Nestjs) και να βελτιώσουμε την απόδοση και την ασφάλεια της εφαρμογής μας. Αυτές οι βιβλιοθήκες είναι κρίσιμες για τη λειτουργία της εφαρμογής και παρέχουν διάφορες λειτουργίες και εργαλεία.

Το pip είναι το πρόγραμμα εγκατάστασης πακέτων της Python. Έχουμε την δυνατότητα να εγκαταστήσουμε πακέτα από το Ευρετήριο Πακέτων Python και άλλα ευρετήρια. Μπορούμε να εγκαταστήσουμε όσες Python Βιβλιοθήκες χρειαζόμαστε για το Django Project μας μέσω ενός requirements.txt αρχείου τρέχοντας την εντολή pip install -r requirements.txt.

Βιβλιοθήκες που χρησιμοποιήθηκαν στο Django:

- asgiref
- black
- boto3
- celery
- channels
- channels-redis
- chardet
- charset_normalizer
- click
- coverage
- django
- django-admin-thumbnails
- django-allauth[mfa]
- django-browser-reload
- django-celery-beat
- django-celery-results
- django-cors-headers
- django-debug-toolbar
- django-filter
- django-js-asset
- django-money
- django-mptt
- django-otp
- django-parler
- django-parler-rest
- django-phonenumber-field[phonenumberslite]
- django-rosetta
- django-storages
- django-stubs
- django-tinymce
- djangorestframework
- djangorestframework-camel-case
- djangorestframework-simplejwt

- dj-rest-auth
- dotty-dict
- drf-spectacular
- Faker
- flake8
- flower
- gitpython
- gunicorn
- hiredis
- httptools
- importlib-resources
- jinja2
- measurement
- multidict
- phonenumbers
- pillow
- pre-commit
- psycopg[binary]
- pydantic
- pydocstyle
- pylint
- python-dotenv
- python-gitlab
- python-semantic-release
- pytest
- pytest-asyncio
- pytest-django
- qrcode
- rich
- urllib3
- uvicorn[standard]
- websockets
- whitenoise

Το Node JS είναι ένα αυτόνομο πλαίσιο ανάπτυξης διαδικτυακών εφαρμογών. Χρησιμοποιήσαμε μόνο το Node Package Manager (npm) που παρέχεται από αυτό το πλαίσιο, για να μπορέσουμε να «συνθέσουμε» ή πιο σωστά να ενσωματώσουμε σύνθετα πακέτα, τα οποία αποτελούνται κυρίως από γλώσσα JavaScript και όχι μόνο, στην ιστοσελίδα μας. Μπορούμε να εγκαταστήσουμε τις Node βιβλιοθήκες που χρειαζόμαστε μέσω ενός package.json αρχείου τρέχοντας την εντολή npm install.

Βιβλιοθήκες που χρησιμοποιήθηκαν στο Nuxt3:

- @babel/core
- @headlessui/vue
- @iconify/json
- @intlify/core-base
- @intlify/message-compiler
- @intlify/shared
- @intlify/vue-i18n-bridge

- @intlify/vue-router-bridge
- @nuxt/devtools
- @nuxt/image
- @nuxt/kit
- @nuxt/schema
- @nuxt/types
- @nuxt/ui
- @nuxtjs/eslint-config
- @nuxtjs/eslint-config-typescript
- @nuxtjs/eslint-module
- @nuxtjs/i18n
- @parcel/watcher
- @pinia/nuxt
- @semantic-release/changelog
- @semantic-release/commit-analyzer
- @semantic-release/git
- @semantic-release/github
- @semantic-release/npm
- @semantic-release/release-notes-generator
- @sindresorhus/slugify
- @tailwindcss/typography
- @testing-library/jest-dom
- @types/js-yaml
- @types/uuid
- @unhead/schema
- @unhead/vue
- @vee-validate/nuxt
- @vee-validate/zod
- @vite-pwa/nuxt
- @vue/test-utils
- @vuepic/vue-datepicker
- @vueuse/core
- @vueuse/integrations
- @vueuse/nuxt
- @vueuse/shared
- autoprefixer
- cookie-es
- defu
- eslint
- eslint-config-prettier
- eslint-plugin-nuxt
- eslint-plugin-prettier
- h3
- idb-keyval
- is-https
- jose
- js-cookie
- js-yaml
- lint-staged
- lottie-web
- magic-string
- nitropack
- nuxt
- nuxt-og-image
- nuxt-schema-org

- nuxt-simple-robots
- nuxt-simple-sitemap
- ofetch
- pinia
- playwright
- postcss
- postcss-html
- postcss-import
- postcss-loader
- prettier
- sass
- sass-loader
- semantic-release
- serve
- std-env
- string-replace-loader
- sweetalert2
- tailwindcss
- translate
- ts-node
- tsc-alias
- typescript
- ufo
- unplugin-auto-import
- unplugin-icons
- unplugin-vue-components
- unstorage
- uuid
- vee-validate
- vite
- vitest
- vue
- vue-i18n
- vue-i18n-routing
- vue-router
- vue-tsc
- webpack
- workbox-build
- workbox-window
- zod

Βιβλιοθήκες που χρησιμοποιήθηκαν στο Nestjs:

- @nestjs/axios
- @nestjs/common
- @nestjs/core
- @nestjs/mapped-types
- @nestjs/platform-express
- lodash
- reflect-metadata
- rimraf
- rxjs
- sharp

- uuid
- axios
- @nestjs/cli
- @nestjs/schematics
- @nestjs/testing
- @semantic-release/changelog
- @semantic-release/commit-analyzer
- @semantic-release/git
- @semantic-release/github
- @semantic-release/npm
- @semantic-release/release-notes-generator
- @swc/cli
- @swc/core
- @types/express
- @types/jest
- @types/lodash
- @types/node
- @types/sharp
- @types/supertest
- @types/uuid
- @typescript-eslint/eslint-plugin
- @typescript-eslint/parser
- eslint
- eslint-config-prettier
- eslint-plugin-prettier
- jest
- prettier
- source-map-support
- supertest
- ts-jest
- ts-loader
- ts-node
- tsconfig-paths
- typescript
- semantic-release

ΚΕΦΑΛΑΙΟ 2. Ανάπτυξη της διαδικτυακής εφαρμογής GrooveShop

2.1 Η έννοια της διαδικτυακής εφαρμογής

Εφαρμογή Ιστού είναι κάθε εφαρμογή που είναι διαθέσιμη στους χρήστες της μέσω του διαδικτύου ή του ενδοδικτύου. Το μόνο εργαλείο για τη χρήση αυτής της εφαρμογής είναι το πρόγραμμα περιήγησης του χρήστη. Ωστόσο, για να εκτελούνται σωστά αυτές οι εφαρμογές, χρειάζονται ισχυρούς υπολογιστές που έχουν σχεδιαστεί ειδικά για την εξυπηρέτηση και την παροχή υπηρεσιών σε περισσότερους από έναν χρήστες.

Επομένως, μια διαδικτυακή εφαρμογή έχει τα ακόλουθα πλεονεκτήματα:

- Η εφαρμογή δεν χρειάζεται καθόλου ή σχεδόν καθόλου χώρο στο δίσκο του χρήστη αφού ολόκληρη η εφαρμογή είναι αποθηκευμένη στον διακομιστή και μόνο κατά τη χρήση της εφαρμογής μπορεί να γίνει μεταφορά δεδομένων στην υπολογιστική μονάδα του χρήστη.

- Οι χρήστες διαδικτυακών εφαρμογών μπορούν να έχουν πρόσβαση στις εφαρμογές που θέλουν να χρησιμοποιήσουν από οποιονδήποτε υπολογιστή ή άλλη συσκευή που διαθέτει internet, χωρίς να εγκαταστήσουν κάποιο πρόσθετο λογισμικό. Δεν υπάρχει ανάγκη για συγκεκριμένη εφαρμογή, καθώς όλα τα λειτουργικά συστήματα είναι εξοπλισμένα με πρόγραμμα περιήγησης ιστού. Αυτή η ιδιότητα των εφαρμογών Ιστού είναι ιδιαίτερα σημαντική για μεγάλες επιχειρήσεις με πολλούς χρήστες που, στην περίπτωση της τοπικής εφαρμογής, θα έπρεπε να εγκαταστήσουν την εφαρμογή σε κάθε υπολογιστή ξεχωριστά.

- Συνεχίζοντας από τα παραπάνω, οι χρήστες της διαδικτυακής εφαρμογής μπορούν να τη χρησιμοποιήσουν ακόμα και όταν δεν βρίσκονται στο γραφείο. Αυτή η δυνατότητα δίνει στους χρήστες την ευελιξία να χρησιμοποιούν την εφαρμογή όπου θέλουν, επιτρέποντάς τους να εργάζονται εξ αποστάσεως ή από το σπίτι.

- Οι διαδικτυακές εφαρμογές είναι συμβατές με όλα τα λειτουργικά συστήματα. Η εφαρμογή εκτελείται μέσω ενός προγράμματος περιήγησης ιστού αντί του υπολογιστή του χρήστη, ώστε να μπορεί να εκτελεστεί σε οποιοδήποτε λειτουργικό σύστημα. Αυτό το χαρακτηριστικό οφείλεται επίσης στην τυποποίηση των γλωσσών προγραμματισμού που χρησιμοποιούνται σε εφαρμογές.

- Οι διαδικτυακές εφαρμογές δεν εκτελούνται στον υπολογιστή του χρήστη και επομένως δεν καταναλώνουν πόρους από το σύστημα. Αυτό καθιστά τις εφαρμογές web ιδιαίτερα ελαφριές στις υπολογιστικές μονάδες.

- Εάν πρέπει να αναβαθμίσουμε την εφαρμογή μας, έχουμε μεγάλο πλεονέκτημα σε σχέση με τις τοπικές εφαρμογές. Με τις παραδοσιακές τοπικές εφαρμογές, οι αναβαθμίσεις συστήματος πρέπει να γίνονται ξεχωριστά σε κάθε υπολογιστή, κάτι που είναι χρονοβόρο και δαπανηρό. Αντίθετα, με τις διαδικτυακές εφαρμογές, οι αναβαθμίσεις γίνονται μόνο στον διακομιστή που φιλοξενεί την εφαρμογή και ταυτόχρονα το αναβαθμισμένο πρόγραμμα είναι διαθέσιμο σε όλους τους χρήστες.

Με αυτόν τον τρόπο εξοικονομείται χρόνος, κάτι που είναι ιδιαίτερα πολύτιμο για τις μεγάλες εταιρείες. Επίσης, ως αποτέλεσμα των παραπάνω, το κόστος των αναβαθμίσεων μειώνεται σημαντικά καθώς απαιτείται μικρότερη προσπάθεια για τη διεκπεραίωσή τους.

- Ένα άλλο πλεονέκτημα των διαδικτυακών εφαρμογών είναι ότι με την έλευση της HTML5, οι συντάκτες εφαρμογών μπορούν εύκολα να εμπλουτίσουν τις εφαρμογές τους με τρόπους που είναι ευκολότεροι, πιο εύχρηστοι και πιο φιλικοί προς το χρήστη. Στο παρελθόν, αυτές οι εφαρμογές υστερούσαν σε εμφάνιση, αλλά τώρα είναι ιδιαίτερα εύκολο να βελτιωθούν.

- Επίσης μπορούν να χρησιμοποιηθούν εκτός σύνδεσης εάν η εφαρμογή έχει κατασκευαστεί σωστά. Για παράδειγμα, εάν η σύνδεσή μας στο διαδίκτυο διακοπεί για κάποιο λόγο, αυτό δεν θα επηρεάσει τους χρήστες μας και θα συνεχίσουν να χρησιμοποιούν κανονικά την εφαρμογή μας. Αυτό επιτυγχάνεται με το πρόγραμμα περιήγησης να διατηρεί αντίγραφο των αρχείων που απαιτούνται για χρήση εκτός σύνδεσης της εφαρμογής στον υπολογιστή του χρήστη και να τα χρησιμοποιεί όπως απαιτείται. Αυτή η δυνατότητα δεν είναι διαθέσιμη για όλες τις εφαρμογές που χρησιμοποιούν HTML5 και είναι διαθέσιμη μόνο για εφαρμογές που παρέχονται για χρήση εκτός Διαδικτύου ή Intranet.

2.2 GrooveShop

2.2.1 Περιγραφή

Η διαδικτυακή εφαρμογή που έχουμε δημιουργήσει με την ονομασία GrooveShop αποτελεί μια ολοκληρωμένη λύση για τη διαχείριση αποθηκών και την λειτουργία ηλεκτρονικών καταστημάτων (e-shop). Είναι ιδανικό για όποιον διαχειρίζεται μεγάλες αποθηκευτικές μονάδες, αντιμετωπίζοντας τα προβλήματα αποδιοργάνωσης που συνήθως εμφανίζονται σε τέτοιες εγκαταστάσεις. Η αποδιοργάνωση αυτή συχνά οδηγεί σε λάθη και καθυστερήσεις που επηρεάζουν την αποτελεσματικότητα των εργασιών. Επομένως, είναι απαραίτητη η υιοθέτηση μιας εφαρμογής που επιτρέπει την ακριβής διαχείριση του αποθέματος, ενώ ταυτόχρονα διευκολύνει την συνεργασία μεταξύ των μελών της ομάδας.

Ένα κρίσιμο στοιχείο για την επιτυχή λειτουργία του GrooveShop είναι η ασφάλεια των δεδομένων και των συναλλαγών. Το σύστημα αυθεντικοποίησης που χρησιμοποιείται βασίζεται στην τεχνολογία των JWT (JSON Web Tokens), εξασφαλίζοντας ένα ασφαλές περιβάλλον για τους χρήστες και την ιδιωτικότητα των δεδομένων τους. Επιπλέον, προσφέρεται η δυνατότητα διαπίστευσης δύο παραγόντων (Two Factor Authentication - 2FA), προσθέτοντας ένα επιπλέον επίπεδο ασφάλειας κατά την πρόσβαση στο σύστημα. Με αυτόν τον τρόπο, εξασφαλίζεται η ακεραιότητα των δεδομένων και των συναλλαγών, ενώ ταυτόχρονα αποτρέπονται οι ανεπιθύμητες προσβάσεις. Το GrooveShop είναι σχεδιασμένο ώστε να παρέχει μια ολοκληρωμένη και ασφαλή λύση για την οργανωμένη διαχείριση αποθηκών και την απρόσκοπτη λειτουργία των ηλεκτρονικών καταστημάτων.

Η εφαρμογή εξοικονομεί χρόνο ενημερώνοντας αυτόματα τα email. Ως απάντηση σε αυτή την ανάγκη δημιουργήσαμε την εφαρμογή GrooveShop, η οποία με τη σειρά της

έρχεται να επιλύσει το πρόβλημα διαδικτυακά. Η πλατφόρμα (Admin Panel) είναι προσβάσιμη μόνο σε χρήστες με τα απαιτούμενα στοιχεία ελέγχου πρόσβασης.

Στόχος της εταιρείας είναι να βοηθά τους εργαζόμενους να διαχειρίζονται τα προϊόντα τους και να λαμβάνουν τις παραγγελίες γρήγορα και αποτελεσματικά.

Οι δύο πυλώνες της εφαρμογής είναι το ηλεκτρονικό κατάστημα (eShop) και η διαχείριση της αποθήκης. Απαιτείται η εισαγωγή βασικών δεδομένων (data entries) για να λειτουργήσει η εφαρμογή.

Αρχικά οι εργαζόμενοι θα πρέπει να χωριστούν σε ομάδες με βάση τον ρόλο και τις δυνατότητες που θέλουμε να έχουν. Αφού δημιουργήσουμε αυτές τις ομάδες και τους δώσουμε τα κατάλληλα δικαιώματα, θα πρέπει και να τις εισάγουμε στην εφαρμογή. Στη συνέχεια θα πρέπει να δημιουργήσουμε τους χρήστες που θέλουμε, να καταγράψουμε τα στοιχεία τους και να τους κατηγοριοποιήσουμε σε κάποια από τις ομάδες. Με την ολοκλήρωση αυτών των ενεργειών, θα έχουμε δημιουργήσει τον χώρο εργασίας που απαιτείται για την υποστήριξη της ιδιότητας του υπαλλήλου και των ρόλων που απαιτούνται από την αποθήκη.

Στη συνέχεια θα πρέπει να δημιουργηθούν τα βασικά δεδομένα (Data Entry) τα οποία είναι:

- Προϊόντα
- Κατηγορίες προϊόντων
- Χώρες
- Περιοχές
- Φόροι (ΦΠΑ)
- Τρόποι Πληρωμής

Αφού ολοκληρωθεί η παραπάνω διαδικασία, θα πρέπει όλα τα προϊόντα να κατανεμηθούν στις κατάλληλες κατηγορίες και να γίνει η εισαγωγή των εικόνων. Ο κάθε πελάτης μπορεί να εισέλθει στην σελίδα του καταστήματος σαν εγγεγραμμένος χρήστης ή σαν φιλοξενούμενος. Ο εγγεγραμμένος χρήστης έχει την δυνατότητα να βλέπει τις παραγγελίες του, τα αγαπημένα του προϊόντα και τις αξιολογήσεις του και να επεξεργαστεί τα στοιχεία του.

Πέρα από αυτά, όλοι οι χρήστες μπορούν να χρησιμοποιήσουν την μπάρα αναζήτησης στο πάνω μέρος της εφαρμογής ή το κουμπί κατηγοριών για την αναζήτηση οποιουδήποτε προϊόντος, να προσθαφαιρέσουν στο καλάθι αγορών τα προϊόντα που τους ενδιαφέρουν και αφού συμπληρώσουν την φόρμα με τα στοιχεία τους, να ολοκληρώσουν την παραγγελία τους. Με την ολοκλήρωση της παραγγελίας, θα σταλεί ένα ενημερωτικό email στον πελάτη αλλά και στον διαχειριστή της εφαρμογής.

2.2.2 Περιγραφή λειτουργιών

Τα χαρακτηριστικά της εφαρμογής λαμβάνουν υπόψη τις ανάγκες της αποθήκης και τις υπηρεσίες των χρηστών του.

Οι διαθέσιμες λειτουργίες της διαχείρισης αποθήκης (Admin Panel) είναι:

- **Σύστημα ελέγχου εισόδου**:
  Για να διαχειριστούμε την αποθήκη με ασφάλεια, πρέπει να βεβαιωθούμε ότι μόνο εξουσιοδοτημένα άτομα έχουν πρόσβαση σε αυτήν. Για να μπορέσει ο χρήστης να χρησιμοποιήσει την εφαρμογή θα πρέπει πρώτα να είναι συνδεδεμένος στο σύστημα από το προσωπικό της αποθήκης. Η εφαρμογή θα δώσει στους χρήστες κωδικούς ώστε να μπορούν να τη χρησιμοποιήσουν. Η εφαρμογή αναζητά αυτούς τους κωδικούς για να επιτρέπει στους χρήστες να εισάγουν πληροφορίες.

- **Σύστημα απόδοσης ρόλων και δικαιωμάτων**:
  Στη δουλειά υπάρχουν ομάδες ανθρώπων που έχουν κοινό στόχο. Αυτές οι ομάδες συνεργάζονται για να κάνουν την επιχείρηση να λειτουργεί ομαλά. Η εφαρμογή μάς επιτρέπει να δημιουργήσουμε τις δικές μας ομάδες και να τους δώσουμε τα δικαιώματα που χρειάζονται για να επιτύχουν τους στόχους τους.

- **Σύστημα καταχώρησης και επεξεργασίας προσωπικού**:
  Η εφαρμογή μάς επιτρέπει να εγγράφουμε άτομα και να παρακολουθούμε τις πληροφορίες τους. Το προσωπικό που είναι υπεύθυνο για αυτό θα εισαγάγει τα δεδομένα μας και θα μας ομαδοποιήσει σύμφωνα με τις αρμοδιότητες που μας έχουν ανατεθεί. Το προσωπικό παρέχει στον νέο υπάλληλο τους κωδικούς για να μπει στην αίτηση. Ο ελεγκτής μπορεί να δει όλα τα στοιχεία του χρήστη, εκτός από τον κωδικό πρόσβασής του.

- **Σύστημα καταχώρησης και επεξεργασίας προϊόντων**:
  Οι αποθήκες βοηθούν να διατηρούνται τα προϊόντα σε καλή κατάσταση και οργανωμένα ώστε να μπορούν να πωλούνται εύκολα. Η εφαρμογή μάς επιτρέπει να καταχωρούμε προϊόντα και να βλέπουμε τις πληροφορίες τους χρησιμοποιώντας το σύστημα. Τα προϊόντα ομαδοποιούνται σε κατηγορίες και κάθε κατηγορία έχει έναν συγκεκριμένο τύπο προϊόντος. Σε κάθε σελίδα προϊόντος, θα βρούμε όλες τις πληροφορίες που χρειαζόμαστε για να βοηθήσουμε την αποθήκη μας να λειτουργεί ομαλά και να βοηθήσουμε το προσωπικό μας να πετύχει τους στόχους του.

- **Σύστημα ανάγνωσης και επεξεργασίας αξιολογήσεων προϊόντων**:
  Το αρμόδιο προσωπικό έχει την δυνατότητα να βλέπει και να εγκρίνει ποια σχόλια είναι κατάλληλα (π.χ. χωρίς βρισιές και προσβλητικά σχόλια) για να δημοσιευτούν στην σελίδα του αντίστοιχου προϊόντος.

- **Σύστημα καταχώρησης και επεξεργασίας κατηγοριών προϊόντων**:
  Η εφαρμογή μας επιτρέπει την καταχώρηση και επεξεργασία των κατηγοριών, στις οποίες μπορούμε να προσθέσουμε τα προϊόντα που επιθυμούμε.

- **Σύστημα καταχώρησης και επεξεργασίας παραγγελιών**:
  Οι παραγγελίες του ηλεκτρονικού καταστήματος είναι οι παραγγελίες που

έρχονται στην αποθήκη από πελάτες μέσω του e-shop (ηλεκτρονικού καταστήματος), τις οποίες μπορεί να επεξεργαστεί το υπεύθυνο προσωπικό. Επίσης μπορεί να δημιουργήσει χειροκίνητα μία παραγγελία σε περίπτωση που ο πελάτης επικοινωνήσει τηλεφωνικά με το κατάστημα.

- **Σύστημα καταχώρησης και επεξεργασίας χωρών και περιοχών**:
  Η εφαρμογή μας επιτρέπει την καταχώρηση των διαθέσιμων χωρών και περιοχών που υποστηρίζει για αποστολές παραγγελιών.

- **Σύστημα καταχώρησης και επεξεργασίας ποσοστών ΦΠΑ**:
  Υπάρχει δυνατότητα δημιουργίας και επεξεργασίας ποσοστών φόρων και να προσθέσει σε κάθε προϊόν τον φόρο που του αντιστοιχεί.

- **Σύστημα καταχώρησης και επεξεργασίας μεθόδων πληρωμών**:
  Μετά την εισαγωγή των διαθέσιμων τρόπων πληρωμών, οι οποίοι είναι μέσω πιστωτικής κάρτας, PayPal, πληρωμής στο φυσικό κατάστημα και αντικαταβολή, θα εμφανιστούν στην σελίδα ολοκλήρωσης των αγορών (checkout) και ο κάθε πελάτης μπορεί να επιλέξει όποιον τρόπο τον εξυπηρετεί καλύτερα.

- **Σύστημα βελτιστοποίησης SEO**:
  Ο διαχειριστής του Marketing της σελίδας μπορεί να προσθέσει τίτλους και περιγραφές για το Meta Title και Description του HTML Document σε ορισμένα στοιχεία όπως προϊόντα και κατηγορίες.

- **Σύστημα υποστήριξης πολλαπλών γλωσσών (Multi Language)**:
  Υπάρχει η δυνατότητα εισαγωγής πληροφοριών σε διαφορετικές γλώσσες για ορισμένα πεδία που έχουν να κάνουν με κείμενα όπως είναι ο τίτλος η περιγραφή και η σύντομη περιγραφή ενός προϊόντος ή κατηγορίας.

- **Σύστημα καταχώρησης και επεξεργασίας Media Sliders**:
  Στην αρχική σελίδα του ηλεκτρονικού καταστήματος υπάρχουν κάποιες θέσεις (slots) που μπορούν να τοποθετηθούν Media Sliders (εικόνες, βίντεο) με την εισαγωγή τους από τον αρμόδιο.

- **Ιστορικό ενεργειών στην εφαρμογή**:
  Στην αρχική σελίδα της εφαρμογής Διαχείριση αποθήκης, θα βρούμε έναν πίνακα στα δεξιά με πληροφορίες σχετικά με τις τελευταίες ενέργειες που έγιναν στην εφαρμογή καθώς και τον αντίστοιχο διαχειριστή ή προσωπικό που έκανε αυτές τις ενέργειες.

- **Αποστολή μηνύματος ηλεκτρονικού ταχυδρομείου στον πελάτη κατά την ολοκλήρωση της παραγγελίας**:
  Όταν το άτομο που έκανε την παραγγελία για το ηλεκτρονικό κατάστημα την ολοκληρώσει, θα αποσταλεί ένα email στον πελάτη που θα τον ενημερώνει ότι η παραγγελία του έχει ολοκληρωθεί και μπορεί να την παραλάβει από το κατάστημα εάν έχει επιλέξει αυτόν τον τρόπο παραλαβής ή ότι η παραγγελία είναι καθ' οδόν εάν έχουν επιλέξει την αντικαταβολή.

Οι διαθέσιμες λειτουργίες της σελίδας του ηλεκτρονικού καταστήματος είναι:

- **Πλοήγηση και παρουσίαση των προϊόντων της αποθήκης**:
  Στο header του ηλεκτρονικού καταστήματος βρίσκεται ένα κουμπί που με το πάτημα του εμφανίζει τις διαθέσιμες κατηγορίες που χωρίζονται τα προϊόντα. Με την επιλογή κάποιας κατηγορίας θα περιηγηθούμε στην σελίδα παρουσίασης των προϊόντων.
  Η γραμμή αναζήτησης που βρίσκεται περίπου στο κεντρο της μπάρα πλοήγησης (Navigation Bar) μπορεί να βοηθήσει στην εύρεση στοχευμένων αποτελεσμάτων. Το λογότυπο στα αριστερά είναι το κύριο λογότυπο του ιστότοπου, πατώντας το με click μάς μεταφέρει στην αρχική σελίδα του καταστήματος . Το καλάθι αγορών περιλαμβάνει μια λίστα με προϊόντα που έχουν προστεθεί και μπορούμε να το επεξεργαστούμε κάνοντας click στο κουμπί με το εικονίδιο του καλαθιού το οποίο βρίσκεται στη δεξιά μεριά της μπάρας.

- **Χώρος παρουσίασης Media Slider**:
  Στην αρχική σελίδα (Homepage) υπάρχουν κάποιες θέσεις όπου εμφανίζονται sliders για την παρουσίαση πολυμέσων (Εικόνες, Βίντεο) , τους οποίους μπορεί να δημιουργήσει και να επεξεργαστεί ο αρμόδιος εργαζόμενος του καταστήματος.

- **Χώρος παρουσίασης προτεινόμενων προϊόντων**:
  Σε ένα σημείο της αρχικής σελίδας βρίσκεται ένα περιεχόμενο όπου εμφανίζονται κάποια προτεινόμενα προϊόντα τα οποία μπορούν να επιλεχθούν μέσω της σελίδας διαχείρισης.

- **Προβολή στοιχείων προϊόντων**:
  Στη σελίδα του κάθε προϊόντος βρίσκονται χρήσιμες πληροφορίες όπως οι φωτογραφίες, ο τίτλος, ο κωδικός, η περιγραφή, η τελική τιμή και έκπτωση σε περίπτωση που υπάρχει καθώς και η διαθεσιμότητα του προϊόντος.

- **Προσθήκη προϊόντων στα αγαπημενα (Εγγεγραμμένοι χρήστες)**:
  Υπάρχει η δυνατότητα προσθήκης ενός προϊόντος στη λίστα αγαπημένων πατώντας click στο κουμπί με το εικονίδιο της καρδιάς το οποίο εμφανίζεται με κόκκινο χρώμα στα προϊόντα που ο χρήστης ήδη έχει προσθέσει στη λίστα.

- **Καταχώρηση, επεξεργασία και διαγραφή αξιολογήσεων προϊόντων (Εγγεγραμμένοι χρήστες)**:
  Οι εγγεγραμμένοι χρήστες της εφαρμογής μπορούν να αξιολογήσουν ένα προϊόν με μια κλίμακα βαθμολογίας από το 1 έως το 10 και να αφήσουν καθώς και να αφήσουν κάποιο σχόλιο. Υπάρχει επίσης η δυνατότητα επεξεργασίας ή διαγραφής μιας αξιολόγησης. Οι μη εγγεγραμμένοι χρήστες μπορούν απλά να δουν αυτές τις αξιολογήσεις.

- **Λειτουργία παραγγελιών**:
  Ο χρήστης μπορεί να παραγγείλει τα προϊόντα που τον ενδιαφέρουν μέσω του ηλεκτρονικού καταστήματος. Η σελίδα περιγραφής προϊόντος και η κάρτα προϊόντος έχουν κουμπί προσθήκης στο καλάθι. Πατώντας το κουμπί, το προϊόν θα μεταφερθεί στο καλάθι αγορών του χρήστη, όπου μπορεί να ολοκληρωθεί η διαδικασία παραγγελίας. Το Καλάθι Παραγγελιών επιτρέπει στους χρήστες να επιλέξουν πόσα από τα επιλεγμένα προϊόντα τους θα ήθελαν να παραγγείλουν, να αφαιρέσουν προϊόντα και να μάθουν τον κωδικό για κάθε προϊόν. Αφού ο χρήστης εισαγάγει τις

απαιτούμενες πληροφορίες στη φόρμα που εμφανίζεται και πατήσει το κουμπί Ολοκλήρωση παραγγελίας, θα γίνει ανακατεύθυνση σε νέα σελίδα με ένα ευχαριστήριο μήνυμα.

- **Ενημέρωση των πελατών για τυχόν πρόσθετο κόστος κατα την παραγγελία**:
  Το κόστος της παραγγελίας θα υπολογιστεί στη δεξιά πλευρά αυτής της σελίδας. Θα υπάρχει επίσης μια περίληψη του κόστους. Ο χρήστης μπορεί να επιλέξει να αγοράσει από το κατάστημα ή να το πληρώσει με αντικαταβολή.

- **Σύστημα ελέγχου σε όλες τις φόρμες (Form Validation)**:
  Υπάρχει ένα σύστημα σε όλες τι φόρμες του ηλεκτρονικού καταστήματος το οποίο έχει την δυνατότητα να αποτρέπει τον χρήστη να καταχωρήσει ελλιπείς ή λανθασμένες πληροφορίες. Εμφανίζονται μηνύματα σφάλματος σε φόρμες και πεδία που έχουν τυχόν λάθος ή είναι κενά.

- **Σελιδοποίηση (Pagination)**:
  Υπάρχει ένας συγκεκριμένος αριθμός αντικειμένων σε κάποιες σελίδες. Εάν επιτευχθεί αυτός ο αριθμός, τα αντικείμενα που περισσεύουν δεν θα εμφανίζονται και ο χρήστης θα μπορεί να έχει πρόσβαση σε αυτά μόνο αλλάζοντας τη σελίδα ή πατώντας τους αριθμούς που δείχνουν πόσες σελίδες έχουν δημιουργηθεί.

- **Εγγραφή χρήστη**:
  Ο κάθε πελάτης έχει την δυνατότητα να κάνει εγγραφή ως μέλος του ηλεκτρονικού καταστήματος συμπληρώνοντας μία φόρμα με το email του και τον κωδικό πρόσβασης που θα επιλέξει. Στη συνέχεια θα λάβει ένα email επιβεβαίωσης, το οποίο περιλαμβάνει ένα link, όπου με ένα κλικ μπορεί να επιβεβαιώσει την εγγραφή του.

- **Έλεγχος ταυτότητας λογαριασμού κοινωνικής δικτύωσης (Social Account Authentication)**:
  Αυτή η λειτουργία επιτρέπει στους χρήστες να συνδέσουν τον λογαριασμό τους στο online κατάστημα χρησιμοποιώντας τα διαπιστευτήρια τους από κοινωνικά δίκτυα όπως το Facebook, το Twitter, το Google κ.λπ.

- **Έλεγχος ταυτότητας πολλαπλών παραγόντων (Multi factor authentication)**:
  Η πολυπαραγοντική ταυτοποίηση είναι ένα σημαντικό μέσο αύξησης της ασφάλειας των λογαριασμών χρηστών. Συνήθως, απαιτείται κάτι που ο χρήστης γνωρίζει (κωδικό πρόσβασης) και κάτι που έχει (π.χ. κινητό τηλέφωνο) για να συνδεθεί. Ο κάθε χρήστης έχει τη δυνατότητα να ενεργοποιησει τον κωδικό μίας χρήσης (One time password) μέσα από τις ρυθμίσεις του λογαριασμού του.

- **Αναζήτηση προϊόντος (Search)**:
  Η παραπάνω λειτουργία περιγράφει τον τρόπο με τον οποίο το online κατάστημα διευκολύνει την αναζήτηση προϊόντων από τους χρήστες. Οι βασικές λειτουργίες περιλαμβάνουν:

    1. Ο χρήστης υποβάλλει ένα ερώτημα αναζήτησης για προϊόντα, πληκτρολογώντας κείμενο στο πεδίο αναζήτησης.
    2. Ο server λαμβάνει το ερώτημα αναζήτησης από τον χρήστη.

3. Ο server χρησιμοποιεί το Django Framework και το PostgreSQL για να εκτελέσει μια σειρά ερωτημάτων στη βάση δεδομένων προκειμένου να βρει προϊόντα που ταιριάζουν με το ερώτημα αναζήτησης.
4. Τα ερωτήματα αναζήτησης εφαρμόζονται σε πολλαπλά πεδία της βάσης δεδομένων, συμπεριλαμβανομένων των ονομάτων και των περιγραφών των προϊόντων, καθώς και των μεταφράσεων αυτών (εάν υπάρχουν σε πολλές γλώσσες).
5. Ο server χρησιμοποιεί διάφορες τεχνικές αξιολόγησης αποτελεσμάτων, όπως την ανάθεση βαθμολογίας στα αποτελέσματα (search rank), τον προσδιορισμό της ομοιότητας (similarity) μεταξύ του ερωτήματος αναζήτησης και των προϊόντων, και τη δημιουργία σύντομων περιλήψεων (headlines) για τα αποτελέσματα.
6. Τα αποτελέσματα της αναζήτησης παρουσιάζονται στον χρήστη, με τη δυνατότητα περιορισμένου αριθμού αποτελεσμάτων για την εξοικονόμηση του χώρου στην οθόνη του.
7. Τα αποτελέσματα περιλαμβάνουν τα προϊόντα που ταιριάζουν με το ερώτημα αναζήτησης, με πληροφορίες όπως το όνομα του προϊόντος (που μπορεί να τονίζεται), τον βαθμό αντιστοίχισης (similarity score), τη βαθμολογία αναζήτησης (search rank), και άλλα δεδομένα που βοηθούν τον χρήστη να επιλέξει το κατάλληλο προϊόν.

- **Λογαριασμός Χρήστη**:
    1. Ιστορικό Παραγγελιών: Έχει την δυνατότητα να βλέπει όλες τις παραγγελίες που έχουν ολοκληρωθεί μαζί με όλα τα στοιχεία τους.
    2. Αγαπημένα Προϊόντα: Ο χρήστης έχει γρήγορη πρόσβαση στα Αγαπημένα του προϊόντα.
    3. Αξιολογήσεις Προϊόντων: Ο χρήστης έχει πρόσβαση στις αξιολογήσεις που έχει καταχωρήσει.
    4. Ρυθμίσεις: Έχει την δυνατότητα να επεξεργαστεί τις προσωπικές του ρυθμίσεις όπως ονοματεπώνυμο, τηλέφωνο, διεύθυνση και άλλα.
    5. Ασφάλεια (Security): Ο χρήστης μπορεί να επεξεργαστεί τους κωδικούς πρόσβασης και να αποσυνδεθεί από όλες τις συσκευές που είναι συνδεδεμένος, έχει επίσης την δυνατότητα να ενεργοποιήσει τον κωδικό μίας χρήσης (One time password) με την χρήση κάποιου Authenticator application κάνοντας scan το qr code που εμφανίζεται στην οθόνη του με την κάμερα του κινητού του. Έπειτα μπορεί να κατεβάσει την λίστα με τους 10 κωδικούς ανάκτησης (recovery codes) για την περίπτωση που δεν έχει πρόσβαση στην συσκευή του, οπως και να παράξει νέους κωδικούς.Υπάρχει και η δυνατότητα απενεργοποίησης αυτής της λειτουργικότητας.

- **Αλλαγή Κωδικού Χρήστη**:
  Σε περίπτωση που κάποιος εγγεγραμμένος χρήστης ξεχάσει τον κωδικό πρόσβασης του, μπορεί δημιουργήσει έναν καινούριο κωδικό.

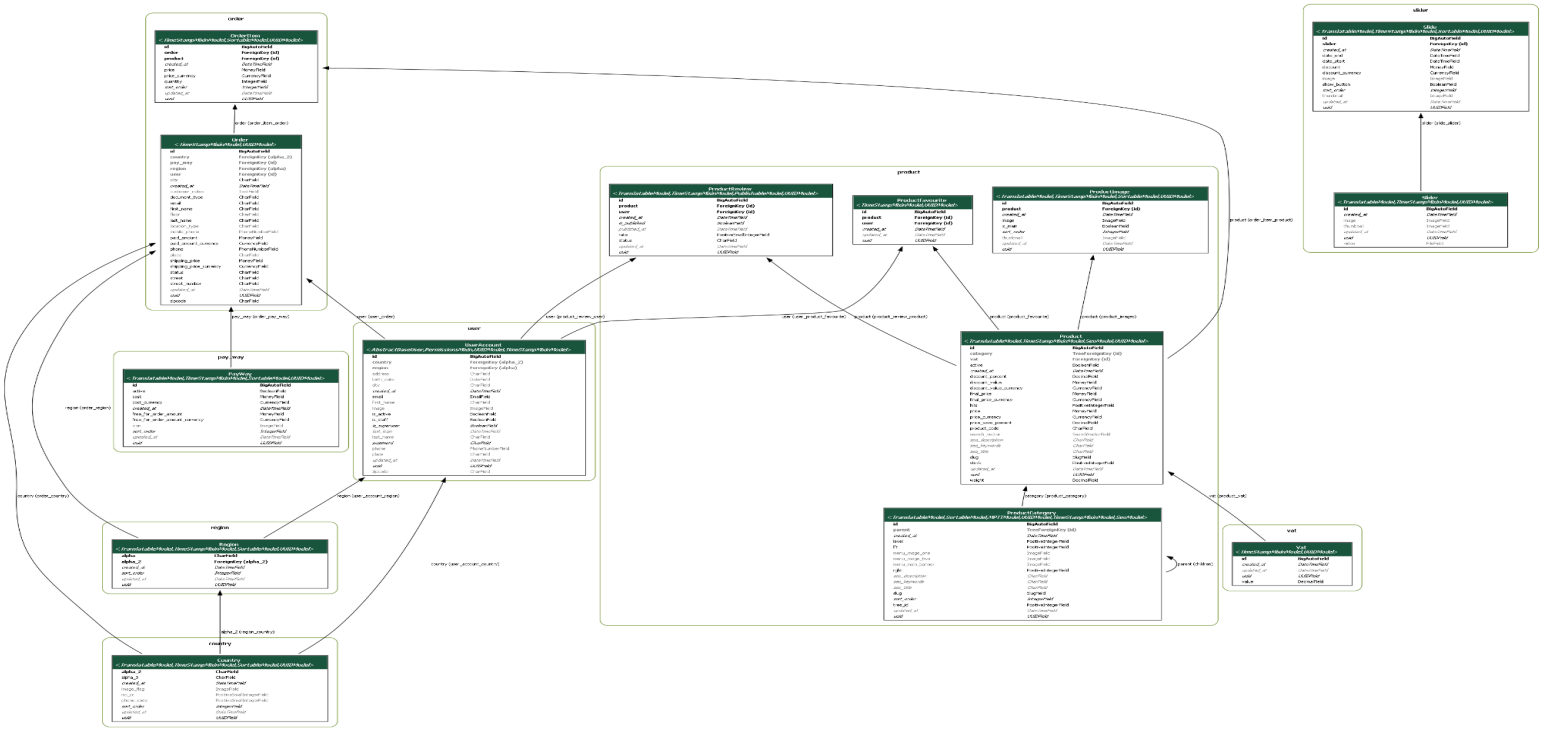
- **Επιλογή Γλώσσας**:
    1. Οι χρήστες έχουν τη δυνατότητα να αλλάζουν τη γλώσσα της ιστοσελίδας σύμφωνα με τις προτιμήσεις τους.
    2. Οι διαθέσιμες γλώσσες περιλαμβάνουν τα ελληνικά, τα αγγλικά και τα γερμανικά.

3. Η λειτουργία αυτή συνήθως παρέχεται μέσω μιας αναπτύξιμης λίστας ή ενός μενού που εμφανίζει τις διαθέσιμες γλώσσες.
4. Οι χρήστες μπορούν να επιλέξουν τη γλώσσα της προτίμησής τους από αυτό το μενού.
5. Μόλις ο χρήστης επιλέξει μια γλώσσα, η ιστοσελίδα αναλαμβάνει να εμφανίσει το περιεχόμενο στην επιλεγμένη γλώσσα.
6. Συνήθως, το περιεχόμενο της ιστοσελίδας πρέπει να είναι διαθέσιμο σε όλες τις διαθέσιμες γλώσσες, και οι χρήστες πρέπει να μπορούν να μετακινούνται εύκολα ανάμεσα στις γλώσσες.
7. Αυτή η λειτουργία βοηθά τους χρήστες που μιλούν διάφορες γλώσσες να περιηγούν και να κατανοούν την ιστοσελίδα σας στην προτιμώμενη τους γλώσσα.

- **Light/Dark Mode**:
  Υπάρχει δυνατότητα εναλλαγής θέματος (Theme mode), όπου αρχικά εμφανίζεται με βάση το θέμα του Browser του χρήστη και αλλάζει με ένα κλικ στο αντίστοιχο εικονίδιο στην μπάρα πλοήγησης.

2.2.3 Entity Relationship Diagram

Στις παρακάτω φωτογραφίες απεικονίζεται η μορφή της βάσης δεδομένων (Χωρίς τα default tables που δημιουργεί το Django) που χρησιμοποιεί η εφαρμογή σε UML διάγραμμα:



2.2.4 Αναλυτική περιγραφή ορισμένων μοντέλων και των πεδίων τους.

Παρακάτω θα περιγράψουμε τα Abstract καθώς και τα Base μοντέλα που έχουμε δημιουργήσει για την εφαρμογή (δεν θα συμπεριληφθούν τα default generated μοντέλα του Django Framework).

2.2.4.1 Abstract Models

Οι Abstract βασικές κλάσεις είναι χρήσιμες όταν θέλουμε να βάλουμε κάποιες κοινές πληροφορίες σε πολλά άλλα μοντέλα, γράφοντας την βασική κλάσης και βάζοντας abstract=True στην κατηγορία Meta. Αυτό το μοντέλο δεν θα χρησιμοποιηθεί στη συνέχεια για τη δημιουργία πίνακα βάσης δεδομένων. Αντίθετα, όταν χρησιμοποιείται ως βασική κλάση για άλλα μοντέλα, τα πεδία του θα προστεθούν σε αυτά της θυγατρικής κλάσης.

SortableModel: Ένα μοντέλο που θα χρησιμοποιηθεί οταν χρειαστεί η λειτουργία της ταξινόμησης.

Τα πεδία του SortableModel:

- sort_order: Σειρά ταξινόμησης.

TimeStampMixinModel: Χρονική σφραγίδα (Timestamp) για το πότε δημιουργηθηκε η ενημερώθηκε μια εγγραφή.

Τα πεδία του TimeStampMixinModel:

- created_at: Ημερομηνία για το πότε δημιουργήθηκε μια εγγραφή.
- updated_at: Ημερομηνία για το πότε ενημερώθηκε μια εγγραφή.

UUIDModel: Καθολικά μοναδικό αναγνωριστικό (universally unique identifier).

Τα πεδία του UUIDModel:

- uuid: Μοναδικό ID που γίνεται generate κατά την εισαγωγή μιας εγγραφής.

PublishableModel: Χρησιμοποιείται σε οντότητες που χρειάζονται δημοσιοποίηση (π.χ. Αξιολογήσεις προϊόντων).

Τα πεδία του PublishableModel:

- published_at: Ημερομηνία για το πότε δημοσιοποιήθηκε μια εγγραφή.
- is_published: Επιλογη για το αν η εγγραφη θα ειναι δημοσιευμενη.

SeoModel: Για την προσθήκη τίτλων και περιγραφών με στόχο την βελτιστοποίηση του SEO στις σελίδες όπου χρειάζεται (π.χ Σελίδα προϊόντος και κατηγορίας).

Τα πεδία του SeoModel:

- seo_title: Ο τίτλος που θα χρησιμοποιηθεί για το meta title HTML element.

- seo_description: Η περιγραφή που θα χρησιμοποιηθεί για το meta title HTML element.

2.2.4.2 Base Models

Ένα μοντέλο είναι η μοναδική, οριστική πηγή πληροφοριών σχετικά με τα δεδομένα μας. Περιέχει τα βασικά πεδία και τις συμπεριφορές των δεδομένων προς αποθήκευση. Γενικά, κάθε μοντέλο αντιστοιχίζεται σε έναν ενιαίο πίνακα βάσης δεδομένων με κάθε χαρακτηριστικό του μοντέλου να αντιπροσωπεύει ένα πεδίο βάσης δεδομένων.

Το Django μάς παρέχει ένα API τύπου ORM (Object–relational mapping) πρόσβασης σε βάση δεδομένων που δημιουργείται αυτόματα.

Στην εφαρμογή μας όλα τα μοντέλα χρησιμοποιούν ως βασικές κλάσεις τα Abstract μοντέλα TimeStampMixinModel και UUIDModel .

Product: Το μοντέλο του προϊόντος. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα TranslatableModel και SeoModel.

Τα πεδία του Product:

- product_code: Ο κωδικός αναφοράς του προϊόντος
- category: Η κατηγορία στην οποία ανήκει το προϊόν
- slug: Το slug του προϊόντος
- price: Η τιμή πώλησης του προϊόντος
- active: Εάν το προϊόν είναι ενεργό
- stock: Ο διαθέσιμος αριθμός του προϊόντος
- discount_percent: Η έκπτωση για το προϊόν (%)
- vat: Το ποσοστό του φόρου
- hits: Οι προβολές του προϊόντος
- weight: Το βάρος του προϊόντος
- final_price: Η τελική τιμή του προϊόντος
- discount_value: Η έκπτωση για το προϊόν σε αξία
- price_save_percent: Η εξοικονόμηση τιμής για το προϊόν σε αξία
- translations: Επιτρέπει την αποθήκευση ονόματος (name) και περιγραφής (description) σε πολλαπλές γλώσσες
- search_vector: Χρησιμοποιείται για την εκτέλεση αναζητήσεων σε κείμενα

ProductImage:Το μοντέλο για τις εικόνες του προϊόντος. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα SortableMode και TranslatableModel.

Τα πεδία του ProductImages:

- product: Το προϊόν που αντιστοιχεί η εικόνα
- image: Η εικόνα του προϊόντος
- thumbnail: Η μικρογραφία της εικόνας
- is_main: Εάν είναι η κύρια εικόνα του προϊόντος
- translations: Επιτρέπει την αποθήκευση τίτλου (title) σε πολλαπλές γλώσσες

ProductCategory: Το μοντέλο της κατηγορίας του προϊόντος. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα MPTTModel (django-mptt lib), SeoModel, TranslatableModel και SortableModel.

Τα πεδία του Category:

- slug: Το slug της κατηγορίας
- description: Η περιγραφή της κατηγορίας
- menu_image_one: Η πρώτη εικόνα για το μενού
- menu_image_two: Η δεύτερη εικόνα για το μενού
- menu_main_banner: Η κύρια εικόνα της κατηγορίας
- parent: Η κύρια κατηγορία της κάθε κατηγορίας
- translations: Επιτρέπει την αποθήκευση ονόματος (name) σε πολλαπλές γλώσσες

ProductFavourite: Το μοντέλο του αγαπημένου προϊόντος.

Τα πεδία του Favourite:

- user: Ο χρήστης που αντιστοιχεί
- product: Το προϊόν που αντιστοιχεί

ProductReview: Το μοντέλο της αξιολόγησης. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα PublishableModel και TranslatableModel.

Τα πεδία του Review:

- product: Το προϊόν που αντιστοιχεί
- user: Ο χρήστης που αντιστοιχεί
- rate: Η βαθμολογία της αξιολόγησης
- status: Η κατάσταση που βρίσκεται η αξιολόγηση
- translations: Επιτρέπει την αποθήκευση για το σχόλιο της αξιολόγησης (comment) σε πολλαπλές γλώσσες

Vat: Το μοντέλο του φόρου.

Τα πεδία του Vat:

- value: Η αριθμητική τιμή για το ποσοστό του φόρου

PayWay: Το μοντέλο για τους τρόπους πληρωμής. Χρησιμοποιεί ως βασική κλάση το Abstract μοντέλο SortableModel.

Τα πεδία του PayWay:

- active: Εάν ο τρόπος πληρωμής είναι ενεργός
- cost: Το έξτρα κόστος που θα προστεθεί στην παραγγελία
- free_for_order_amount: Το ελάχιστο ποσό της παραγγελίας χωρίς επιβάρυνση έξτρα κόστους
- icon: Η εικόνα της μεταφορικής
- translations: Επιτρέπει την αποθήκευση ονόματος (name) σε πολλαπλές γλώσσες

Order: Το μοντέλο της παραγγελίας.

Τα πεδία του Order:

- user: Ο εγγεγραμμένος χρήστης που έκανε την παραγγελία
- pay_way: Ο τρόπος πληρωμής της παραγγελίας
- country: Η χώρα που θα σταλεί η παραγγελία
- region: Η περιφέρεια που θα σταλεί η παραγγελία
- floor: Ο όροφος του κτιρίου που θα σταλεί η παραγγελία
- location_type: Ο τύπος τοποθεσίας που θα σταλεί η παραγγελία
- email: Η ηλεκτρονική διεύθυνση του πελάτη
- first_name: Το όνομα του πελάτη
- last_name: Το επώνυμο του πελάτη
- street: Η οδός του πελάτη
- street_number: Ο αριθμός οδού του πελάτη
- city: Η πόλη κατοικίας του πελάτη
- zipcode: Ο ταχυδρομικός κώδικας της περιοχής παράδοσης
- place: Ο τόπος κατοικίας του πελάτη
- phone: Ο αριθμός τηλεφώνου του πελάτη
- mobile_phone: Ο αριθμός κινητού τηλεφώνου του πελάτη
- customer_notes: Οι σημειώσεις του πελάτη
- status: Η κατάσταση της παραγγελίας
- shipping_price: Το κόστος μεταφορικών για την παραγγελία
- document_type: Ο τον τύπο του εγγράφου για την παραγγελία
- paid_amount: Το ποσό πληρωμής του πελάτη

OrderItem: Το μοντέλο ενός προϊόντος της παραγγελίας. Χρησιμοποιεί ως βασική κλάση το Abstract μοντέλο SortableModel.

Τα πεδία του OrderItem:

- order: Η παραγγελία που αντιστοιχεί
- product: Το προϊόν που αντιστοιχεί
- price: Η συνολική τιμή
- quantity: Η ποσότητα του προϊόντος

Slider: Το μοντέλο του Ολισθητή. Χρησιμοποιεί ως βασική κλάση το Abstract μοντέλο TranslatableModel.

Τα πεδία του Slider:

- image: Η εικόνα του Ολισθητή
- thumbnail: Η μικρογραφία του Ολισθητή
- translations: Επιτρέπει την αποθήκευση για το όνομα (name), το url, τον τίτλο (title) και την περιγραφή (description) σε πολλαπλές γλώσσες

Slide: Το μοντέλο μιας Ολίσθησης. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα SortableModel και TranslatableModel

Τα πεδία του Slide:

- slider: Ο Ολισθητής που αντιστοιχεί
- discount: Η έκπτωση της Ολίσθησης
- show_button: Εάν θα εμφανίζεται το κουμπί της Ολίσθησης
- date_start: Η ημερομηνία έναρξης της Ολίσθησης
- date_end: Η ημερομηνία λήξης της Ολίσθησης
- image: Η εικόνα της Ολίσθησης
- thumbnail: Η μικρογραφία της Ολίσθησης
- translations: Επιτρέπει την αποθήκευση για το όνομα (name), το url, τον τίτλο (title), τον υπότιτλο (subtitle), την περιγραφή (description) και το κείμενο για το κουμπί της Ολίσθησης (button label) σε πολλαπλές γλώσσες

UserAccount: Το μοντέλο του λογαριασμού του χρήστη. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα AbstractBaseUser και PermissionsMixin (Django μοντέλα).

Τα πεδία του UserAccount:

- email: Η ηλεκτρονική διεύθυνση του χρήστη
- first_name: Το όνομα του χρήστη
- last_name: Το επώνυμο του χρήστη
- phone: Ο αριθμός τηλεφώνου του χρήστη
- city: Η πόλη κατοικίας του χρήστη
- zipcode: Ο ταχυδρομικός κώδικας κατοικίας του χρήστη
- address: Η διεύθυνση κατοικίας του χρήστη
- place: Ο τόπος κατοικίας του χρήστη
- country: Η χώρα κατοικίας του χρήστη
- region: Η περιοχή κατοικίας του χρήστη
- image: Η εικόνα που έχει επιλέξει ο χρήστης
- is_active: Εάν είναι ενεργός ο λογαριασμός του χρήστη
- is_staff: Εάν ο χρήστης είναι διαχειριστής
- birth_date: Η ημερομηνία γέννησης του χρήστη

Country: Το μοντέλο της χώρας. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα SortableModel και TranslatableModel.

Τα πεδία του Country:

- alpha_2: Ο κωδικός χώρας δύο γραμμάτων που ορίζεται στο ISO 3166-1
- alpha_3: Ο κωδικός χώρας τριών γραμμάτων που ορίζεται στο ISO 3166-1
- iso_cc: Ο κωδικός της χώρας
- phone_code: Ο κωδικός τηλεφώνου της χώρας
- image_flag: Η εικόνα της σημαίας της χώρας
- translations: Επιτρέπει την αποθήκευση ονόματος (name) σε πολλαπλές γλώσσες

Region: Το μοντέλο του τόπου. Χρησιμοποιεί ως βασικές κλάσεις τα Abstract μοντέλα SortableModel και TranslatableModel.

Τα πεδία του Region:

- alpha: Ο κωδικός του τόπου που ορίζεται στο ISO 3166-1
- alpha_2: Ο κωδικός της χώρας δύο γραμμάτων που αντιστοιχεί
- translations: Επιτρέπει την αποθήκευση ονόματος (name) σε πολλαπλές γλώσσες

ΚΕΦΑΛΑΙΟ 3. Αρχιτεκτονική Της Εφαρμογής

3.1 Αναλυτική Περιγραφή

3.1.1 JSON

Το JSON (JavaScript Object Notation) είναι μια ελαφριά μορφή ανταλλαγής δεδομένων που είναι εύκολο να διαβάσουν και να γράψουν οι άνθρωποι και εύκολα να αναλύσουν και να δημιουργήσουν οι μηχανές. Βασίζεται σε ένα υποσύνολο της γλώσσας προγραμματισμού JavaScript, Standard ECMA-262 3rd Edition - Δεκέμβριος 1999. Το JSON είναι μια μορφή κειμένου που είναι εντελώς ανεξάρτητη από τη γλώσσα, αλλά χρησιμοποιεί συμβάσεις που είναι γνωστές στους προγραμματιστές της οικογένειας γλωσσών C, συμπεριλαμβανομένης της C , C++, C#, Java, JavaScript, Perl, Python και πολλά άλλα.

Αυτές οι ιδιότητες καθιστούν την JSON ιδανική γλώσσα ανταλλαγής δεδομένων. Το JSON χρησιμοποιείται κυρίως για τη μετάδοση δεδομένων μεταξύ ενός διακομιστή και μιας εφαρμογής Ιστού, ως εναλλακτική της XML. Μερικές φορές χρησιμοποιείται επίσης για την αποθήκευση απλών δομών δεδομένων και διαμορφώσεων.

Ένα αντικείμενο JSON είναι μια συλλογή ζευγών κλειδιών-τιμών, παρόμοια με ένα λεξικό ή έναν χάρτη σε άλλες γλώσσες προγραμματισμού. Τα κλειδιά είναι πάντα συμβολοσειρές και οι τιμές μπορεί να είναι οποιαδήποτε τιμή JSON (όπως συμβολοσειρά, αριθμός, αντικείμενο, πίνακας, true, false ή null). Ακολουθεί ένα παράδειγμα απλού αντικειμένου JSON:

```
{
  "name": "John Smith",
  "age": 30,
  "city": "New York"
}
```

3.1.2 HTTP

Το HTTP (Hypertext Transfer Protocol) είναι ένα πρωτόκολλο για τη μεταφορά δεδομένων μέσω δικτύου. Το HTTP είναι το πρωτόκολλο που χρησιμοποιείται για την ανάκτηση ιστοσελίδων από το Διαδίκτυο και για την αποστολή δεδομένων σε ιστοσελίδες. Το HTTP λειτουργεί σε περιβάλλον πελάτη-διακομιστή, όπου ο πελάτης (π.χ. ένα πρόγραμμα περιήγησης ιστού) στέλνει αιτήματα στον διακομιστή (π.χ. διακομιστής ιστού) και ο διακομιστής στέλνει απαντήσεις.

Το HTTP χρησιμοποιείται γενικά για την αποστολή και λήψη δεδομένων με τη μορφή HTML, XML και άλλοι τύποι περιεχομένου ιστού. Έχει επίσης τη δυνατότητα να χρησιμοποιείται για αποστολή και λήψη άλλων τύπων δεδομένων, όπως εικόνες, βίντεο και ήχου.

Το HTTP έχει μια απλή και καθολική δομή που επιτρέπει τη χρήση του για μια μεγάλη ποικιλία εφαρμογών. Είναι ένα πρωτόκολλο χωρίς κατάσταση, που σημαίνει ότι δεν παρακολουθεί την κατάσταση μιας σύνδεσης μεταξύ του πελάτη και του διακομιστή. Αυτό επιτρέπει στο HTTP να είναι εξαιρετικά επεκτάσιμο και να μπορεί να χειρίζεται μεγάλο αριθμό αιτημάτων ταυτόχρονα.

3.1.3 Rest API

Το API REST (Representational State Transfer) είναι ένα αρχιτεκτονικό στυλ λογισμικού που ορίζει ένα σύνολο περιορισμών που θα χρησιμοποιηθούν για τη δημιουργία υπηρεσιών Ιστού. Οι υπηρεσίες Ιστού που συμμορφώνονται με το αρχιτεκτονικό στυλ REST, που ονομάζονται υπηρεσίες RESTful Web, παρέχουν διαλειτουργικότητα μεταξύ συστημάτων υπολογιστών στο Διαδίκτυο. Οι υπηρεσίες RESTful Web επιτρέπουν στα αιτούντα συστήματα να έχουν πρόσβαση και να χειρίζονται αναπαραστάσεις κειμένου των πόρων του Ιστού χρησιμοποιώντας ένα ομοιόμορφο και προκαθορισμένο σύνολο λειτουργιών χωρίς κατάσταση. Άλλα είδη υπηρεσιών Ιστού, όπως οι υπηρεσίες Ιστού SOAP, εκθέτουν τα δικά τους αυθαίρετα σύνολα λειτουργιών.

Τα API REST χρησιμοποιούν μεθόδους HTTP για να υποδείξουν την προβλεπόμενη ενέργεια που πρέπει να εκτελεστεί σε έναν συγκεκριμένο πόρο. Οι πιο συνηθισμένες μέθοδοι HTTP είναι οι GET, POST, PUT και DELETE. Ένα αίτημα GET ανακτά μια αναπαράσταση ενός πόρου, ενώ ένα αίτημα POST δημιουργεί έναν νέο πόρο. Ένα αίτημα PUT ενημερώνει έναν υπάρχοντα πόρο και ένα αίτημα DELETE διαγράφει έναν πόρο. Τα API REST έχουν σχεδιαστεί για να είναι απλά και εύχρηστα και συχνά χρησιμοποιούνται για τη δημιουργία σύγχρονων εφαρμογών ιστού και κινητών.

Τα API REST χρησιμοποιούν τυπικούς κωδικούς κατάστασης HTTP για να υποδείξουν την επιτυχία ή την αποτυχία ενός αιτήματος API. Χρησιμοποιούν επίσης κεφαλίδες HTTP και παραμέτρους URL για να παρέχουν πρόσθετες πληροφορίες σχετικά με το αίτημα και την απόκριση.

Τα API REST είναι ευέλικτα και επεκτάσιμα και χρησιμοποιούνται ευρέως στην ανάπτυξη σύγχρονων εφαρμογών ιστού και κινητών. Επιτρέπουν στους προγραμματιστές να δημιουργούν API που μπορούν εύκολα να καταναλωθούν από μια ποικιλία πελατών, συμπεριλαμβανομένων των προγραμμάτων περιήγησης ιστού, των φορητών συσκευών και των εφαρμογών από την πλευρά του διακομιστή.

3.2 Django Rest Framework

Το Django REST Framework είναι μια ισχυρή και ευέλικτη εργαλειοθήκη για τη δημιουργία web API (Application Programming Interfaces) στο πλαίσιο web Django. Παρέχει μια σειρά εργαλείων και λειτουργιών που διευκολύνουν τη δημιουργία API που μπορούν να χειριστούν ένα ευρύ φάσμα απαιτήσεων και περιπτώσεων χρήσης. Μερικά από τα βασικά χαρακτηριστικά του πλαισίου Django REST περιλαμβάνουν:

- **Σειριοποίηση (Serialization)**: Το πλαίσιο Django REST παρέχει ενσωματωμένη υποστήριξη για σειριοποίηση και αποσειριοποίηση δεδομένων, η οποία μάς επιτρέπει

να μετατρέπουμε εύκολα τα μοντέλα Django και τα σύνολα ερωτημάτων μας σε JSON, XML ή άλλες μορφές.

- **Αίτημα ανάλυσης (Analysis request)**: Το πλαίσιο Django REST μπορεί να αναλύσει εισερχόμενα αιτήματα HTTP και να τα μετατρέψει σε αντικείμενα Python, επιτρέποντάς μας να γράψουμε κώδικα που είναι αγνωστικιστικός στη μορφή των εισερχόμενων δεδομένων.

- **Έλεγχος ταυτότητας και άδεια (Authentication and authorization)**: Το πλαίσιο Django REST παρέχει ένα ευρύ φάσμα κλάσεων ελέγχου ταυτότητας και αδειών που διευκολύνουν την ασφάλεια του API και τον έλεγχο της πρόσβασης στα δεδομένα μας.

- **Throttling**: Το πλαίσιο Django REST περιλαμβάνει υποστήριξη για περιορισμό ρυθμών, που μας επιτρέπει να ελέγχουμε πόσα αιτήματα μπορεί να κάνει ένας πελάτης μέσα σε μια συγκεκριμένη χρονική περίοδο.

- **Τεκμηρίωση (Documentation)**: Το πλαίσιο Django REST περιλαμβάνει ενσωματωμένη υποστήριξη για τη δημιουργία τεκμηρίωσης API, η οποία διευκολύνει τους προγραμματιστές να κατανοήσουν πώς να χρησιμοποιούν το API μας.

Συνολικά, το πλαίσιο Django REST είναι μια ισχυρή και ευέλικτη εργαλειοθήκη που διευκολύνει τη δημιουργία ισχυρών και επεκτάσιμων web API στο πλαίσιο ιστού Django. Χρησιμοποιείται ευρέως στην ανάπτυξη API που βασίζονται στον ιστό και είναι μια δημοφιλής επιλογή για προγραμματιστές που θέλουν να δημιουργήσουν API γρήγορα και εύκολα.

3.3 Deployment του GrooveShop

3.3.1 Nginx

Το NGINX είναι ένας δωρεάν, ανοιχτού κώδικα, υψηλής απόδοσης διακομιστής HTTP και λογισμικό αντίστροφου διακομιστή μεσολάβησης. Χρησιμοποιείται ευρέως για τη βελτίωση της απόδοσης και της αξιοπιστίας των εφαρμογών Ιστού, καθώς και για τη μείωση του φόρτου στους διακομιστές Ιστού ενεργώντας ως αντίστροφος διακομιστής μεσολάβησης. Το NGINX είναι γνωστό για την υψηλή απόδοση, τη σταθερότητα και τη χαμηλή κατανάλωση πόρων, καθιστώντας το μια δημοφιλή επιλογή για πολλές εφαρμογές web. Εκτός από τις δυνατότητες του διακομιστή HTTP, το NGINX μπορεί επίσης να χρησιμοποιηθεί ως εξισορρόπηση φορτίου, προσωρινή μνήμη HTTP και ως αντίστροφος διακομιστής για TCP, UDP και άλλους τύπους κίνησης. Είναι εξαιρετικά παραμετροποιήσιμο και μπορεί να επεκταθεί με μονάδες για να προσθέσει πρόσθετη λειτουργικότητα. Το NGINX χρησιμοποιείται από πολλούς ιστότοπους υψηλής επισκεψιμότητας, συμπεριλαμβανομένων των Netflix, Hulu και WordPress.com.

3.3.2 Docker

Το Docker είναι ένα εργαλείο που έχει σχεδιαστεί για να διευκολύνει τη δημιουργία, την ανάπτυξη και την εκτέλεση εφαρμογών χρησιμοποιώντας κοντέινερ. Τα κοντέινερ επιτρέπουν σε έναν προγραμματιστή να συσκευάσει μια εφαρμογή με όλα τα μέρη που χρειάζεται, όπως βιβλιοθήκες και άλλες εξαρτήσεις, και να τα αποστείλει όλα ως ένα πακέτο. Χρησιμοποιώντας κοντέινερ, οι προγραμματιστές μπορούν να είναι σίγουροι ότι η εφαρμογή τους θα εκτελείται σε οποιοδήποτε άλλο μηχάνημα, ανεξάρτητα από τυχόν προσαρμοσμένες ρυθμίσεις που μπορεί να έχει το μηχάνημα, οι οποίες θα μπορούσαν να διαφέρουν από το μηχάνημα που χρησιμοποιείται για τη σύνταξη και τη δοκιμή του κώδικα.

Το Docker επιτρέπει στους χρήστες να διαχειρίζονται και να αναπτύσσουν εφαρμογές με φορητό και συνεπή τρόπο, διευκολύνοντας την ανάπτυξη και την κυκλοφορία λογισμικού. Χρησιμοποιείται από προγραμματιστές και διαχειριστές συστημάτων για τη δημιουργία, αποστολή και εκτέλεση εφαρμογών.

Το Docker παρέχει ένα σύνολο εργαλείων και υπηρεσιών για τη δημιουργία και τη διαχείριση κοντέινερ, συμπεριλαμβανομένου του Docker Engine, το οποίο είναι χρόνος εκτέλεσης για τη δημιουργία και τη λειτουργία κοντέινερ, και το Docker Hub, το οποίο είναι ένα μητρώο που βασίζεται σε σύννεφο για την αποθήκευση και την κοινή χρήση εικόνων κοντέινερ.

3.3.3 GitHub

Το GitHub είναι μια πλατφόρμα φιλοξενίας και συνεργασίας σε έργα λογισμικού. Είναι μια διαδικτυακή πλατφόρμα που χρησιμοποιεί το σύστημα ελέγχου έκδοσης Git για την αποθήκευση και τη διαχείριση αποθετηρίων κώδικα. Το GitHub παρέχει μια σειρά εργαλείων και δυνατοτήτων στους προγραμματιστές για να εργαστούν σε έργα, όπως έλεγχος έκδοσης, διαχείριση έργου, έλεγχος κώδικα και πολλά άλλα.

Το GitHub επιτρέπει στους προγραμματιστές να δημιουργούν και να διαχειρίζονται αποθετήρια για τα έργα τους, καθώς και να συνεργάζονται με άλλους προγραμματιστές σε αυτά τα έργα. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν το GitHub για να παρακολουθούν τις αλλαγές στον κώδικά τους, να συνεργάζονται με άλλους για την ανάπτυξη κώδικα και να μοιράζονται τον κώδικά τους με την ευρύτερη κοινότητα. Το GitHub προσφέρει επίσης μια σειρά εργαλείων για τη διαχείριση έργου, συμπεριλαμβανομένης της δυνατότητας ανάθεσης εργασιών, παρακολούθησης της προόδου και συζήτησης ιδεών με τα μέλη της ομάδας.

Εκτός από τις δυνατότητες συνεργασίας του βασικού κώδικα και ελέγχου εκδόσεων, το GitHub παρέχει επίσης μια σειρά εργαλείων στους προγραμματιστές για να δημιουργήσουν και να αναπτύξουν τις εφαρμογές τους, συμπεριλαμβανομένων ενσωματώσεων με δημοφιλείς πλατφόρμες συνεχούς ενοποίησης και ανάπτυξης. Το GitHub χρησιμοποιείται ευρέως από προγραμματιστές και οργανισμούς σε όλο τον κόσμο και είναι απαραίτητο εργαλείο για πολλά έργα ανάπτυξης λογισμικού.

3.4 Testing και Coverage

3.4.1 Testing

Το Django είναι ένα δημοφιλές πλαίσιο ιστού γραμμένο σε Python που συνοδεύεται από ένα ενσωματωμένο πλαίσιο δοκιμών. Το πλαίσιο δοκιμών Django μάς βοηθά να δοκιμάσουμε τις εφαρμογές Django παρέχοντας μια σουίτα δοκιμών και διάφορα εργαλεία για τη δοκιμή διαφορετικών τμημάτων της εφαρμογής μας.

Υπάρχουν διάφοροι τύποι δοκιμών που μπορούμε να εκτελέσουμε με το πλαίσιο δοκιμών Django:

Δοκιμές μονάδων (Unit tests): Αυτές οι δοκιμές χρησιμοποιούνται για τον έλεγχο μεμονωμένων μονάδων κώδικα, όπως συναρτήσεις ή μεθόδους. Οι δοκιμές μονάδων έχουν σχεδιαστεί για να είναι γρήγορες και μεμονωμένες και δεν πρέπει να εξαρτώνται από εξωτερικούς πόρους ή κατάσταση.

- **Δοκιμές ενσωμάτωσης (Integration testing)**: Αυτές οι δοκιμές χρησιμοποιούνται για να ελέγξουν πώς συνεργάζονται διαφορετικά μέρη της εφαρμογής μας. Οι δοκιμές ενοποίησης μπορεί να περιλαμβάνουν τη δοκιμή πολλαπλών στοιχείων της εφαρμογής μας, όπως η βάση δεδομένων, το σύστημα αρχείων ή εξωτερικά API.

- **Λειτουργικές δοκιμές (Functional tests)**: Αυτές οι δοκιμές προσομοιώνουν τη συμπεριφορά ενός χρήστη που αλληλεπιδρά με την εφαρμογή μας μέσω ενός προγράμματος περιήγησης ιστού. Οι λειτουργικές δοκιμές χρησιμοποιούνται για τη δοκιμή της συνολικής λειτουργικότητας της εφαρμογής μας, συμπεριλαμβανομένου του τρόπου με τον οποίο ανταποκρίνεται στις εισαγωγές των χρηστών και χειρίζεται διαφορετικούς τύπους δεδομένων.

- Για να εκτελέσουμε δοκιμές στο Django, μπορούμε να χρησιμοποιήσουμε την εντολή δοκιμής ‘python manage.py test’, η οποία θα ανακαλύψει και θα εκτελέσει όλες τις δοκιμές στο έργο Django μας. Μπορούμε επίσης να καθορίσουμε το όνομα μιας συγκεκριμένης εφαρμογής ή μιας δοκιμαστικής περίπτωσης για να εκτελέσουμε μόνο ένα υποσύνολο των δοκιμών μας.

3.4.2 Coverage

Η κάλυψη Django είναι ένα εργαλείο που μπορεί να χρησιμοποιηθεί για τη μέτρηση της δοκιμαστικής κάλυψης της εφαρμογής Django. Αναλύει τη βάση κωδίκων μας και αναφέρει ποιες γραμμές κώδικα εκτελούνται κατά την εκτέλεση των δοκιμών και ποιες γραμμές δεν καλύπτονται από τις δοκιμές. Αυτό μπορεί να είναι χρήσιμο για τον εντοπισμό περιοχών του κώδικά μας που ενδέχεται να μην έχουν ελεγχθεί διεξοδικά και για τη διασφάλιση ότι οι δοκιμές μας παρέχουν ολοκληρωμένη κάλυψη του κώδικά μας.

Για να χρησιμοποιήσουμε την κάλυψη Django, θα χρειαστεί να εγκαταστήσουμε το πακέτο κάλυψης και να το προσθέσουμε στις εξαρτήσεις του έργου Django. Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε το εργαλείο γραμμής εντολών κάλυψης για να εκτελέσουμε

τις δοκιμές μας και να δημιουργήσουμε μια αναφορά σχετικά με την κάλυψη δοκιμής. Μπορούμε επίσης να χρησιμοποιήσουμε τη μονάδα κάλυψης στον κώδικα Python μας για να μετρήσουμε προγραμματικά τη δοκιμαστική κάλυψη συγκεκριμένων τμημάτων της εφαρμογής μας.

Ακολουθεί ένα παράδειγμα του τρόπου χρήσης του εργαλείου γραμμής εντολών κάλυψης για τη μέτρηση της δοκιμαστικής κάλυψης ενός έργου Django:

```
$ coverage run --source='.' manage.py test
$ coverage report
```

Αυτό θα εκτελέσει τις δοκιμές μας στο Django και θα δημιουργήσει μια αναφορά σχετικά με την κάλυψη της δοκιμής. Η επιλογή --source καθορίζει τους καταλόγους που πρέπει να περιλαμβάνονται στην αναφορά κάλυψης και η εντολή δοκιμής manager.py εκτελεί τις δοκιμές Django. Η εντολή αναφοράς κάλυψης δημιουργεί μια αναφορά που παραθέτει τις γραμμές κώδικα που καλύπτονται από τις δοκιμές, καθώς και αυτές που δεν καλύπτονται.

Μπορούμε επίσης να χρησιμοποιήσουμε τη μονάδα κάλυψης στον κώδικα Python για να μετρήσουμε προγραμματικά τη δοκιμαστική κάλυψη συγκεκριμένων τμημάτων της εφαρμογής μας.

Για παράδειγμα:

```
import coverage

coverage.start()
# Run some code here
coverage.stop()
coverage.report()
```

Αυτό θα ξεκινήσει μια μέτρηση κάλυψης, θα εκτελέσει κάποιο κωδικό, θα σταματήσει τη μέτρηση και θα δημιουργήσει μια αναφορά σχετικά με τη δοκιμαστική κάλυψη.

3.5 Εργαλεία που χρησιμοποιήθηκαν

Η ανάπτυξη μιας σύγχρονης ιστοσελίδας απαιτεί τη χρήση ενός ευρίσκοντος εργαλείων και τεχνολογιών για να δημιουργήσετε μια εφαρμογή που είναι αξιόπιστη, αποδοτική και φιλική προς τον χρήστη. Στην περίπτωσή μας, έχουμε χρησιμοποιήσει μια ποικιλία από εργαλεία και τεχνολογίες που συνεργάζονται αρμονικά για τη δημιουργία της εφαρμογής μας. Παρακάτω, θα προσφέρουμε μια επισκόπηση των κύριων εργαλείων που χρησιμοποιήθηκαν στον κύκλο ανάπτυξης της εφαρμογής μας:

3.5.1 Django

Το Django είναι ένα Framework Python υψηλού επιπέδου που ενθαρρύνει την ταχεία ανάπτυξη και τον καθαρό, ρεαλιστικό σχεδιασμό. Κατασκευασμένο από έμπειρους προγραμματιστές, φροντίζει για μεγάλο μέρος της ταλαιπωρίας της ανάπτυξης ιστού, ώστε

να μπορούμε να εστιάσουμε στη σύνταξη της εφαρμογής μας χωρίς να χρειάζεται να ανακαλύψουμε ξανά τον τροχό. Είναι δωρεάν και ανοιχτού κώδικα.

Μερικά από τα κύρια χαρακτηριστικά του Django είναι:

- Μια στιβαρή και ευέλικτη αρχιτεκτονική MVC (μοντέλο-προβολή-πρότυπο).
- Ένα πανίσχυρο ORM (αντικειμενο-σχεσιακός χάρτης) που υποστηρίζει ένα ευρύ φάσμα backend της βάσης δεδομένων.
- Μια ενσωματωμένη διαχειριστική διεπαφή για τη διαχείριση της εφαρμογής μας Ενσωματωμένη υποστήριξη για έλεγχο ταυτότητας χρήστη, δικαιώματα και εξουσιοδότηση.
- Εκτεταμένη τεκμηρίωση και μια μεγάλη και ενεργή κοινότητα.

Το Django είναι κατάλληλο για την ανάπτυξη όλων των ειδών διαδικτυακών εφαρμογών, από μικρά προσωπικά ιστολόγια έως εφαρμογές μεγάλων επιχειρήσεων. Είναι γρήγορο, ευέλικτο και ασφαλές και μπορεί να χειριστεί υψηλά φορτία κυκλοφορίας χωρίς να ιδρώσει. Αν θέλουμε να δημιουργήσουμε μια εφαρμογή web σε Python, το Django είναι μια εξαιρετική επιλογή.

3.5.2 Nest.js

Το Nest.js είναι ένα προοδευτικό πλαίσιο για τη δημιουργία αποτελεσματικών, επεκτάσιμων εφαρμογών Node.js από την πλευρά του διακομιστή. Χρησιμοποιεί TypeScript, ένα αυστηρό συντακτικό υπερσύνολο της JavaScript, ως κύρια γλώσσα προγραμματισμού. Το Nest.js είναι χτισμένο πάνω από το δημοφιλές πλαίσιο Node.js Express και χρησιμοποιεί επίσης άλλες δημοφιλείς βιβλιοθήκες όπως το Fastify, το Socket.io και το MongoDB.

Μερικά από τα κύρια χαρακτηριστικά του Nest.js είναι:

- Μια αρθρωτή αρχιτεκτονική που διευκολύνει τη δημιουργία επαναχρησιμοποιούμενου και συντηρήσιμου κώδικα.
- Διακοσμητές, που είναι ένα μοναδικό χαρακτηριστικό του TypeScript, που διευκολύνουν τον ορισμό και τη χρήση ελεγκτών, υπηρεσιών και άλλων στοιχείων Nest.js.
- Ένα ισχυρό σύστημα έγχυσης εξάρτησης (DI) που βοηθά στη διαχείριση των σχέσεων μεταξύ διαφορετικών στοιχείων στην εφαρμογή μας.
- Μια ενοποίηση με την GraphQL, μια ευέλικτη γλώσσα ερωτημάτων δεδομένων που μας επιτρέπει να ζητάτε ακριβώς τα δεδομένα που χρειάζεστε και τίποτα περισσότερο.
- Μια ενσωματωμένη μονάδα δοκιμών που διευκολύνει τη σύνταξη και την εκτέλεση δοκιμών μονάδας και ολοκλήρωσης.

- Το Nest.js είναι μια καλή επιλογή για τη δημιουργία επεκτάσιμων, εταιρικού επιπέδου εφαρμογών Node.js. Η αρθρωτή αρχιτεκτονική του και η χρήση του TypeScript καθιστούν εύκολη τη σύνταξη καθαρού, συντηρήσιμου κώδικα και η ενσωμάτωσή του με δημοφιλείς βιβλιοθήκες όπως η Express και η MongoDB διευκολύνουν τη δημιουργία ισχυρών και πλούσιων σε δυνατότητες εφαρμογών.

3.5.3 Vue.js

Το Vue.js είναι ένα προοδευτικό πλαίσιο JavaScript για τη δημιουργία διεπαφών χρήστη. Έχει σχεδιαστεί για να είναι ελαφρύ, εύκολο στην εκμάθηση και εύκολο να ενσωματωθεί με άλλες βιβλιοθήκες ή υπάρχοντα έργα.

Ένα από τα βασικά χαρακτηριστικά του Vue.js είναι τα αντιδραστικά στοιχεία του, τα οποία μας επιτρέπουν να αποδίδουμε δηλωτικά δυναμικά πρότυπα που ενημερώνονται αυτόματα όταν αλλάζουν τα υποκείμενα δεδομένα. Το Vue.js παρέχει επίσης έναν απλό αλλά ισχυρό τρόπο ορισμού και διαχείρισης στοιχείων, καθιστώντας εύκολη τη δημιουργία και τη συντήρηση μεγάλων και πολύπλοκων εφαρμογών.

Ακολουθούν μερικές από τις βασικές έννοιες που πρέπει να γνωρίζουμε για το Vue.js:

- **Σύνταξη προτύπου (Template syntax)**: Το Vue.js χρησιμοποιεί μια δηλωτική σύνταξη προτύπου που μας επιτρέπει να ορίσουμε τη διεπαφή χρήστη της εφαρμογής μας χρησιμοποιώντας πρότυπα που μοιάζουν με HTML. Αυτά τα πρότυπα μεταγλωτίζονται σε εικονικούς κόμβους DOM (Document Object Model) που ενημερώνονται αποτελεσματικά κάθε φορά που αλλάζουν τα υποκείμενα δεδομένα.

- **Αντιδραστικά στοιχεία (Reactive components)**: Στο Vue.js, τα στοιχεία ορίζονται ως ένας συνδυασμός προτύπου, δεδομένων και μεθόδων. Τα δεδομένα γίνονται αντιδραστικά χρησιμοποιώντας ένα ειδικό αντικείμενο Vue.js που ονομάζεται "στιγμιότυπο Vue", το οποίο παρακολουθεί τις αλλαγές στα δεδομένα και ενημερώνει αυτόματα τη διεπαφή χρήστη.

- **Στοιχεία ενός αρχείου (Single-file components)**: Το Vue.js μάς επιτρέπει να ορίσουμε τα στοιχεία μας σε ένα μόνο αρχείο, χρησιμοποιώντας έναν συνδυασμό HTML, JavaScript και CSS. Αυτό διευκολύνει την οργάνωση και τη διατήρηση του κώδικά μας και μας επιτρέπει επίσης να χρησιμοποιούμε προηγμένους προεπεξεργαστές όπως Sass ή Less.

- **Υπολογιζόμενες ιδιότητες (Computed properties)**: Το Vue.js παρέχει έναν τρόπο ορισμού υπολογισμένων ιδιοτήτων, οι οποίες είναι συναρτήσεις που αποθηκεύονται στην κρυφή μνήμη με βάση τις εξαρτήσεις τους και επαναξιολογούνται μόνο όταν αλλάξουν αυτές οι εξαρτήσεις. Αυτό μας επιτρέπει να ορίσουμε σύνθετα, προερχόμενα δεδομένα που ενημερώνονται αυτόματα.

- **Οδηγίες (Directives)**: Το Vue.js παρέχει ένα σύνολο οδηγιών, οι οποίες είναι ειδικά χαρακτηριστικά HTML που ξεκινούν με το πρόθεμα "v-". Αυτές οι οδηγίες μάς επιτρέπουν να συνδέουμε δεδομένα με τα χαρακτηριστικά ενός στοιχείου ή να συνδέουμε εκφράσεις στο περιεχόμενο του στοιχείου.

- **Προσθήκες (Plugins)**: Το Vue.js διαθέτει ένα σύστημα προσθηκών που μας επιτρέπει να επεκτείνουμε τη βασική του λειτουργικότητα με πρόσθετες λειτουργίες. Υπάρχουν πολλές διαθέσιμες προσθήκες τρίτων που παρέχουν τα πάντα, από δρομολόγηση και διαχείριση κατάστασης έως επικύρωση και διεθνοποίηση φόρμας.

3.5.4 Nuxt 3

Το Nuxt 3 είναι η τελευταία έκδοση του δημοφιλούς πλαισίου Vue.js για την κατασκευή εφαρμογών ιστού, ιστότοπων και στατικών τοποθεσιών. Πρόκειται για μια σημαντική αναβάθμιση που φέρνει σημαντικές βελτιώσεις και νέα χαρακτηριστικά στο πλαίσιο, μεταξύ των οποίων:

- **Αρθρωτή αρχιτεκτονική**: Το Nuxt 3 εισάγει μια νέα αρθρωτή αρχιτεκτονική που επιτρέπει στους προγραμματιστές να χρησιμοποιούν μόνο τα μέρη του πλαισίου που χρειάζονται, χωρίς να δεσμεύονται από μια συγκεκριμένη δομή έργου ή διαμόρφωση. Αυτό διευκολύνει την κατασκευή και τη συντήρηση σύνθετων εφαρμογών και ιστότοπων.

- **Βελτιωμένη απόδοση**: Το Nuxt 3 διαθέτει ένα νέο σύστημα δημιουργίας που βελτιστοποιεί τις επιδόσεις μειώνοντας το μέγεθος της τελικής εξόδου, βελτιώνοντας την προσωρινή αποθήκευση και μειώνοντας τους χρόνους φόρτωσης. Υποστηρίζει επίσης την αντικατάσταση εν θερμώ μονάδων για ταχύτερη ανάπτυξη και αποσφαλμάτωση.

- **Απλούστερη διαμόρφωση**: Το Nuxt 3 έχει απλοποιήσει τη διαμόρφωση και εισήγαγε μια νέα μορφή αρχείου διαμόρφωσης που διευκολύνει τη διαχείριση και την κατανόηση των ρυθμίσεων του έργου. Το νέο σύστημα διαμόρφωσης καθιστά επίσης δυνατή τη διαμόρφωση του πλαισίου χρησιμοποιώντας μεταβλητές περιβάλλοντος.

- **Βελτιωμένη υποστήριξη TypeScript**: Το Nuxt 3 έχει βελτιωμένη υποστήριξη για TypeScript, καθιστώντας ευκολότερη τη συγγραφή κώδικα με ασφάλεια τύπου και την ενσωμάτωση με βιβλιοθήκες και εργαλεία που βασίζονται σε TypeScript.

- **Δημιουργία στατικού ιστότοπου**: Το Nuxt 3 διαθέτει μια νέα λειτουργία δημιουργίας στατικού ιστότοπου που επιτρέπει στους προγραμματιστές να δημιουργούν στατικά αρχεία HTML για τον ιστότοπο ή την εφαρμογή τους. Αυτό καθιστά δυνατή τη δημιουργία γρήγορων, κλιμακούμενων και ασφαλών ιστότοπων χωρίς να απαιτείται διακομιστής ή βάση δεδομένων.

- **Βελτιωμένη τεκμηρίωση**: Το Nuxt 3 διαθέτει βελτιωμένη τεκμηρίωση που διευκολύνει τους προγραμματιστές να μάθουν το πλαίσιο, να βρουν απαντήσεις στις ερωτήσεις τους και να συνεισφέρουν στην κοινότητα.

Συνολικά, το Nuxt 3 είναι μια σημαντική ενημέρωση που φέρνει σημαντικές βελτιώσεις και νέα χαρακτηριστικά στο πλαίσιο Nuxt.js. Η αρθρωτή αρχιτεκτονική του, η βελτιωμένη απόδοση, η απλούστερη διαμόρφωση και η βελτιωμένη υποστήριξη TypeScript διευκολύνουν την κατασκευή και τη συντήρηση σύνθετων εφαρμογών και ιστότοπων, ενώ η

νέα λειτουργία δημιουργίας στατικών ιστότοπων καθιστά δυνατή τη δημιουργία γρήγορων, κλιμακούμενων και ασφαλών ιστότοπων χωρίς να απαιτείται διακομιστής ή βάση δεδομένων.

3.5.5 Postgresql

Το PostgreSQL, επίσης γνωστό ως Postgres, είναι ένα δωρεάν και ανοιχτού κώδικα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων που δίνει έμφαση στην επεκτασιμότητα και τη συμμόρφωση με την SQL. Χρησιμοποιείται ως κύριος χώρος αποθήκευσης δεδομένων ή αποθήκη δεδομένων για πολλές εφαρμογές ιστού, κινητές συσκευές, γεωχωρικές, αναλυτικές και άλλες εφαρμογές.

Μερικά από τα βασικά χαρακτηριστικά της PostgreSQL είναι:

- Υποστήριξη πολλαπλών τύπων δεδομένων, συμπεριλαμβανομένων πρωτόγονων τύπων όπως ακέραιοι αριθμοί και συμβολοσειρές, καθώς και πιο πολύπλοκοι τύποι όπως πίνακες, JSON και XML.

- Προηγμένη ευρετηρίαση και βελτιστοποίηση ερωτημάτων, συμπεριλαμβανομένης της υποστήριξης για αναζήτηση πλήρους κειμένου και χωρική ευρετηρίαση.

- Υποστήριξη για ενεργοποιητές, αποθηκευμένες διαδικασίες και προβολές, που μας επιτρέπει να ορίσουμε προσαρμοσμένη λογική και αφαιρέσεις δεδομένων.

- Υποστήριξη για συναλλαγές και έλεγχο συγχρονισμού πολλαπλών εκδόσεων (MVCC), που μας επιτρέπει να διασφαλίσουμε την ακεραιότητα και τη συνέπεια των δεδομένων.

- Υποστήριξη για αναπαραγωγή και υψηλή διαθεσιμότητα, που μας επιτρέπει να κλιμακώσουμε τη βάση δεδομένων μας σε πολλούς διακομιστές και να εξασφαλίσουμε χρόνο λειτουργίας.

Το PostgreSQL είναι ένα ισχυρό και ευέλικτο σύστημα βάσης δεδομένων που είναι κατάλληλο για ένα ευρύ φάσμα εφαρμογών. Είναι γνωστό για τη σταθερότητα, την απόδοση και το πλούσιο σύνολο χαρακτηριστικών του και χρησιμοποιείται από πολλούς οργανισμούς υψηλού προφίλ σε όλο τον κόσμο. Εάν χρειάζεστε ένα ισχυρό και αξιόπιστο σύστημα βάσης δεδομένων για την εφαρμογή μας, η PostgreSQL είναι μια εξαιρετική επιλογή.

3.5.5 Vite

Το Vite (προφέρεται "veet") είναι ένα εργαλείο κατασκευής για σύγχρονη ανάπτυξη ιστού που στοχεύει να είναι γρήγορο και ελαφρύ. Έχει σχεδιαστεί για να λειτουργεί με το πλαίσιο JavaScript Vue.js, αλλά μπορεί επίσης να χρησιμοποιηθεί με άλλα πλαίσια και βιβλιοθήκες.

Ένα από τα κύρια χαρακτηριστικά του Vite είναι η ικανότητά του να εκτελεί "hot module αντικατάσταση" (HMR), το οποίο μας επιτρέπει να κάνουμε αλλαγές στον κώδικά μας και να βλέπουμε τις ενημερώσεις στο πρόγραμμα περιήγησης σε πραγματικό χρόνο,

χωρίς να χρειάζεται να ανανεώσουμε τη σελίδα με μη αυτόματο τρόπο. Αυτό μπορεί να επιταχύνει σημαντικά τη διαδικασία ανάπτυξης και να διευκολύνει την πραγματοποίηση αλλαγών και την επανάληψη των έργων μας.

Ένα άλλο βασικό χαρακτηριστικό του Vite είναι η ικανότητά του να δημιουργεί αποτελεσματικά για την παραγωγή. Χρησιμοποιεί σύγχρονες λειτουργίες JavaScript και βελτιστοποιήσεις για τη δημιουργία μικρών, γρήγορων δεσμίδων που είναι εύκολο να αναπτυχθούν.

Το Vite είναι χτισμένο πάνω από το σύστημα εγγενούς λειτουργικής μονάδας ES (ESM), το οποίο του επιτρέπει να εκμεταλλεύεται τις πιο πρόσφατες δυνατότητες JavaScript και να εκτελεί γρήγορες, αποτελεσματικές κατασκευές. Χρησιμοποιεί επίσης ένα απλό αρχείο διαμόρφωσης, που ονομάζεται "vite.config.js", για να μας επιτρέψει να προσαρμόσουμε τη διαδικασία κατασκευής και να ενσωματώσουμε πρόσθετα εργαλεία και πρόσθετα όπως απαιτείται.

Συνολικά, το Vite είναι ένα ισχυρό εργαλείο που μπορεί να μας βοηθήσει να δημιουργήσουμε γρήγορα και να επαναλάβουμε τις σύγχρονες εφαρμογές ιστού χρησιμοποιώντας το Vue.js ή άλλα πλαίσια. Είναι εύκολο στην εκμάθηση και τη χρήση, και μπορεί να μας βοηθήσει να επωφεληθούμε από τις πιο πρόσφατες δυνατότητες και βελτιστοποιήσεις JavaScript για να δημιουργήσουμε γρήγορες, αποτελεσματικές και επεκτάσιμες εφαρμογές web.

3.5.6 GitHub actions

Το GitHub Actions είναι μια πλατφόρμα συνεχούς ενοποίησης και παράδοσης (CI/CD) που μας βοηθά να αυτοματοποιήσουμε τις ροές εργασιών ανάπτυξης λογισμικού μας. Με το GitHub Actions, μπορούμε να ορίσουμε προσαρμοσμένες διαμορφώσεις ροής εργασίας στο αποθετήριό μας για τη δημιουργία, τη δοκιμή, τη συσκευασία, την κυκλοφορία ή την ανάπτυξη του κώδικά μας. Μπορούμε επίσης να χρησιμοποιήσουμε προκατασκευασμένες ενέργειες που έχουν δημιουργηθεί από την κοινότητα του GitHub ή να δημιουργήσουμε τις δικές μας ενέργειες που ταιριάζουν στις συγκεκριμένες ανάγκες μας.

Μερικά από τα βασικά χαρακτηριστικά του GitHub Actions περιλαμβάνουν:

- **Προσαρμογή ροής εργασιών (Workflow customization)**: Μπορούμε να ορίσουμε προσαρμοσμένες ροές εργασίας στο αποθετήριο μας χρησιμοποιώντας αρχεία YAML και να χρησιμοποιήσουμε ένα ευρύ φάσμα προκατασκευασμένων ενεργειών ή να δημιουργήσουμε τις δικές μας ενέργειες για να ταιριάζουν στις συγκεκριμένες ανάγκες μας.

- **Ενσωματωμένη υποστήριξη για δημοφιλείς γλώσσες και τεχνολογίες**: Το GitHub Actions διαθέτει ενσωματωμένη υποστήριξη για δημοφιλείς γλώσσες και τεχνολογίες όπως JavaScript, Python, Ruby, Java, .NET και άλλα.

- **Ενσωμάτωση με το GitHub**: Το GitHub Actions είναι στενά ενσωματωμένο με το GitHub, ώστε να μπορούμε να ενεργοποιούμε ενέργειες που βασίζονται σε συμβάντα στο χώρο αποθήκευσης μας, όπως όταν προωθείται μια νέα δέσμευση ή ανοίγει ένα αίτημα έλξης.

- **Επεκτασιμότητα και αξιοπιστία**: Το GitHub Actions είναι χτισμένο πάνω σε μια εξαιρετικά επεκτάσιμη και αξιόπιστη υποδομή, ώστε να μπορούμε να την εμπιστεύεστε για την ομαλή και συνεπή εκτέλεση των ροών εργασίας μας.

Το GitHub Actions είναι ένα ισχυρό εργαλείο για την αυτοματοποίηση των ροών εργασιών ανάπτυξης λογισμικού μας και μπορεί να μας βοηθήσει να εξοικονομήσουμε χρόνο και προσπάθεια αυτοματοποιώντας επαναλαμβανόμενες εργασίες και απελευθερώνοντας την ομάδα μας να επικεντρωθεί σε πιο σημαντικές εργασίες. Εάν θέλουμε να απλοποιήσουμε τη διαδικασία CI/CD και να κάνουμε τη διαδικασία ανάπτυξής μας πιο αποτελεσματική, το GitHub Actions είναι μια εξαιρετική επιλογή.

3.5.7 SCSS

Η SCSS (συντομογραφία του "Sassy CSS") είναι μια γλώσσα προεπεξεργαστή για CSS που μας επιτρέπει να χρησιμοποιούμε μια ποικιλία προηγμένων τεχνικών για τη σύνταξη και τη συντήρηση των φύλλων στυλ μας. Βασίζεται στη σύνταξη του CSS, αλλά προσθέτει μια σειρά από πρόσθετες λειτουργίες που διευκολύνουν τη σύνταξη και τη διατήρηση μεγάλων και πολύπλοκων φύλλων στυλ.

Εδώ είναι μερικά από τα βασικά χαρακτηριστικά του SCSS:

- **Μεταβλητές**: Το SCSS μάς επιτρέπει να ορίσουμε μεταβλητές που μπορούν να χρησιμοποιηθούν σε όλο το φύλλο στυλ μας. Αυτό διευκολύνει την επαναχρησιμοποίηση τιμών, όπως τα χρώματα, τα μεγέθη γραμματοσειρών και τις κλίμακες διαστήματος, και βοηθά στο να διατηρούμε τα φύλλα στυλ μας DRY (Don't Repeat Yourself).

- **Ένθεση**: Το SCSS μάς επιτρέπει να τοποθετούμε επιλογές ο ένας μέσα στον άλλο, κάτι που μπορεί να βοηθήσει στη μείωση του πλεονασμού και να κάνει τα φύλλα στυλ μας πιο ευανάγνωστα. Για παράδειγμα, μπορούμε να τοποθετήσουμε μια κλάση μέσα σε ένα στοιχείο για να εφαρμόσουμε στυλ μόνο σε αυτό το στοιχείο όταν έχει την κλάση:

```
.my-element {
  color: red;
  .my-class {
    font-size: 20px;
  }
}
```

- **Mixins**: Το SCSS μάς επιτρέπει να ορίσουμε mixins, τα οποία είναι επαναχρησιμοποιήσιμα μπλοκ στυλ που μπορούν να συμπεριληφθούν σε πολλά μέρη. Τα Mixins μπορούν να δεχτούν επιχειρήματα, καθιστώντας εύκολη την προσαρμογή της συμπεριφοράς τους.

- **Λειτουργίες (Functions)**: Το SCSS περιλαμβάνει έναν αριθμό ενσωματωμένων συναρτήσεων που μας επιτρέπουν να εκτελούμε υπολογισμούς και να χειρίζεστε τιμές. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση φωτισμού για

να προσαρμόσουμε τη φωτεινότητα ενός χρώματος ή τη συνάρτηση % για να μετατρέψουμε μια τιμή σε ποσοστό.

- **Εισαγωγές (Imports)**: Το SCSS μάς επιτρέπει να εισάγουμε φύλλα στυλ από άλλα αρχεία, τα οποία μπορεί να είναι χρήσιμα για την οργάνωση μεγάλων φύλλων στυλ ή την κοινή χρήση κοινών στυλ μεταξύ έργων.

Το SCSS είναι ένα ισχυρό εργαλείο που μπορεί να μας βοηθήσει να γράφουμε και να διατηρούμε αποτελεσματικά και διατηρούμενα φύλλα στυλ. Χρησιμοποιείται ευρέως στην κοινότητα ανάπτυξης web front-end και υποστηρίζεται από πολλά δημοφιλή εργαλεία και συστήματα κατασκευής.

3.5.8 Linter

Ο linter είναι ένα εργαλείο που αναλύει τον πηγαίο κώδικα για να επισημάνει σφάλματα προγραμματισμού, σφάλματα, στιλιστικά λάθη και ύποπτες κατασκευές. Ο όρος "linter" προήλθε ως ένα portmanteau του "lint", που είναι ένας όρος για μικρά κομμάτια βαμβακιού ή άλλων υπολειμμάτων που αφήνονται σε ένα νήμα ή ύφασμα μετά την επεξεργασία, και "interpreter". Ακριβώς όπως το χνούδι μπορεί να συσσωρευτεί σε φυσικά υλικά και να παρεμποδίσει τη σωστή λειτουργία τους, ο κώδικας "χνούδι" μπορεί να συσσωρευτεί στο λογισμικό και να προκαλέσει προβλήματα.

Οι Linters χρησιμοποιούνται συνήθως για τον έλεγχο του πηγαίου κώδικα γραμμένου σε γλώσσες προγραμματισμού όπως C, C++, C#, Java, JavaScript, PHP, Python και Ruby. Μπορούν να αναλύσουν τον κώδικα για ένα ευρύ φάσμα ζητημάτων, συμπεριλαμβανομένων των συντακτικών σφαλμάτων, των συμβάσεων ονομασίας, των προβλημάτων μορφοποίησης και των τρωτών σημείων ασφαλείας. Ορισμένα γραμμάρια μπορούν επίσης να επιβάλουν ένα συγκεκριμένο στυλ κωδικοποίησης ή βέλτιστες πρακτικές, διευκολύνοντας τις ομάδες προγραμματιστών να διατηρήσουν σταθερή ποιότητα κώδικα.

Οι Linters μπορούν να εκτελεστούν χειροκίνητα ή να ενσωματωθούν σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) ή σε μια διαδικασία κατασκευής. Μπορούν να ρυθμιστούν ώστε να ελέγχουν συγκεκριμένα μέρη του κώδικα ή να αγνοούν ορισμένους τύπους ζητημάτων. Ορισμένες ράβδους μπορούν επίσης να ρυθμιστούν ώστε να διορθώνουν αυτόματα ορισμένους τύπους προβλημάτων, όπως προβλήματα μορφοποίησης.

Οι Linters μπορούν να είναι χρήσιμα για τον εντοπισμό και τη διόρθωση προβλημάτων στον κώδικα πριν από την ανάπτυξή του, καθώς μπορούν να εντοπίσουν ζητήματα που μπορεί να μην είναι άμεσα εμφανή και μπορούν να βοηθήσουν στη βελτίωση της συνολικής ποιότητας και αξιοπιστίας του κώδικα. Ωστόσο, είναι σημαντικό να σημειωθεί ότι τα linters δεν υποκαθιστούν τον ενδελεχή έλεγχο και τον εντοπισμό σφαλμάτων, καθώς ενδέχεται να μην εντοπίσουν όλα τα είδη σφαλμάτων ή ζητημάτων.

3.5.8.1 Typescript

Υπάρχουν πολλές δημοφιλείς γραμμές TypeScript διαθέσιμες, όπως:

- **TSLint**: Αυτή είναι η πιο ευρέως χρησιμοποιούμενη ράβδος TypeScript. Διατηρείται ενεργά και έχει ένα μεγάλο σύνολο κανόνων που μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε για να προσαρμόσουμε τη διαδικασία του λιπαντικού.

- **ESLint**: Αυτό είναι ένα δημοφιλές linter JavaScript που υποστηρίζει επίσης TypeScript. Έχει ένα μεγάλο σύνολο κανόνων και είναι εξαιρετικά παραμετροποιήσιμο.

- **TypeScript-ESLint**: Αυτή είναι μια προσθήκη για το ESLint που προσθέτει υποστήριξη για TypeScript. Μας επιτρέπει να χρησιμοποιούμε τους κανόνες και τις επιλογές διαμόρφωσης του ESLint με τον κώδικα TypeScript.

Για να χρησιμοποιήσουμε μια γραμμή TypeScript, θα χρειαστεί να την εγκαταστήσουμε και να την διαμορφώσουμε στο έργο μας. Αυτό συνήθως περιλαμβάνει τη δημιουργία ενός αρχείου διαμόρφωσης (όπως tslint.json ή .eslintrc.json) που καθορίζει τους κανόνες και τις ρυθμίσεις για το linter. Στη συνέχεια, θα εκτελέσουμε το linter ως μέρος της διαδικασίας κατασκευής μας ή ως ένα άγκιστρο προ-δέσμευσης για να εντοπίσουμε τυχόν προβλήματα πριν δεσμεύσουμε τον κώδικά μας.

Αξίζει να σημειωθεί ότι το ίδιο το TypeScript περιλαμβάνει επίσης έναν ενσωματωμένο έλεγχο τύπων που μπορεί να εντοπίσει σφάλματα τύπου στον κώδικά μας. Ωστόσο, αυτό δεν είναι το ίδιο με ένα γρανάζι, και ένα linter μπορεί να εντοπίσει πρόσθετα ζητήματα, όπως σφάλματα στυλ και σύνταξης που δεν τα κάνει ο ελεγκτής τύπων.

3.5.8.2 Python

Υπάρχουν πολλά διαθέσιμα Python linters και μπορούν να είναι χρήσιμα για μια ποικιλία εργασιών, όπως:

- **Εύρεση συντακτικών σφαλμάτων**: Ένα linter μπορεί να μας βοηθήσει να βρούμε συντακτικά σφάλματα στον κώδικά μας, όπως αγκύλες που λείπουν, μη κλειστά εισαγωγικά ή μη έγκυρους χαρακτήρες. Αυτά τα σφάλματα μπορεί να προκαλέσουν την αποτυχία εκτέλεσης του κώδικά μας ή την παραγωγή μη αναμενόμενων αποτελεσμάτων.

- **Επιβολή κατευθυντήριων γραμμών στυλ**: Πολλά linters μας επιτρέπουν να καθορίσουμε ένα σύνολο οδηγιών στυλ, όπως τον οδηγό στυλ PEP 8 για Python. Ένα linter μπορεί στη συνέχεια να ελέγξει τον κώδικά μας για ζητήματα όπως λανθασμένη εσοχή, περιττό κενό διάστημα ή χρήση απαγορευμένων ονομάτων συναρτήσεων.

- **Προσδιορισμός πιθανών προβλημάτων**: Ορισμένες γραμμές μπορούν επίσης να εντοπίσουν πιθανά προβλήματα στον κώδικά μας, όπως μεταβλητές που δεν χρησιμοποιούνται ή τη χρήση συναρτήσεων που έχουν καταργηθεί. Αυτό μπορεί να μας βοηθήσει να βρούμε και να διορθώσουμε προβλήματα προτού γίνουν πιο σοβαρά προβλήματα.

- **Κώδικας μορφοποίησης**: Ορισμένοι linters περιλαμβάνουν εργαλεία μορφοποίησης που μπορούν να διαμορφώσουν αυτόματα τον κώδικά μας για να ανταποκρίνονται σε έναν συγκεκριμένο οδηγό στυλ. Αυτό μπορεί να είναι χρήσιμο για να διασφαλίσουμε ότι ο κώδικάς μας είναι σταθερά μορφοποιημένος και ευανάγνωστος.

Για να χρησιμοποιήσουμε ένα linter Python, θα πρέπει συνήθως να εγκαταστήσουμε το linter και τυχόν απαιτούμενες εξαρτήσεις και στη συνέχεια να εκτελέσουμε το linter στον κώδικά μας. Πολλά linters έχουν διεπαφές γραμμής εντολών που μας επιτρέπουν να καθορίσουμε τα αρχεία ή τους καταλόγους που θα ελεγχθούν και να προσαρμόσουμε τη συμπεριφορά του linter χρησιμοποιώντας τις επιλογές της γραμμής εντολών. Μερικά linters έχουν επίσης γραφικές διεπαφές χρήστη ή μπορούν να ενσωματωθούν με επεξεργαστές κώδικα ή άλλα εργαλεία ανάπτυξης.

Στην εργασία μας χρησιμοποιήθηκαν οι εξής Python Linters:

- **mypy**:

Το mypy είναι ένας έλεγχος στατικού τύπου για την Python. Μας επιτρέπει να προσθέτουμε σχολιασμούς τύπου στον κώδικα Python μας και ελέγχει ότι οι τύποι μεταβλητών και παραστάσεων είναι σωστοί κατά το χρόνο εκτέλεσης. Αυτό μπορεί να μας βοηθήσει να εντοπίσουμε σφάλματα τύπου και άλλα ζητήματα προτού αναπτύξουμε τον κώδικά μας, και μπορεί επίσης να κάνει τον κώδικά μας πιο ευανάγνωστο και διατηρήσιμο, προσθέτοντας σαφείς πληροφορίες τύπου.

Το mypy υλοποιείται ως λειτουργική μονάδα Python και μπορεί να εγκατασταθεί χρησιμοποιώντας pip. Μόλις εγκατασταθεί, μπορούμε να εκτελέσουμε το mypy στον κώδικά μας χρησιμοποιώντας το εργαλείο γραμμής εντολών mypy. Το mypy θα αναλύσει τον κώδικά μας και θα αναφέρει τυχόν σφάλματα τύπου ή προβλήματα που εντοπίζει.

Για να χρησιμοποιήσουμε το mypy, θα χρειαστεί να προσθέσουμε σχολιασμούς τύπου στον κώδικά μας. Οι σχολιασμοί τύπων γράφονται χρησιμοποιώντας μια ειδική σύνταξη που ακολουθεί τη μεταβλητή ή την έκφραση που θέλουμε να σχολιάσουμε. Για παράδειγμα:

```
def greet(name: str) -> str:
    return f'Hello, {name}!'
```

Σε αυτό το παράδειγμα, η παράμετρος ονόματος σημειώνεται με τον τύπο str και η συνάρτηση χαιρετισμού σχολιάζεται με τον τύπο επιστροφής str. Το mypy θα ελέγξει ότι η παράμετρος ονόματος είναι πάντα μια συμβολοσειρά και ότι η συνάρτηση χαιρετισμού επιστρέφει πάντα μια συμβολοσειρά.

Το mypy μπορεί επίσης να ρυθμιστεί ώστε να χρησιμοποιεί στελέχη προσαρμοσμένου τύπου και αρχεία υποδείξεων για να παρέχει πληροφορίες τύπου για εξωτερικές βιβλιοθήκες και λειτουργικές μονάδες. Αυτό μπορεί να είναι χρήσιμο εάν η βιβλιοθήκη δεν περιλαμβάνει σχολιασμούς τύπου στον πηγαίο κώδικα.

Το mypy είναι ένα δημοφιλές εργαλείο μεταξύ των προγραμματιστών Python και χρησιμοποιείται ευρέως σε περιβάλλοντα παραγωγής για την καταγραφή σφαλμάτων τύπου

και τη βελτίωση της ποιότητας του κώδικα Python. Συντηρείται ενεργά και έχει μια ισχυρή κοινότητα συντελεστών και χρηστών.

- **flake8**:

Το Flake8 είναι ένα εργαλείο για τον έλεγχο του κώδικα Python για σφάλματα σύνταξης, προβλήματα στυλ και άλλα πιθανά προβλήματα. Είναι ένα περιτύλιγμα γύρω από μια σειρά από άλλα εργαλεία, συμπεριλαμβανομένων των PyFlakes, pycodestyle και ελέγχου πολυπλοκότητας McCabe, και μπορεί να χρησιμοποιηθεί για τον έλεγχο του κώδικα Python για διάφορα ζητήματα.

Εδώ είναι μερικά από τα βασικά χαρακτηριστικά του Flake8:

- **Έλεγχος σύνταξης**: Το Flake8 ελέγχει τον κώδικα Python μας για συντακτικά σφάλματα, όπως αγκύλες που λείπουν, μη έγκυρους χαρακτήρες ή μη κλειστά εισαγωγικά. Μπορεί επίσης να ελέγξει τον κώδικά μας για ζητήματα όπως αχρησιμοποίητες μεταβλητές ή χρήση καταργημένων συναρτήσεων.

- **Έλεγχος στυλ**: Το Flake8 μπορεί να ελέγξει τον κώδικά μας για ζητήματα στυλ, όπως λανθασμένη εσοχή, περιττό κενό διάστημα ή χρήση απαγορευμένων ονομάτων συναρτήσεων. Μπορεί επίσης να ελέγξει τον κωδικό μας για συμμόρφωση με έναν συγκεκριμένο οδηγό στυλ, όπως το PEP 8.

- **Έλεγχος πολυπλοκότητας**: Το Flake8 περιλαμβάνει τον έλεγχο πολυπλοκότητας McCabe, ο οποίος μπορεί να ελέγξει τον κώδικά μας για υψηλά επίπεδα πολυπλοκότητας. Αυτό μπορεί να μας βοηθήσει να προσδιορίσουμε περιοχές του κώδικά μας που μπορεί να είναι δύσκολο να κατανοηθούν ή να διατηρηθούν και μπορεί να μας βοηθήσει να βελτιώσουμε τη συνολική ποιότητα του κώδικά μας.

Για να χρησιμοποιήσουμε το Flake8, θα χρειαστεί να το εγκαταστήσουμε και τυχόν απαιτούμενες εξαρτήσεις και στη συνέχεια να το εκτελέσουμε στον κώδικά μας. Το Flake8 διαθέτει μια διεπαφή γραμμής εντολών που μας επιτρέπει να καθορίσουμε τα αρχεία ή τους καταλόγους που θα ελεγχθούν και να προσαρμόσουμε τη συμπεριφορά του χρησιμοποιώντας επιλογές γραμμής εντολών. Μπορούμε επίσης να ενσωματώσουμε το Flake8 με επεξεργαστές κώδικα ή άλλα εργαλεία ανάπτυξης χρησιμοποιώντας πρόσθετα ή άλλους μηχανισμούς ενσωμάτωσης.

3.5.8.3 SCSS

Το SCSS (Sassy CSS) είναι ένας προεπεξεργαστής για CSS που προσθέτει χαρακτηριστικά όπως μεταβλητές, mixins και ένθετους κανόνες στη γλώσσα CSS. Μάς επιτρέπει να γράφουμε πιο συνοπτικό και διατηρήσιμο CSS χρησιμοποιώντας αυτές τις πρόσθετες δυνατότητες.

Το linter είναι ένα εργαλείο που αναλύει τον κώδικά μας και ελέγχει για ζητήματα όπως συντακτικά σφάλματα, προβλήματα στυλ και πιθανά σφάλματα.
Τα Linters μπορούν να μας βοηθήσουν να διατηρήσουμε μια συνεπή βάση κώδικα και να εντοπίσουμε προβλήματα πριν αναπτύξουμε τον κώδικά μας.

Υπάρχουν πολλές δημοφιλείς βάσεις SCSS διαθέσιμες, όπως:

- **Stylelint**: Πρόκειται για μια ευρέως χρησιμοποιούμενη θήκη SCSS που έχει ένα μεγάλο σύνολο κανόνων και είναι εξαιρετικά διαμορφώσιμη. Μπορεί να χρησιμοποιηθεί με άλλους προεπεξεργαστές όπως το Less και το Sass.

- **Sass-Lint**: Αυτό είναι ένα linter ειδικά για Sass και SCSS. Έχει ένα σύνολο κανόνων τους οποίους μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε για να προσαρμόσουμε τη διαδικασία του λιπαντικού.

Για να χρησιμοποιήσουμε μια επένδυση SCSS, θα χρειαστεί να την εγκαταστήσουμε και να την διαμορφώσουμε στο έργο μας. Αυτό συνήθως περιλαμβάνει τη δημιουργία ενός αρχείου διαμόρφωσης (όπως .stylelintrc ή .sass-lint.yml) που καθορίζει τους κανόνες και τις ρυθμίσεις για το linter. Στη συνέχεια, θα εκτελέσουμε το linter ως μέρος της διαδικασίας κατασκευής μας ή ως ένα άγκιστρο προ-δέσμευσης για να εντοπίσουμε τυχόν προβλήματα πριν δεσμεύσουμε τον κώδικά μας.

Αξίζει να σημειωθεί ότι υπάρχουν επίσης διαθέσιμα εργαλεία για την αυτόματη μορφοποίηση του κώδικα SCSS ώστε να ακολουθεί ένα συγκεκριμένο σύνολο οδηγιών στυλ, όπως το Prettier και το Sassfmt. Αυτά τα εργαλεία μπορούν να μας βοηθήσουν να επιβάλουμε ένα συνεπές στυλ στη βάση κωδίκων μας και να μειώσουμε την ανάγκη για μη αυτόματο έλεγχο κώδικα.

3.5.9 Prettier

Το Prettier είναι ένας μορφοποιητής κώδικα που διαμορφώνει αυτόματα τον κώδικά μας σύμφωνα με ένα σύνολο προκαθορισμένων κανόνων. Έχει σχεδιαστεί για να απομακρύνει το βάρος της μορφοποίησης από τους προγραμματιστές, ώστε να μπορούν να επικεντρωθούν στο περιεχόμενο του κώδικά τους και όχι στη μορφοποίησή του.

Το Prettier υποστηρίζει ένα ευρύ φάσμα γλωσσών προγραμματισμού, συμπεριλαμβανομένων των JavaScript, TypeScript, HTML, XML, Markdown και πολλών άλλων. Μπορεί να χρησιμοποιηθεί σε διάφορα περιβάλλοντα, όπως αυτόνομα εργαλεία γραμμής εντολών, επεξεργαστές κώδικα και ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE).

Εδώ είναι μερικά από τα βασικά χαρακτηριστικά του Prettier:

- **Αυτόματη μορφοποίηση**: Το Prettier επαναμορφοποιεί αυτόματα τον κώδικά μας σύμφωνα με ένα σύνολο προκαθορισμένων κανόνων, όπως το μέγεθος της εσοχής, το μήκος γραμμής και η χρήση ερωτηματικών. Αυτό σημαίνει ότι δεν χρειάζεται να ανησυχούμε για τη μη αυτόματη μορφοποίηση του κώδικά μας και μπορούμε να εστιάσουμε στο περιεχόμενο του κώδικά μας.

- **Υποστήριξη για πολλές γλώσσες**: Το Prettier υποστηρίζει ένα ευρύ φάσμα γλωσσών προγραμματισμού, καθιστώντας το εύκολο στη χρήση με μια ποικιλία έργων. Διαθέτει επίσης πρόσθετα για πολλούς επεξεργαστές κώδικα και IDE, καθιστώντας εύκολη την ενσωμάτωση με την υπάρχουσα ροή εργασίας μας.

- **Κανόνες με δυνατότητα διαμόρφωσης**: Ενώ το Prettier συνοδεύεται από ένα σύνολο προεπιλεγμένων κανόνων μορφοποίησης, μπορούμε επίσης να προσαρμόσουμε αυτούς τους κανόνες ώστε να ταιριάζουν στις προτιμήσεις μας ή στον οδηγό στυλ του έργου μας. Αυτό μπορεί να γίνει χρησιμοποιώντας ένα αρχείο διαμόρφωσης ή μέσω επιλογών γραμμής εντολών.

- **Μορφοποίηση κατά την αποθήκευση**: Πολλοί επεξεργαστές κώδικα και IDE έχουν προσθήκες που μας επιτρέπουν να μορφοποιούμε αυτόματα τον κώδικά μας κάθε φορά που αποθηκεύουμε ένα αρχείο. Αυτό μπορεί να μας βοηθήσει να διασφαλίσουμε ότι ο κώδικάς μας είναι πάντα σωστά μορφοποιημένος και μπορεί να μας εξοικονομήσει χρόνο και προσπάθεια.

Το Prettier είναι ένα δημοφιλές εργαλείο μεταξύ των προγραμματιστών και χρησιμοποιείται συχνά σε συνδυασμό με άλλα εργαλεία, όπως τα linters, τα οποία μπορούν να ελέγξουν τον κώδικά μας για συντακτικά λάθη και άλλα ζητήματα. Μαζί, αυτά τα εργαλεία μπορούν να μας βοηθήσουν να γράφουμε και να διατηρούμε κώδικα υψηλής ποιότητας, συνεπούς και εύκολα αναγνώσιμο.

3.5.10 Sourcetree

Το SourceTree είναι μια γραφική διεπαφή χρήστη (GUI) για συστήματα ελέγχου εκδόσεων όπως το Git και το Mercurial. Επιτρέπει στους προγραμματιστές να διαχειρίζονται τα αποθετήρια κώδικα τους, να προβάλλουν και να επιλύουν διενέξεις και να εκτελούν άλλες εργασίες ελέγχου έκδοσης χρησιμοποιώντας μια οπτική διεπαφή αντί της γραμμής εντολών.

Ορισμένα βασικά χαρακτηριστικά του SourceTree περιλαμβάνουν:

- **Οπτικοποίηση του ιστορικού του αποθετηρίου και των αλλαγών**: Το SourceTree εμφανίζει το ιστορικό δέσμευσης και τις αλλαγές σε γραφική μορφή, διευκολύνοντας την κατανόηση της εξέλιξης της βάσης κώδικα.

- **Διαχείριση διακλαδώσεων και συγχωνεύσεων**: Το SourceTree μάς επιτρέπει να δημιουργούμε και να εναλλάσσουμε κλάδους, να προβάλλουμε και να επιλύουμε διενέξεις κατά τη συγχώνευση κλάδων και να εκτελούμε άλλες εργασίες διαχείρισης υποκαταστημάτων.

- **Δυνατότητες συνεργασίας**: Το SourceTree περιλαμβάνει εργαλεία για τη συνεργασία με άλλους προγραμματιστές, όπως τη δυνατότητα προβολής και ελέγχου αιτημάτων έλξης, συζήτησης αλλαγών και συνεργασίας σε αξιολογήσεις κώδικα.

- **Ενσωμάτωση με άλλα εργαλεία**: Το SourceTree μπορεί να ενσωματωθεί με άλλα εργαλεία, όπως προγράμματα παρακολούθησης ζητημάτων και συστήματα ελέγχου κώδικα, επιτρέποντάς μας να διαχειρίζεστε τη ροή εργασιών ανάπτυξής μας από μια ενιαία διεπαφή.

Για να χρησιμοποιήσουμε το SourceTree, θα χρειαστεί να το εγκαταστήσουμε στον υπολογιστή μας και να το ρυθμίσουμε με το σύστημα ελέγχου έκδοσης της επιλογής μας. Το SourceTree είναι διαθέσιμο για Windows, macOS και Linux.

Το SourceTree αναπτύσσεται και συντηρείται από την Atlassian, μια εταιρεία που ειδικεύεται στην ανάπτυξη λογισμικού και στα εργαλεία συνεργασίας. Χρησιμοποιείται ευρέως από προγραμματιστές και ομάδες που εργάζονται σε έργα που χρησιμοποιούν Git ή Mercurial για έλεγχο έκδοσης.

ΚΕΦΑΛΑΙΟ 4. User Interface And User Experience

4.1 User Interface του Admin Panel

Ο πίνακας διαχείρισης παρέχει ένα εύχρηστο διαδικτυακό περιβάλλον για τη διαχείριση των δεδομένων που είναι αποθηκευμένα στη βάση δεδομένων μιας εφαρμογής Django.

Ο πίνακας διαχείρισης του Django είναι κατασκευασμένος χρησιμοποιώντας το ενσωματωμένο σύστημα προτύπων του Django και είναι ιδιαίτερα προσαρμόσιμος. Η διεπαφή χρήστη (UI) του admin panel έχει σχεδιαστεί ώστε να είναι απλή και διαισθητική, καθιστώντας τη χρήση του εύκολη για τους προγραμματιστές και τους διαχειριστές του ιστότοπου.

Ο πίνακας διαχείρισης παρέχει μια σειρά χαρακτηριστικών, όπως:

- **Έλεγχος ταυτότητας και εξουσιοδότηση χρηστών**: Ο πίνακας διαχείρισης παρέχει ένα ενσωματωμένο σύστημα ελέγχου ταυτότητας, το οποίο επιτρέπει στους διαχειριστές του ιστότοπου να δημιουργούν και να διαχειρίζονται λογαριασμούς χρηστών, καθώς και να εκχωρούν δικαιώματα σε αυτούς τους χρήστες.

- **Διαχείριση μοντέλων**: Ο πίνακας διαχείρισης παρέχει μια διαδικτυακή διεπαφή για τη διαχείριση των μοντέλων δεδομένων που ορίζονται στην εφαρμογή Django. Αυτό περιλαμβάνει τη δημιουργία, την επεξεργασία και τη διαγραφή εγγραφών στη βάση δεδομένων.

- **Προσαρμογή**: Ο πίνακας διαχείρισης είναι εξαιρετικά προσαρμόσιμος, επιτρέποντας στους προγραμματιστές να δημιουργούν προσαρμοσμένες προβολές, φόρμες και πρότυπα που ανταποκρίνονται στις συγκεκριμένες ανάγκες τους.

- **Αναζήτηση και φιλτράρισμα**: Ο πίνακας διαχείρισης περιλαμβάνει ένα σύστημα αναζήτησης και φιλτραρίσματος που επιτρέπει στους χρήστες να βρίσκουν και να επεξεργάζονται εύκολα δεδομένα.

- **Ενέργειες**: Ο πίνακας διαχείρισης παρέχει έναν τρόπο εκτέλεσης μαζικών ενεργειών σε επιλεγμένες εγγραφές. Για παράδειγμα, μπορούμε να διαγράψουμε πολλές εγγραφές ταυτόχρονα ή να ενημερώσουμε πολλές εγγραφές με μία μόνο ενέργεια.

Συνολικά, ο πίνακας διαχείρισης του Django είναι ένα ισχυρό εργαλείο για τη διαχείριση των δεδομένων στη βάση δεδομένων της εφαρμογής Django. Το παραμετροποιήσιμο UI του το καθιστά εύκολο στη χρήση και παρέχει ένα ευρύ φάσμα λειτουργιών για προγραμματιστές και διαχειριστές ιστότοπων.

4.2 User Interface and User Experience του E-Shop

Το e-shop μας αποτελεί τον κύριο κανάλι μέσω του οποίου προσφέρουμε τα προϊόντα/υπηρεσίες μας στους πελάτες μας. Η βελτιστοποίηση της εμπειρίας του χρήστη και η ευκολία στην πλοήγηση είναι για εμάς προτεραιότητα. Η σημασία του UI/UX στην επιτυχία ενός e-shop είναι αδιαμφισβήτητη, καθώς ένα οργανωμένο και ευχάριστο περιβάλλον αναζήτησης συμβάλλει στην ικανοποίηση του πελάτη.

Η δομή και ο σχεδιασμός του e-shop μας, που είναι χτισμένο με το προηγμένο front-end framework Nuxt.js, παρέχουν μία άρτια και ολοκληρωμένη εμπειρία στον χρήστη. Το Nuxt.js, επιτρέπει την δημιουργία ενός γρήγορου και αποκρισιμότητου website, προσφέροντας ένα άνετο και απρόσκοπτο περιβάλλον αναζήτησης και αγοράς προϊόντων.

- **UI (User Interface)**: Ο σχεδιασμός της διεπαφής του e-shop μας έχει γίνει με τέτοιο τρόπο ώστε να είναι λειτουργικός και ελκυστικός. Το μενού είναι οργανωμένο και παρέχει εύκολη πρόσβαση στα προϊόντα. Οι καρτέλες των προϊόντων παρουσιάζονται με καθαρό τρόπο, ενώ το καλάθι αγορών είναι πάντα προσβάσιμο. Η χρήση χρωμάτων, γραφικών, και τυπογραφίας έχει γίνει με τρόπο που ενισχύει την αισθητική και την ευχαρίστηση του χρήστη.

- **UX (User Experience)**: Η δομή και ο σχεδιασμός του e-shop είναι τέτοιοι που διευκολύνουν τον χρήστη στην αναζήτηση και αγορά προϊόντων. Χάρη στην ευελιξία που προσφέρει το Nuxt.js, μπορούμε γρήγορα και αποτελεσματικά να εντοπίσουμε και να διορθώσουμε τυχόν προβλήματα UI, βελτιστοποιώντας συνεχώς την εμπειρία του χρήστη.

Το e-shop μας είναι πλήρως προσβάσιμο από διάφορες συσκευές και browsers, προκειμένου να είναι προσβάσιμο από όλους τους πελάτες μας, ανεξαρτήτως του τρόπου πρόσβασής τους στο Internet. Η απόδοση του e-shop είναι για εμάς προτεραιότητα, και χρησιμοποιούμε ποικίλες τεχνικές για να εξασφαλίσουμε την ταχύτητα φόρτωσης και την άριστη λειτουργία της πλατφόρμας.

ΚΕΦΑΛΑΙΟ 5. Μεθοδολογία

Η πτυχιακή εργασία αφορά την ανάπτυξη ενός ευρύτατου συστήματος που περιλαμβάνει backend API και Admin panel σε Django, frontend σε Nuxt.js και ένα media stream application σε Nest.js. Καταρχάς, θα αναλυθούν ξεχωριστά οι τεχνολογίες που χρησιμοποιήθηκαν στο project.

Η υλοποίηση ξεκίνησε με τη δημιουργία των βασικών δομών δεδομένων στο Django και συνεχίστηκε με τη σχεδίαση της διεπαφής χρήστη στο Nuxt.js. Ταυτόχρονα, αναπτύχθηκαν οι βασικές λειτουργίες του media streaming application στο Nest.js. Μετά την ολοκλήρωση των βασικών λειτουργιών και της διεπαφής, προχωρήσαμε στην ενσωμάτωση των τριών μερών, εξασφαλίζοντας την ομαλή επικοινωνία μεταξύ τους. Τέλος, ελέγχθηκε η λειτουργικότητα και η ασφάλεια του συστήματος, καθώς και η απόδοση του media streaming application, εξασφαλίζοντας την ομαλή λειτουργία σε όλες τις πτυχές του project.

5.1 Backend API και Admin Panel (Django)

5.1.1 Βασικές λειτουργίες

- **Δημιουργία Μοντέλων**: Ορίστηκαν τα βασικά μοντέλα δεδομένων στο Django για την αποθήκευση πληροφοριών.
- **Διαχείριση Διαδρομών**: Ορισμός των διαδρομών (routes) στο αρχείο urls.py για την προσπέλαση των διάφορων σελίδων και API endpoints.
- **Δημιουργία Ελεγκτών (Controllers)**: Υλοποίηση των ελεγκτών για τη διαχείριση των αιτήσεων του χρήστη και την προβολή των δεδομένων.
- **Admin Panel**: Σχεδίαση και υλοποίηση του Admin Panel με χρήση των ενσωματωμένων εργαλείων του Django.
- **Αυθεντικοποίηση και Εξουσιοδότηση**: Ένας άλλος σημαντικός τομέας όπου το Django αποδεικνύεται πολύτιμο είναι η αυθεντικοποίηση και η εξουσιοδότηση. Το Django διαχειρίζεται την αυθεντικοποίηση των χρηστών μέσω JWT (JSON Web Tokens), προσφέροντας έναν ασφαλή και αποτελεσματικό τρόπο για την επαλήθευση των χρηστών και τη διατήρηση της περιόδου λειτουργίας τους στο σύστημα. Μέσω των JWT tokens, είναι επίσης δυνατή η εξουσιοδότηση των χρηστών, εξασφαλίζοντας ότι έχουν πρόσβαση μόνο στους πόρους και τις λειτουργίες που τους επιτρέπεται.

5.1.2 Αποσπάσματα Κώδικα

Παρακάτω παρουσιάζονται αποσπάσματα από τον κώδικα της εφαρμογής, τα οποία περιλαμβάνουν παραδείγματα από Model, Urls, Admin, View, Task, Serializer, Paginator, Filter, Test και Command.

5.1.3 Model (Το μοντέλο που αντιπροσωπεύει τις δομές δεδομένων)

```
class Product(TranslatableModel, TimeStampMixinModel, SeoModel,
UUIDModel):
    id = models.BigAutoField(primary_key=True)
    product_code = models.CharField(
        _("Product Code"), unique=True, max_length=100,
default=uuid.uuid4
    )
    category = TreeForeignKey(
        "product.ProductCategory",
        on_delete=models.SET_NULL,
        related_name="product_category",
        null=True,
        blank=True,
    )
    slug = models.SlugField(_("Slug"), max_length=255, unique=True)
    price = MoneyField(
        _("Price"),
        max_digits=19,
        decimal_places=4,
        default=0,
    )
    active = models.BooleanField(_("Active"), default=True)
    stock = models.PositiveIntegerField(_("Stock"), default=0)
    discount_percent = models.DecimalField(
        _("Discount Percent"), max_digits=11, decimal_places=2,
default=0.0
    )
    vat = models.ForeignKey(
        "vat.Vat",
        related_name="product_vat",
        blank=True,
        null=True,
        on_delete=models.SET_NULL,
    )
    hits = models.PositiveIntegerField(_("Hits"), default=0)
    weight = models.DecimalField(
        _("Weight (kg)"), max_digits=11, decimal_places=2,
default=0.0
    )

    # final_price, discount_value, price_save_percent are
calculated fields on save method
    final_price = MoneyField(
        _("Final Price"),
        max_digits=19,
        decimal_places=4,
        default=0,
        editable=False,
    )
    discount_value = MoneyField(
```

```
        _("Discount Value"),
        max_digits=19,
        decimal_places=4,
        default=0,
        editable=False,
    )
    price_save_percent = models.DecimalField(
        _("Price Save Percent"),
        max_digits=11,
        decimal_places=2,
        default=0.0,
        editable=False,
    )

    translations = TranslatedFields(
        name=models.CharField(
            _("Name"), max_length=255, blank=True, null=True,
db_index=True
        ),
        description=HTMLField(_("Description"), blank=True,
null=True, db_index=True),
    )
    search_vector = SearchVectorField(blank=True, null=True)

    objects = ProductManager()

    class Meta(TypedModelMeta):
        verbose_name = _("Product")
        verbose_name_plural = _("Products")
        ordering = ["-created_at"]
        indexes = [
            GinIndex(fields=["search_vector"],
name="product_search_vector_idx"),
        ]

    def __unicode__(self):
        return self.safe_translation_getter("name",
any_language=True) or ""

    def __str__(self):
        return self.safe_translation_getter("name",
any_language=True) or ""

    @property
    def likes_counter(self) -> int:
        favourites =
ProductFavourite.objects.filter(product=self).aggregate(
            count=Count("id")
        )
        cnt: int = 0
        if favourites["count"] is not None:
            cnt = int(favourites["count"])
        return cnt

    @property
    def review_average(self) -> float:
```

```
        reviews = ProductReview.objects.filter(product=self,
status="True").aggregate(
            average=Avg("rate")
        )
        avg: float = 0.0
        if reviews["average"] is not None:
            avg = float(reviews["average"])
        return avg

    @property
    def review_counter(self) -> int:
        reviews = ProductReview.objects.filter(product=self,
status="True").aggregate(
            count=Count("id")
        )
        cnt: int = 0
        if reviews["count"] is not None:
            return int(reviews["count"])
        return cnt

    @property
    def vat_percent(self) -> Decimal | int:
        if self.vat:
            return self.vat.value
        return 0

    @property
    def vat_value(self) -> Money:
        if self.vat:
            value = (self.price.amount * self.vat.value) / 100
            return Money(value, settings.DEFAULT_CURRENCY)
        return Money(0, settings.DEFAULT_CURRENCY)
```

Η κλάση Product αντιπροσωπεύει ένα προϊόν στη βάση δεδομένων. Η κλάση κληρονομεί από τα TranslatableModel, TimeStampMixinModel, SeoModel και UUIDModel, προσφέροντας λειτουργίες μετάφρασης, χρονοσήματος, SEO και μοναδικού αναγνωριστικού.

Ορίζονται πολλά πεδία, όπως product_code, category, slug, price, active, stock, discount_percent, vat, hits και weight. Κάποια από αυτά τα πεδία είναι απλά, όπως ακέραιοι ή χαρακτήρες, ενώ άλλα είναι πιο πολύπλοκα, όπως οι σχέσεις ForeignKey.

Τρία πεδία (final_price, discount_value, price_save_percent) είναι υπολογισμένα και δεν είναι επεξεργάσιμα, παράγονται αυτόματα βάσει των τιμών των υπόλοιπων πεδίων.

Η κλάση περιλαμβάνει επίσης μεταφραστικά πεδία για το name και την description, καθώς και ένα πεδίο αναζήτησης (search_vector). Πέραν των πεδίων, το μοντέλο ορίζει διάφορες ιδιότητες, όπως likes_counter, review_average, review_counter, vat_percent και vat_value, οι οποίες προσφέρουν υπολογισμένες τιμές βασισμένες στα δεδομένα του προϊόντος.

5.1.4 Urls (Ορισμός των διαδρομών της εφαρμογής)

```
urlpatterns = [
    # Product
    path(
        "product/",
        ProductViewSet.as_view({"get": "list", "post": "create"}),
        name="product-list",
    ),
    path(
        "product/<int:pk>/",
        ProductViewSet.as_view(
            {
                "get": "retrieve",
                "put": "update",
                "patch": "partial_update",
                "delete": "destroy",
            }
        ),
        name="product-detail",
    ),
    path(
        "product/<int:pk>/update_product_hits/",
        ProductViewSet.as_view({"post": "update_product_hits"}),
        name="product-update-product-hits",
    ),
]

urlpatterns = format_suffix_patterns(urlpatterns)
```

Στο παραπάνω παράδειγμα κώδικα καθορίζουμε τις διαδρομές που σχετίζονται με τα προϊόντα στην εφαρμογή μας.

Ας αναλύσουμε τα βασικά σημεία αυτών των διαδρομών:

- **path("product/")**: Αυτή η διαδρομή αντιστοιχεί στη λίστα των προϊόντων. Χρησιμοποιεί την προβολή ProductViewSet για τις HTTP GET και POST αιτήσεις. Το όνομα της διαδρομής είναι "product-list".

- **path("product/<int:pk>/")**: Αυτή η διαδρομή αντιστοιχεί στη λεπτομερή προβολή ενός προϊόντος με ένα συγκεκριμένο αναγνωριστικό (primary key). Χρησιμοποιεί την προβολή ProductViewSet για τις HTTP GET, PUT, PATCH και DELETE αιτήσεις. Το <int:pk> είναι μια μεταβλητή που αντιστοιχίζεται στο πρωτεύον κλειδί του προϊόντος. Το όνομα της διαδρομής είναι "product-detail".

- **path("product/<int:pk>/update_product_hits/")**: Αυτή η διαδρομή αντιστοιχεί στην ενημέρωση των προβολών (hits) ενός συγκεκριμένου προϊόντος με βάση το πρωτεύον κλειδί. Χρησιμοποιεί την προβολή ProductViewSet για τις HTTP POST αιτήσεις. Το <int:pk> αναπαριστά το πρωτεύον κλειδί του προϊόντος που θα ενημερωθεί. Το όνομα της διαδρομής είναι "product-update-product-hits".

5.1.5 Admin (Προσαρμογή της διαχειριστικής διεπαφής)

```
@admin.register(Product)
class ProductAdmin(TranslatableAdmin, ExportModelAdmin):
    list_display = [
        "id",
        "category",
        "price",
        "colored_stock",
        "boolean_status",
        "image_tag",
        "likes_counter",
    ]
    search_fields = ["id", "category__name", "translations__name",
"product_code"]
    list_filter = ["category"]
    inlines = [ProductImageInline]
    readonly_fields = ("image_tag",)

    def get_prepopulated_fields(self, request, obj=None) -> dict:
        # can't use `prepopulated_fields = ..` because it breaks
the admin validation
        # for translated fields. This is the official django-parler
workaround.
        return {
            "slug": ("name",),
        }

    def boolean_status(self, obj) -> bool:
        return True if obj.active else False

    setattr(
        boolean_status,
        "boolean",
        True,
    )

    actions = [
        "export_csv",
        "export_xml",
    ]
```

Ο παραπάνω κώδικας ορίζει τη διαχειριστική διεπαφή για το μοντέλο Product στην εφαρμογή μας. Προσαρμόζουμε τη διαχειριστική διεπαφή ώστε να παρέχει εξειδικευμένες λειτουργίες για τη διαχείριση των προϊόντων στην εφαρμογή μας. Συγκεκριμένα, καθορίζουμε τον τρόπο προβολής και επεξεργασίας των προϊόντων στον διαχειριστικό πίνακα (admin panel), καθιστώντας τον πιο λειτουργικό και ευκολοδιάβαστο για τους διαχειριστές της εφαρμογής μας.

Επιπλέον, προσθέτουμε διάφορες λειτουργίες όπως η δυνατότητα αναζήτησης και φιλτραρίσματος των προϊόντων, καθώς και η δυνατότητα εξαγωγής δεδομένων σε αρχεία CSV και XML. Αυτές οι λειτουργίες βοηθούν στην αποτελεσματική διαχείριση των προϊόντων στην εφαρμογή μας και στη διευκόλυνση της διαδικασίας διαχείρισης δεδομένων.

5.1.6 View (Η λογική πίσω από τις διαδρομές)

```
class ProductViewSet(BaseExpandView, ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    pagination_class = ProductPagination
    filter_backends = [DjangoFilterBackend,
PascalSnakeCaseOrderingFilter, SearchFilter]
    filterset_class = ProductFilter
    ordering_fields = [
        "price",
        "created_at",
        "discount_value",
        "final_price",
        "price_save_percent",
        "review_average",
        "likes_counter",
    ]
    ordering = ["-created_at"]
    search_fields = ["id"]

    @method_decorator(cache_page(60 * 60 * 2))
    def list(self, request, *args, **kwargs) -> Response:
        queryset = self.filter_queryset(self.get_queryset())
        page = self.paginate_queryset(queryset)
        if page is not None:
            serializer = self.get_serializer(page, many=True)
            return self.get_paginated_response(serializer.data)
        serializer = self.get_serializer(queryset, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    def create(self, request, *args, **kwargs) -> Response:
        serializer = self.get_serializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    def retrieve(self, request, pk=None, *args, **kwargs) ->
Response:
        product = get_object_or_404(Product, pk=pk)
        serializer = self.get_serializer(product)
        return Response(serializer.data, status=status.HTTP_200_OK)

    def update(self, request, pk=None, *args, **kwargs) ->
Response:
        product = get_object_or_404(Product, pk=pk)
        serializer = self.get_serializer(product,
data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_200_OK)
```

```
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    def partial_update(self, request, pk=None, *args, **kwargs) ->
Response:
        product = get_object_or_404(Product, pk=pk)
        serializer = self.get_serializer(product,
data=request.data, partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_200_OK)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    def destroy(self, request, pk=None, *args, **kwargs) ->
Response:
        product = get_object_or_404(Product, pk=pk)
        product.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

    @action(detail=True, methods=["POST"])
    def update_product_hits(self, request, pk=None, *args,
**kwargs) -> Response:
        product = get_object_or_404(Product, pk=pk)
        data = {"hits": product.hits + 1}
        serializer = self.get_serializer(product, data=data,
partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_200_OK)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

Ο παραπάνω κώδικας καθορίζει τη λογική πίσω από τις διαδρομές που σχετίζονται με το μοντέλο προϊόντος (Product) στην εφαρμογή μας.

Συγκεκριμένα:

- Η κλάση ProductViewSet κληρονομεί από τον BaseExpandView και τον ModelViewSet. Ορίζει το queryset για την ανάκτηση όλων των προϊόντων, το serializer_class που χρησιμοποιείται για τη μετατροπή των δεδομένων των προϊόντων σε μορφή JSON, καθώς και διάφορες ρυθμίσεις όπως τον τρόπο διαχείρισης της σελιδοποίησης, τα φίλτρα, τα πεδία που μπορούν να χρησιμοποιηθούν για τον ταξινομικό φορέα (ordering), και τα πεδία αναζήτησης (search_fields).

- Η μέθοδος list χρησιμοποιείται για την ανάκτηση μιας λίστας προϊόντων με σελιδοποίηση. Εάν υπάρχουν πολλά προϊόντα, τα δεδομένα σελιδοποιούνται για αποτελεσματική ανάκτηση. Η μέθοδος διαχειρίζεται τη φιλτράριση και τη σελιδοποίηση των προϊόντων.

- Οι μέθοδοι create, retrieve, update, partial_update και destroy αντιστοιχούν στις βασικές λειτουργίες CRUD (Δημιουργία, Ανάκτηση, Ενημέρωση, Μερική Ενημέρωση και Διαγραφή) για τα προϊόντα. Κάθε μέθοδος αναλαμβάνει τη διαχείριση αντίστοιχης αιτήσεις HTTP (POST, GET, PUT, PATCH, DELETE) και επικοινωνεί με τη βάση δεδομένων.

- Η δράση update_product_hits είναι μια προσαρμοσμένη δράση που αυξάνει τον αριθμό των προβολών ενός προϊόντος κατά 1 κάθε φορά που καλείται. Χρησιμοποιείται για να καταγράψει την προβολή ενός προϊόντος.

5.1.7 Task (Το κομμάτι του κώδικα που χειρίζεται κάποια εργασία)

```
@shared_task(bind=True, name="Clear Carts For None Users Task")
def clear_carts_for_none_users_task():
    from cart.models import Cart

    null_carts = Cart.objects.filter(user=None)
    null_carts.delete()

    message = f"Cleared {len(null_carts)} null carts."

    logger.info(message)
    return message
```

Ο παραπάνω κώδικας περιλαμβάνει μια εργασία με την ονομασία 'Clear Carts For None Users Task' στην εφαρμογή μας. Αυτή η εργασία είναι υπεύθυνη για τη διαχείριση ενός σημαντικού κομματιού της λογικής της εφαρμογής.

Συγκεκριμένα:

- Στην αρχή, χρησιμοποιούμε τον δεκοράτορα '@shared_task' για να δηλώσουμε την εργασία μας. Αυτό μας επιτρέπει να την εκτελέσουμε ασύγχρονα στο πλαίσιο της εφαρμογής.

- Στο εσωτερικό της εργασίας, πραγματοποιούμε τα παρακάτω βήματα:
    1. Εισάγουμε το μοντέλο 'Cart' από την εφαρμογή μας.
    2. Αναζητούμε και διαγράφουμε όλα τα καλάθια που δεν αντιστοιχούν σε κανέναν χρήστη. Καταγράφουμε τον αριθμό των διαγραφέντων καλαθιών και το αποθηκεύουμε σε ένα μήνυμα.
    3. Καταγράφουμε το μήνυμα στον καταγραφέα (logger) με επίπεδο πληροφοριών (info).
    4. Τέλος, επιστρέφουμε το μήνυμα που δείχνει πόσα καλάθια καθαρίστηκαν. Αυτή η εργασία μας βοηθά να διατηρήσουμε την εφαρμογή μας αποδοτική και να διαχειριστούμε αποτελεσματικά τα καλάθια αγορών που δεν αντιστοιχούν σε κανέναν χρήστη.

5.1.8 Serializer (Ορίζει πώς θα μετατρέπονται τα δεδομένα μοντέλου σε JSON και αντίστροφα)

```
class ProductSerializer(TranslatableModelSerializer,
BaseExpandSerializer):
   translations =
TranslatedFieldsFieldExtend(shared_model=Product)
   category =
PrimaryKeyRelatedField(queryset=ProductCategory.objects.all())
   vat = PrimaryKeyRelatedField(queryset=Vat.objects.all())
   price = MoneyField(max_digits=19, decimal_places=4)
   final_price = MoneyField(max_digits=19, decimal_places=4,
read_only=True)
   discount_value = MoneyField(max_digits=19, decimal_places=4,
read_only=True)
   vat_value = MoneyField(max_digits=19, decimal_places=4,
read_only=True)

   class Meta:
       model = Product
       fields = (
           "translations",
           "id",
           "slug",
           "category",
           "absolute_url",
           "price",
           "vat",
           "vat_percent",
           "vat_value",
           "final_price",
           "hits",
           "likes_counter",
           "stock",
           "active",
           "weight",
           "seo_title",
           "seo_description",
           "seo_keywords",
           "uuid",
           "discount_percent",
           "discount_value",
           "price_save_percent",
           "created_at",
           "updated_at",
           "main_image_absolute_url",
           "main_image_filename",
           "review_average",
           "review_counter",
       )
```

Ο παραπάνω κώδικας αφορά τη δήλωση του σειριαλιστή (serializer) για το μοντέλο "Product" στην εφαρμογή μας. Αυτός ο σειριαλιστής ορίζει πώς τα δεδομένα του μοντέλου θα μετατρέπονται σε μορφή JSON κατά την αποστολή τους μέσω της API μας και αντίστροφα κατά την αποδοχή δεδομένων από τον πελάτη.

Συγκεκριμένα:

- Χρησιμοποιούμε τον σειριαλιστή "TranslatableModelSerializer" για να υποστηρίξουμε μεταφράσιμα πεδία.

- Ορίζουμε το πεδίο "translations" χρησιμοποιώντας την κλάση "TranslatedFieldsFieldExtend" για να υποστηρίξουμε τις μεταφράσεις των πεδίων.

- Ορίζουμε τα υπόλοιπα πεδία του σειριαλιστή, όπως "category", "vat", "price", "final_price", "discount_value", "vat_value", κ.λπ., καθώς και τα πεδία που προέρχονται από το μοντέλο "Product".

- Στην κλάση "Meta", καθορίζουμε το μοντέλο που αντιστοιχεί στον σειριαλιστή και τα πεδία που πρέπει να περιληφθούν στο JSON αποτέλεσμα. Αυτά τα πεδία περιλαμβάνουν τα δεδομένα του προϊόντος, όπως η κατηγορία, η τιμή, το κωδικό του προϊόντος, το URL, το βάρος, τον τίτλο SEO, την περιγραφή SEO, κ.λπ.

Ο σειριαλιστής αυτός είναι υπεύθυνος για τον τρόπο μετατροπής των αντικειμένων του μοντέλου "Product" σε μορφή JSON που μπορεί να αποσταλεί στην εφαρμογή του πελάτη και για τον τρόπο αποδοχής JSON δεδομένων από τον πελάτη και μετατροπής τους σε αντικείμενα του μοντέλου κατά την επεξεργασία τους στην πλευρά του διακομιστή.

5.1.9 Paginator (Χειρίζεται τη διαίρεση των δεδομένων σε σελίδες)

```
class LimitOffsetPaginator(pagination.LimitOffsetPagination):
   def get_paginated_response(self, data) -> Response:
       return Response(
           {
               "links": {
                   "next": self.get_next_link(),
                   "previous": self.get_previous_link(),
               },
               "count": self.count,
               "total_pages": math.ceil(self.count / self.limit),
               "page_size": self.limit,
               "page_total_results": len(data),
               "page": math.ceil(self.offset / self.limit) + 1,
               "results": data,
           }
       )
```

Ο παραπάνω κώδικας αναφέρεται σε έναν προσαρμοσμένο paginator (διαχειριστής σελιδοποίησης) με το όνομα "LimitOffsetPaginator". Ο ρόλος του paginator είναι να διαχειρίζεται τη διαίρεση των δεδομένων σε σελίδες, προκειμένου να επιτρέπει την αναζήτηση και την πλοήγηση στα αποτελέσματα μιας αναζήτησης ή μιας λίστας αντικειμένων.

Ο συγκεκριμένος "LimitOffsetPaginator" επεκτείνει την κλάση "pagination.LimitOffsetPagination" που παρέχεται από το Django. Στη συνάρτηση "get_paginated_response", ορίζουμε πώς θα δημιουργηθεί η απόκριση (response) που θα σταλεί πίσω στον πελάτη μετά από μια αίτηση για σελιδοποίηση.

Η απόκριση περιλαμβάνει τα εξής δεδομένα:

- **"links"**: Συνδέσμους προς την επόμενη και προηγούμενη σελίδα (αν υπάρχουν).
- **"count"**: Το συνολικό πλήθος των αντικειμένων.
- **"total_pages"**: Ο συνολικός αριθμός των σελίδων που χρειάζονται για να εμφανιστούν όλα τα αντικείμενα.
- **"page_size"**: Το πλήθος των αντικειμένων που εμφανίζονται σε κάθε σελίδα.
- **"page_total_results"**: Το πλήθος των αντικειμένων στην τρέχουσα σελίδα.
- **"page"**: Ο αριθμός της τρέχουσας σελίδας.
- **"results"**: Τα δεδομένα της τρέχουσας σελίδας.

Ο "LimitOffsetPaginator" επιτρέπει τη διαίρεση των αποτελεσμάτων σε μικρότερες σελίδες, προσφέροντας έτσι μια ευέλικτη λύση για την επεξεργασία και την προβολή μεγάλου όγκου δεδομένων στην εφαρμογή μας.

5.1.10 Filter (Χειρίζεται τη φιλτράρισμα των δεδομένων)

```
class ProductFilter(filters.FilterSet):
   min_final_price =
filters.NumberFilter(field_name="final_price", lookup_expr="gte")
   max_final_price =
filters.NumberFilter(field_name="final_price", lookup_expr="lte")
   category = filters.CharFilter(field_name="category__id",
method="filter_category")

   class Meta:
       model = Product
       fields = ["min_final_price", "max_final_price", "category"]

   def filter_category(self, queryset, name, value):
       category_ids = value.split("_")
       return queryset.filter(category__id__in=category_ids)
```

Ο παραπάνω κώδικας αναφέρεται στη δημιουργία ενός προσαρμοσμένου φίλτρου για το μοντέλο Product στην εφαρμογή μας. Το φίλτρο αυτό επιτρέπει στους χρήστες να πραγματοποιούν προσαρμοσμένες αναζητήσεις βάσει συγκεκριμένων κριτηρίων. Αναλύοντας τα σημαντικότερα στοιχεία:

- Ορίζουμε τον τύπο του φίλτρου για το πεδίο final_price. Ορίζουμε δύο φίλτρα, το min_final_price και το max_final_price, που επιτρέπουν την αναζήτηση προϊόντων με βάση την τελική τιμή τους. Το lookup_expr καθορίζει τον τρόπο με τον οποίο θα εφαρμοστεί το φίλτρο (σε αυτήν την περίπτωση, "gte" για μεγαλύτερο ή ίσο από και "lte" για μικρότερο ή ίσο από).

- Το πεδίο category επιτρέπει την αναζήτηση προϊόντων βάσει της κατηγορίας τους. Το πεδίο αυτό δέχεται μια λίστα με τα αναγνωριστικά των κατηγοριών που επιθυμεί ο χρήστης να φιλτράρει και εφαρμόζει το φίλτρο ανάλογα.

- Ορίζουμε την κλάση Meta για να καθορίσουμε το μοντέλο που θα χρησιμοποιηθεί για το φίλτρο (στην περίπτωση μας, το μοντέλο Product) και τα πεδία που θα είναι διαθέσιμα για φιλτράρισμα.

- Η μέθοδος filter_category χρησιμοποιείται για το φιλτράρισμα βάσει της κατηγορίας. Αυτή η μέθοδο διαχειρίζεται τον διαχωρισμό των αναγνωριστικών των κατηγοριών από το πεδίο category και εφαρμόζει το φίλτρο στη συνέχεια.

Με τη χρήση αυτού του φίλτρου, οι χρήστες μπορούν να πραγματοποιούν εξειδικευμένες αναζητήσεις για προϊόντα βάσει τιμής και κατηγορίας, προσφέροντας μια πιο εξατομικευμένη εμπειρία αναζήτησης στην εφαρμογή μας.

5.1.11 Test (Τα tests που ελέγχουν τη σωστή λειτουργία της εφαρμογής)

```
class TestMakeThumbnail(TestCase):
    def test_make_thumbnail(self):
        # Create a test image using PIL
        img = Image.new("RGB", size=(300, 200), color="red")
        img_io = BytesIO()
        img.save(img_io, format="JPEG")
        img_io.seek(0)
        test_image = File(img_io, name="test_image.jpg")

        # Create a thumbnail using the make_thumbnail function
        thumbnail = make_thumbnail(test_image, (100, 100))

        # Ensure the thumbnail is a File object
        self.assertIsInstance(thumbnail, File)

        # Ensure the thumbnail is a JPEG
        self.assertEqual(thumbnail.name.split(".")[-1], "jpg")

        # Ensure the thumbnail is 100x100
        thumbnail_img = Image.open(thumbnail)
        self.assertEqual(thumbnail_img.size, (100, 67))

    def tearDown(self) -> None:
        super().tearDown()
        pass
```

Ο παραπάνω κώδικας αναφέρεται σε μια μονάδα δοκιμών (test) που χρησιμοποιείται για να ελέγξει τη σωστή λειτουργία μιας λειτουργίας δημιουργίας μικρογραφιών (thumbnails) στην εφαρμογή μας. Ας αναλύσουμε αυτό το test:

- Δημιουργία δοκιμαστικής εικόνας: Δημιουργούμε μια εικόνα 300x200 pixels με κόκκινο χρώμα χρησιμοποιώντας το PIL.

- Κατασκευή ενός αρχείου εικόνας: Αποθηκεύουμε τη δοκιμαστική εικόνα σε ένα αρχείο JPEG και το αντιστρέφουμε σε ένα αντικείμενο τύπου File.

- Δημιουργία μικρογραφίας: Χρησιμοποιούμε τη λειτουργία make_thumbnail για να δημιουργήσουμε μια μικρογραφία με διαστάσεις 100x100 pixels από τη δοκιμαστική εικόνα.

- Έλεγχος: Ελέγχουμε αν η μικρογραφία είναι ένα αντικείμενο τύπου File, αν το όνομα της είναι "jpg", και αν οι διαστάσεις της είναι 100x100 pixels.

Τα tests όπως αυτό εξασφαλίζουν ότι ο κώδικας λειτουργεί σωστά και δεν έχει προβλήματα όταν προστίθεται νέα λειτουργικότητα ή αλλαγές στον κώδικα. Επίσης, είναι ένας τρόπος να βεβαιωθούμε ότι οι αλλαγές στον κώδικα δεν έχουν ανεπιθύμητες επιπτώσεις σε υπάρχουσες λειτουργίες.

5.1.12 Command (Προσαρμοσμένες εντολές που μπορούν να εκτελεστούν από το command line)

```
class Command(BaseCommand):
    help = "Seed Product model."

    def add_arguments(self, parser):
        parser.add_argument(
            "total_products",
            type=int,
            help="Indicates the number of products to be seeded.",
            default=1000,
            nargs="?",
        )

    def handle(self, *args, **options):
        total_products = options["total_products"]
        total_time = 0
        start_time = time.time()
        available_languages = [
            lang["code"] for lang in
settings.PARLER_LANGUAGES[settings.SITE_ID]
        ]

        if total_products < 1:
            self.stdout.write(
                self.style.WARNING("Total number of products must
be greater than 0.")
            )
            return

        categories = list(ProductCategory.objects.all())
        vats = list(Vat.objects.all())

        if not categories or not vats:
            self.stdout.write(
```

```
                self.style.ERROR("Insufficient data. Aborting
seeding Product model.")
            )
            return

        if not available_languages:
            self.stdout.write(self.style.ERROR("No languages
found."))
            return

        objects_to_insert = []
        with transaction.atomic():
            for _ in range(total_products):
                category = faker.random_element(categories)
                vat = faker.random_element(vats)
                slug = faker.unique.slug()
                price = faker.pydecimal(left_digits=4,
right_digits=2, positive=True)
                stock = faker.random_int(min=0, max=1000)
                discount_percent = faker.pydecimal(
                    left_digits=2, right_digits=2, positive=True,
max_value=99
                )
                weight = faker.pydecimal(left_digits=3,
right_digits=2, positive=True)

                product_slug_exists =
Product.objects.filter(slug=slug).exists()
                if product_slug_exists:
                    continue

                product = Product(
                    slug=slug,
                    category=category,
                    vat=vat,
                    price=price,
                    stock=stock,
                    discount_percent=discount_percent,
                    weight=weight,
                )
                objects_to_insert.append(product)
            Product.objects.bulk_create(objects_to_insert)

            for product in objects_to_insert:
                for lang in available_languages:
                    lang_seed = hash(f"{lang}{product.id}")
                    faker.seed_instance(lang_seed)

                    name = f"{faker.word()}_{product.id}"
                    description = f"{faker.text()}_{product.id}"
                    product.set_current_language(lang)
                    product.name = name
                    product.description = description
                    product.save()

        end_time = time.time()
```

```
        execution_time = end_time - start_time
        total_time += execution_time
        self.stdout.write(
            self.style.SUCCESS(
                f"{len(objects_to_insert)} Product instances
created successfully in {execution_time:.2f} seconds."
            )
        )
```

Ο παραπάνω κώδικας αναφέρεται σε ένα προσαρμοσμένο command του Django που χρησιμοποιείται για τη σπορά (seeding) δεδομένων στο μοντέλο Product της εφαρμογής σας.

Ας αναλύσουμε τον κώδικα και τις λειτουργίες που προσφέρει:

- Δημιουργία Ορισμάτων: Η μέθοδος add_arguments προσθέτει ένα προαιρετικό όρισμα total_products στην εντολή για τον αριθμό των προϊόντων που θα σπείρει. Μπορείτε να καθορίσετε τον αριθμό αυτόν όταν καλείτε την εντολή από τη γραμμή εντολών.

- Δημιουργία Δεδομένων: Στη μέθοδο handle, γίνεται η δημιουργία δεδομένων προϊόντων. Αυτά τα δεδομένα περιλαμβάνουν κατηγορίες (categories) και φόρους (vats), και έπειτα δημιουργούνται τυχαία προϊόντα με τυχαία χαρακτηριστικά.

- Έλεγχοι: Υπάρχουν διάφοροι έλεγχοι για να βεβαιωθούμε ότι τα δεδομένα που χρειαζόμαστε είναι διαθέσιμα και ότι τα ορίσματα είναι έγκυρα.

- Δημιουργία Δεδομένων σε Πολλές Γλώσσες: Τα προϊόντα δημιουργούνται σε πολλές γλώσσες, εφόσον υπάρχουν γλώσσεις που καθορίζονται στις ρυθμίσεις (settings.PARLER_LANGUAGES).

- Στατιστικά: Εν τέλει, εμφανίζεται μια επιτυχημένη ενημέρωση με τον χρόνο που χρειάστηκε για τη δημιουργία των προϊόντων.

5.2 Frontend (Nuxt.js): Σχεδίαση Διεπαφής:

5.2.1 Βασικές λειτουργίες

- Σχεδιασμός των σελίδων και των στοιχείων της διεπαφής χρήστη.

- Διαχείριση Καταστάσεων: Χρήση Pinia για τη διαχείριση των καταστάσεων της εφαρμογής.

- Επικοινωνία με το Backend: Δημιουργία υπηρεσιών για την επικοινωνία με το backend API.

5.2.2 Αποσπάσματα Κώδικα

Παρακάτω παρουσιάζονται αποσπάσματα από τον κώδικα της εφαρμογής, τα οποία περιλαμβάνουν παραδείγματα από Page, Component, Layout, Composable, Middleware, Module, Plugin, Store, Util καθώς και αποσπάσματα του Nuxt κατάλογο διακομιστή (server directory) Api, Middleware και Utils.

5.2.3 Client

Από τον κατάλογο πελάτη του Nuxt (client directory):

5.2.3.1 Page (Σελίδα)

```
<script lang="ts" setup>
import emptyIcon from '~icons/mdi/package-variant-remove'

const cartStore = useCartStore()
const { cart, pending } = storeToRefs(cartStore)

const { t } = useLang()

definePageMeta({
 layout: 'page'
})
useServerHead(() => ({
 title: t('pages.cart.title'),
 meta: [
  {
   name: 'description',
   content: t('pages.cart.description')
  },
  {
   name: 'keywords',
   content: t('pages.cart.keywords')
  }
 ]
}))
useServerSeoMeta({
 title: t('pages.cart.title'),
 description: t('pages.cart.description')
})
</script>

<template>
 <PageWrapper class="container grid gap-4 grid-rows-auto-1fr">
  <div class="grid grid-cols-2 items-center">
   <PageTitle:text="$t('pages.cart.title')" class="capitalize" />
   <h2 class="grid justify-items-center justify-self-end">
    <MainButton
    :text="$t('pages.cart.checkout')"
    :type="'link'"
```

```
      class="font-extrabold capitalize"
     :to="'checkout'"
     />
    </h2>
   </div>
   <PageBody>
    <template v-if="!pending.cart && cart?.cartItems?.length">
     <div class="grid grid-rows-repeat-auto-fill-mimax-100-130
gap-4">
      <CartItemCard
       v-for="(cartItem, index) in cart.cartItems"
      :key="index"
      :cart-item="cartItem"
      />
     </div>
    </template>
    <template v-if="!pending.cart && !cart?.cartItems?.length">
     <EmptyState:icon="emptyIcon">
      <template #actions>
       <MainButton
       :text="$t('common.empty.button')"
       :type="'link'"
       :to="'index'"
       ></MainButton>
      </template>
     </EmptyState>
    </template>
   </PageBody>
  </PageWrapper>
</template>
```

Σε αυτό το παράδειγμα, η σελίδα του καλαθιού (cart page) της εφαρμογής μου υλοποιείται ως μια Vue σελίδα που χρησιμοποιεί το composition API του Vue 3. Η σελίδα αυτή αποτελείται από διάφορα στοιχεία, όπως το PageWrapper, το PageTitle, το PageBody, το MainButton και το CartItemCard, τα οποία χρησιμοποιούνται για την κατασκευή της δομής της σελίδας.

Ο κώδικας ξεκινά με την εισαγωγή των απαραίτητων εξαρτήσεων και τη δήλωση των καταστημάτων (stores) και Σύνθετες Λειτουργίες (composables) που χρησιμοποιούνται, όπως το cartStore, το useLang και τα σχετικά hooks για τη διαχείριση της κατάστασης της σελίδας.

Στη συνέχεια, χρησιμοποιούνται τα hooks definePageMeta και useServerHead για τη δήλωση των μεταδεδομένων της σελίδας, όπως τον τίτλο, την περιγραφή και τις λέξεις-κλειδιά που θα χρησιμοποιηθούν για το SEO.

Στο τμήμα του template, χρησιμοποιούνται τα στοιχεία PageWrapper, PageTitle και PageBody για τη δημιουργία της βασικής δομής της σελίδας. Το στοιχείο MainButton χρησιμοποιείται για την παροχή ενός κουμπιού που οδηγεί τον χρήστη στη σελίδα του ταμείου (checkout). Το στοιχείο CartItemCard χρησιμοποιείται για την παρουσίαση των προϊόντων που βρίσκονται στο καλάθι. Τέλος, το στοιχείο EmptyState χρησιμοποιείται για την παρουσίαση ενός μηνύματος που ενημερώνει τον χρήστη ότι το καλάθι είναι άδειο, μαζί με ένα κουμπί που τον καλεί να συνεχίσει τις αγορές του.

5.2.3.2 Component (Στοιχείο Διεπαφής)

```
<template>
 <UContainer class="mt-4">
  <UForm class="space-y-4":state="fields" autocomplete="on"
@submit="onSubmit">
   <UFormGroup
    v-for="{
     as,
     name,
     label,
     autocomplete = 'off',
     readonly = false,
     children = []
    } in schema.fields"
   :key="name"
   :label="label"
   :name="name"
    v-bind="fields[name].value"
   >
    <UInput
    :id="name"
    :as="as"
    :name="name"
     v-bind="fields[name].value"
    :autocomplete="autocomplete"
    :aria-readonly="readonly"
    :readonly="readonly"
     class="grid gap-1"
    >
     <div v-if="children">
      <DynamicFormChildren:children="children" />
     </div>
    </UInput>
   </UFormGroup>
   <UButton
    v-if="submitButton"
   :aria-busy="isSubmitting"
   :disabled="submitButtonDisabled"
    type="submit"
    >{{ buttonLabel }}</UButton
   >
   <UButton
    v-if="resetButton"
    class="ml-4"
    color="white"
    variant="outline"
    type="button"
    @click="resetForm()"
   >
    {{ resetLabel }}
   </UButton>
  </UForm>
```

```
 </UContainer>
</template>

<script lang="ts" setup>
import { z } from 'zod'
import { useForm } from 'vee-validate'
import type { PropType } from 'vue'
import { toTypedSchema } from '@vee-validate/zod'
import type { DynamicFormFields, DynamicFormSchema,
DynamicFormState } from '~/types/form'

// Define the UI configuration for Nuxt-UI
const nuxtUiConfig = (state: DynamicFormState) => {
 return {
  props: {
   error: state.errors[0]
  }
 }
}

// Define the props for the component
const props = defineProps({
 schema: {
  type: Object as PropType<DynamicFormSchema>,
  required: true
 },
 submitButton: {
  type: Boolean,
  default: true
 },
 resetButton: {
  type: Boolean,
  default: false
 },
 buttonLabel: {
  type: String,
  default: 'Submit'
 },
 resetLabel: {
  type: String,
  default: 'Reset'
 },
 disableSubmitUntilValid: {
  type: Boolean,
  default: false
 }
})

// Create an array of field names from the schema object
const schemaFieldNames = props.schema.fields.map((field) =>
field.name)

// Use schema.fields to generate a Zod schema object
const generatedSchema = z.object(
 Object.fromEntries(props.schema.fields.map((field) =>
[field.name, field.rules]))
```

```
)

// Convert the generated Zod schema object to a VeeValidate
compatible schema object
const validationSchema = toTypedSchema(generatedSchema)

// Define the form bindings and validation rules using
VeeValidate's useForm hook
const { defineComponentBinds, handleSubmit, resetForm, errors,
isSubmitting } = useForm({
 validationSchema
})

// Create an object of field bindings using defineComponentBinds
and nuxtUiConfig functions
function createFields(keys: string[]) {
 const fieldValues: DynamicFormFields<any, string, any> = {}
 keys.forEach((fieldName) => {
  fieldValues[fieldName] = defineComponentBinds(fieldName,
nuxtUiConfig)
 })
 return fieldValues
}
const fields = createFields(schemaFieldNames)

// Define the submit event emitter using defineEmits function
const emit = defineEmits(['submit'])

// Define the submit event handler using handleSubmit function and
emit function
const onSubmit = handleSubmit((values) => {
 emit('submit', values)
})

// Define the form state for Nuxt UI
const state = computed(() => {
 return Object.fromEntries(
  Object.entries(fields).map(([key, value]) => [key,
value.value.modelValue])
 )
})

// Define the submit button disabled state
const submitButtonDisabled = computedAsync(async () => {
 return await validationSchema
  .parse(state.value)
  .then((result) => {
   const liveResultValid = result.errors.length === 0
   return isSubmitting.value ||
    Object.keys(errors.value).length > 0 ||
    props.disableSubmitUntilValid
    ? !liveResultValid
   : false
  })
  .catch(() => {
   return true
```

```
  })
}, !!props.disableSubmitUntilValid)
</script>
```

Στο παράδειγμα αυτό, δημιουργήσαμε μια δυναμική φόρμα που χρησιμοποιεί το VeeValidate για την επικύρωση των δεδομένων που εισάγονται από τον χρήστη. Η φόρμα αυτή είναι δυναμική καθώς τα πεδία της δημιουργούνται αυτόματα βάσει ενός σχήματος που περνάμε ως prop.

Αρχικά, ορίσαμε τις απαραίτητες εξαρτήσεις και το UI configuration για το Nuxt-UI. Στη συνέχεια, ορίσαμε τα props για το component μας και δημιουργήσαμε το σχήμα επικύρωσης με τη χρήση της βιβλιοθήκης zod. Το σχήμα επικύρωσης μετατράπηκε σε ένα αντικείμενο συμβατό με το VeeValidate χρησιμοποιώντας τη συνάρτηση toTypedSchema.

Στη συνέχεια, χρησιμοποιήσαμε το hook useForm του VeeValidate για να ορίσουμε τα bindings και τους κανόνες επικύρωσης της φόρμας μας. Χρησιμοποιήσαμε επίσης τη συνάρτηση createFields για να δημιουργήσουμε τα bindings για τα πεδία της φόρμας μας.

Τέλος, ορίσαμε το event emitter για το submit event και τον χειριστή για το submit event που θα εκτελεί την επικύρωση της φόρμας και θα εκπέμπει το submit event με τα δεδομένα της φόρμας. Ορίσαμε επίσης την κατάσταση της φόρμας για το Nuxt UI και την κατάσταση για το κουμπί submit.

5.2.3.3 Layout (Διάταξη)

```
<script lang="ts" setup>
defineSlots<{
 default(props: {}): any
 main(props: {}): any
 header(props: {}): any
 footer(props: {}): any
 'app-before'(props: {}): any
 'app-after'(props: {}): any
}>()
</script>

<template>
 <div class="relative">
  <div id="app-before">
   <slot name="app-before" />
  </div>
  <slot name="header">
   <PageHeader>
    <PageNavbar />
   </PageHeader>
  </slot>
  <slot name="main">
   <Main class="relative flex-1 flex flex-col w-full h-full">
    <PageSection class="flex flex-col">
     <div class="flex-1 w-full flex flex-col">
      <slot />
     </div>
```

```
    </PageSection>
   </Main>
  </slot>
  <slot name="footer">
   <Footer />
  </slot>
  <div id="app-after">
   <slot name="app-after" />
  </div>
 </div>
</template>
```

Στο παράδειγμα αυτό, δημιουργήσαμε μια διάταξη που χρησιμοποιεί την τεχνική των slots για την προσθήκη δυναμικού περιεχομένου σε διάφορα μέρη της σελίδας. Τα slots μας είναι ευέλικτα και μας επιτρέπουν να τοποθετούμε διάφορα στοιχεία στη διάταξή μας με βάση τις ανάγκες της εφαρμογής μας.

Ξεκινήσαμε ορίζοντας τα slots που θα χρησιμοποιήσουμε στο component μας με τη βοήθεια της συνάρτησης defineSlots. Ορίσαμε τα slots "default", "main", "header", "footer", "app-before" και "app-after" για να καλύψουμε τις διάφορες περιοχές της διάταξής μας.

Στο template μας, έχουμε μια διάταξη που αποτελείται από διάφορα divs και slots. Το "app-before" και το "app-after" είναι slots που μπορούν να περιέχουν περιεχόμενο πριν και μετά από την κύρια διάταξη της εφαρμογής μας, αντίστοιχα.
Το slot "header" περιέχει τα στοιχεία PageHeader και PageNavbar, ενώ το slot "footer" περιέχει το στοιχείο Footer. Το slot "main" περιέχει το κύριο περιεχόμενο της σελίδας, και είναι ενσωματωμένο στο στοιχείο Main. Το κύριο περιεχόμενο της σελίδας περικλείεται στο στοιχείο PageSection, το οποίο προσθέτει επιπλέον διαμόρφωση και στυλ στο περιεχόμενο.

5.2.3.4 Composable (Σύνθετη Λειτουργία)

```
export const useText = () => {

 export function capitalize(str: string, allWords = false): string
{
   if (allWords) {
     return str
       .split(' ')
       .map((word) => word.charAt(0).toUpperCase() +
word.slice(1))
       .join(' ')
   }
   return str.charAt(0).toUpperCase() + str.slice(1)
 }

 export function contentShorten(
   content: string | null | undefined = '',
   from = 0,
   to = 200,
   suffix = '...'
 ): string {
   if (!content) return ''
```

```
  if (content.length < to) return content
  return content.substring(from, to) + suffix
}

export function contentShortenByWords(
  content: string,
  from = 0,
  to = 200,
  suffix = '...'
): string {
  const words = content.split(' ')
  if (words.length < to) return content
  return words.slice(from, to).join(' ') + suffix
}

export function cleanHtml(html: string): string {
  return html.replace(/<\/?[^>]+(>|$)/g, '')
}

return {
 contentShorten,
 contentShortenByWords,
 capitalize,
 cleanHtml
}
}
```

Στο παράδειγμα αυτό, δημιουργήσαμε μια σύνθετη λειτουργία με την ονομασία "useText", η οποία περιλαμβάνει μερικές χρήσιμες συναρτήσεις για τον χειρισμό και τη μορφοποίηση κειμένου.

Η συνάρτηση "capitalize" μετατρέπει το πρώτο γράμμα της παραδοσιακής συμβολοσειράς σε κεφαλαίο. Μπορούμε επίσης να καθορίσουμε εάν θέλουμε να κάνουμε κεφαλαίο το πρώτο γράμμα κάθε λέξης στη συμβολοσειρά. Η συνάρτηση "contentShorten" περικόπτει το κείμενο σε μια επιθυμητή διάρκεια και προσθέτει ένα επίθημα (π.χ. "...") στο τέλος, αν είναι απαραίτητο. Η συνάρτηση "contentShortenByWords" λειτουργεί παρόμοια με την "contentShorten", αλλά κόβει το κείμενο με βάση τον αριθμό των λέξεων αντί για τον αριθμό των χαρακτήρων. Η συνάρτηση "cleanHtml" αφαιρεί όλες τις ετικέτες HTML από μια συμβολοσειρά.

Με αυτές τις συναρτήσεις, μπορούμε να διαχειριστούμε και να μορφοποιήσουμε κείμενα πολύ πιο εύκολα και αποδοτικά στην εφαρμογή μας.

5.2.3.5 Middleware (Ενδιάμεσο Λογισμικό)

```
export default defineNuxtRouteMiddleware((to) => {
 const config = useRuntimeConfig()
 const publicConfig = config.public.auth

 if (to.path === publicConfig.redirect.login) {
  return
```

```
 }

 const { user } = useAuthSession()

 if (user.value) {
  return navigateTo(publicConfig.redirect.home)
 }
})
```

Σε αυτό το παράδειγμα, δημιουργήσαμε ένα ενδιάμεσο λογισμικό (middleware) που ελέγχει την κατάσταση σύνδεσης του χρήστη και ανακατευθύνει τον χρήστη σε διάφορες σελίδες ανάλογα με την κατάστασή του.

Πρώτα, λαμβάνουμε τη διαμόρφωση από το useRuntimeConfig και αποθηκεύουμε το δημόσιο μέρος της διαμόρφωσης στη μεταβλητή publicConfig. Αυτό περιλαμβάνει τις διευθύνσεις URL για τη σύνδεση και την αρχική σελίδα.

Ελέγχουμε εάν ο χρήστης είναι συνδεδεμένος με τη χρήση του useAuthSession. Εάν ο χρήστης είναι συνδεδεμένος, τότε τον ανακατευθύνουμε στην αρχική σελίδα. Εάν ο χρήστης δεν είναι συνδεδεμένος, τότε δεν κάνουμε τίποτα και επιτρέπουμε στον χρήστη να προχωρήσει στη σελίδα που επιθυμεί.

Με αυτόν τον τρόπο, εξασφαλίζουμε ότι οι χρήστες που είναι συνδεδεμένοι δεν μπορούν να προσπελάσουν σελίδες που μπορούν να κατευθυνθούν μόνο οι μη συνδεδεμένοι χρήστες όπως η σελίδες της εγγραφής (register) και σύνδεσης (login).

5.2.3.6 Module (Ενότητα)

```
import { addVitePlugin, defineNuxtModule } from '@nuxt/kit'
import MagicString from 'magic-string'

export default defineNuxtModule({
 meta: {
  name: '@groove/purge-comments'
 },
 setup() {
  addVitePlugin({
   name: 'purge-comments',
   enforce: 'pre',
   transform: (code: string, id: string) => {
    if (!id.endsWith('.vue') || !code.includes('<!--')) return

    const s = new MagicString(code)
    s.replace(/<!--(?:.*?)-->/gs, '')

    if (s.hasChanged()) {
     return {
      code: s.toString(),
      map: s.generateMap({ source: id, includeContent: true })
     }
    }
   }
```

```
  })
 }
})
```

Σε αυτήν την ενότητα, παρουσιάζουμε έναν ειδικό module που έχει σχεδιαστεί για το Nuxt framework. Ο κύριος στόχος αυτής της ενότητας είναι να εξαφανίσει τα σχόλια από τον κώδικα .vue κατά τη διάρκεια της διαδικασίας build με τη χρήση του Vite.

Αρχικά, εισάγουμε τις απαραίτητες βιβλιοθήκες και εργαλεία. Το addVitePlugin και defineNuxtModule προέρχονται από το @nuxt/kit, ενώ το MagicString είναι ένα εργαλείο που βοηθά στη διαχείριση και τροποποίηση των string, ιδιαίτερα όταν χρειάζεται να διατηρηθούν source maps.

Στη συνέχεια, καθορίζουμε το module μέσα στη defineNuxtModule. Το meta αντικείμενο παρέχει μεταδεδομένα για το module, όπως το όνομά του.

Στο setup, προσθέτουμε το Vite plugin με τη βοήθεια της addVitePlugin. Αυτό το plugin έχει το όνομα purge-comments και ο στόχος του είναι να εξαφανίσει τα σχόλια από τα αρχεία .vue. Εάν το αρχείο δεν έχει την επέκταση .vue ή δεν περιέχει σχόλια, τότε το plugin δεν κάνει τίποτα. Εάν υπάρχουν σχόλια, τότε τα αφαιρεί με τη χρήση του MagicString και ενημερώνει τον κώδικα καθώς και το αντίστοιχο source map.

Με αυτόν τον τρόπο, εξασφαλίζουμε ότι οι ενδιάμεσοι κώδικες που δημιουργούνται κατά τη διαδικασία build είναι καθαρότεροι και χωρίς περιττά σχόλια, βελτιστοποιώντας την απόδοση και το μέγεθος του τελικού κώδικα.

5.2.3.7 Plugin (Πρόσθετο)

```
import { GlobalEvents } from '~/events/global'

export default defineNuxtPlugin((nuxtApp) => {
 // called right before setting a new locale
 nuxtApp.hook(
  'i18n:beforeLocaleSwitch',
  ({ oldLocale, newLocale, initialSetup, context }) => {
   const bus =
useEventBus<string>(GlobalEvents.ON_BEFORE_LANGUAGE_SWITCH)
   bus.emit(GlobalEvents.ON_BEFORE_LANGUAGE_SWITCH, {
    oldLocale,
    newLocale,
    initialSetup,
    context
   })
  }
 )
 // called right after a new locale has been set
 nuxtApp.hook('i18n:localeSwitched', ({ oldLocale, newLocale }) =>
{
  const bus = useEventBus<string>(GlobalEvents.ON_LANGUAGE_UPDATED)
  bus.emit(GlobalEvents.ON_LANGUAGE_UPDATED, {
   oldLocale,
```

```
    newLocale
  })
 })
})
```

Σε αυτό το πρόσθετο, εστιάζουμε στον χειρισμό της αλλαγής της γλώσσας στην εφαρμογή μας, χρησιμοποιώντας το i18n framework για την υποστήριξη πολλαπλών γλωσσών.

Αρχικά, εισάγουμε το GlobalEvents από το αρχείο ~/events/global. Στη συνέχεια, καθορίζουμε το πρόσθετο με την χρήση της defineNuxtPlugin. Μέσα στη συνάρτηση του πρόσθετου, ορίζουμε δύο hooks του i18n framework.

Το πρώτο hook, i18n:beforeLocaleSwitch, καλείται λίγο πριν αλλάξει η γλώσσα. Μέσα σε αυτό το hook, χρησιμοποιούμε το useEventBus για να πάρουμε το event bus που αντιστοιχεί στο ON_BEFORE_LANGUAGE_SWITCH event και στη συνέχεια εκπέμπουμε το event μαζί με τα αντίστοιχα δεδομένα του hook.

Το δεύτερο hook, i18n:localeSwitched, καλείται λίγο μετά από την αλλαγή της γλώσσας. Αντίστοιχα, χρησιμοποιούμε το useEventBus για να πάρουμε το event bus που αντιστοιχεί στο ON_LANGUAGE_UPDATED event και στη συνέχεια εκπέμπουμε το event μαζί με τα αντίστοιχα δεδομένα του hook.

Με αυτόν τον τρόπο, καταφέρνουμε να χειριζόμαστε την αλλαγή της γλώσσας στην εφαρμογή μας με μια κομψή και αποτελεσματική λύση.

5.2.3.8 Store (Αποθήκη Κατάστασης)

```
export const useProductStore = defineStore('product', () => {
 const product = ref<Product | null>(null)
 const pending = ref<PendingRecord>(pendingFactory())
 const error = ref<ErrorRecord>(errorsFactory())

 async function fetchProduct(id: string | number) {
  const {
   data,
   error: productError,
   pending: productPending,
   refresh
  } = await useFetch<Product>(`/api/product/${id}`, {
   method: 'get'
  })
  product.value = data.value
  error.value.product = productError.value
  pending.value.product = productPending.value

  return {
   data,
   error: productError,
   pending: productPending,
   refresh
  }
```

```
 }

 async function createProduct(body: ProductCreateBody) {
  const {
   data,
   error: productError,
   pending: productPending,
   refresh
  } = await useFetch<Product>(`/api/products`, {
   method: 'post',
   body
  })
  product.value = data.value
  error.value.product = productError.value
  pending.value.product = productPending.value

  return {
   data,
   error: productError,
   pending: productPending,
   refresh
  }
 }

 async function updateProductHits(id: string | number) {
  const {
   error: productError,
   pending: productPending,
   refresh
  } = await useFetch(`/api/product/${id}/update-product-hits`, {
   method: 'post'
  })
  error.value.product = productError.value
  pending.value.product = productPending.value

  return {
   error: productError,
   pending: productPending,
   refresh
  }
 }

 return {
  product,
  pending,
  error,
  fetchProduct,
  createProduct,
  updateProductHits
 }
})
```

Στον παραπάνω κώδικα, έχουμε ένα store του Pinia που χρησιμοποιείται για τη διαχείριση της κατάστασης των προϊόντων στην εφαρμογή. Ορίζονται τρεις βασικές

μεταβλητές, product, pending, και error, που κρατούν την τρέχουσα κατάσταση του προϊόντος, τυχόν σφάλματα και την κατάσταση των εκκρεμοτήτων αντίστοιχα.

Επίσης, ορίζονται τρεις λειτουργίες, fetchProduct, createProduct, και updateProductHits, που χρησιμοποιούν την useFetch για να ανακτήσουν ή να ενημερώσουν τα δεδομένα των προϊόντων από το API. Αυτές οι λειτουργίες ενημερώνουν τις μεταβλητές product, error, και pending ανάλογα με το αποτέλεσμα του αιτήματος.

Στο τέλος, ορίζεται μια λειτουργία επιστροφής που επιστρέφει όλες τις μεταβλητές και τις λειτουργίες που έχουν οριστεί, ώστε να μπορούν να χρησιμοποιηθούν από άλλα μέρη της εφαρμογής.

5.2.3.9 Util (Εργαλείο)

```
export function hexToRgb(hex: string): { r: number; g: number; b:
number } {
 return {
  r: parseInt(hex.substring(1, 3), 16),
  g: parseInt(hex.substring(3, 5), 16),
  b: parseInt(hex.substring(5, 7), 16)
 }
}

export function rgbToHex(r: number, g: number, b: number): string
{
 return '#' + ((1 << 24) + (r << 16) + (g << 8) +
b).toString(16).slice(1)
}
```

Σε αυτό το εργαλείο, προσφέρουμε δύο χρήσιμες συναρτήσεις για την μετατροπή των χρωμάτων από την μορφή τους σε hex (δεκαεξαδικό) σε RGB (κόκκινο, πράσινο, μπλε) και αντίστροφα.

Η πρώτη συνάρτηση, hexToRgb, δέχεται ως είσοδο ένα string που αντιπροσωπεύει ένα χρώμα σε δεκαεξαδική μορφή (π.χ., #ff5733) και επιστρέφει ένα αντικείμενο με τα αντίστοιχα τρία στοιχεία r, g, b που αντιπροσωπεύουν τις τιμές των κόκκινου, πράσινου και μπλε χρώματος αντίστοιχα.

Η δεύτερη συνάρτηση, rgbToHex, δέχεται ως είσοδο τρεις αριθμούς που αντιπροσωπεύουν τις τιμές των κόκκινου, πράσινου και μπλε χρώματος αντίστοιχα, και επιστρέφει ένα string που αντιπροσωπεύει το χρώμα σε δεκαεξαδική μορφή.

Με αυτόν τον τρόπο, καθιστούμε ευκολότερο τον χειρισμό και την μετατροπή των χρωμάτων στην εφαρμογή μας, ενισχύοντας την ευελιξία και την ευκολία χρήσης του κώδικά μας.

5.2.4 Server

Από τον κατάλογο διακομιστή του Nuxt (server directory):

5.2.4.1 Api (Προγραμματιστικό Περιβάλλον Εφαρμογής)

```
export default defineWrappedResponseHandler(async (event: H3Event)
=> {
 const config = useRuntimeConfig()
 const params = parseParamsAs(event, ZodProductParams)
 const response = await
$api(`${config.public.apiBaseUrl}/product/${params.id}/`, event)
 return await parseDataAs(response, ZodProduct)
})
```

Στο παρακάτω παράδειγμα του κώδικά μας, παρουσιάζουμε μια συνάρτηση που χρησιμοποιεί το Nuxt API για να ανακτήσει στοιχεία για ένα συγκεκριμένο προϊόν από τον διακομιστή μας. Η συνάρτηση αυτή χρησιμοποιεί την μέθοδο defineWrappedResponseHandler για να καθορίσει μια λειτουργία ανταπόκρισης που θα εκτελεστεί όταν λάβουμε την αντίστοιχη αίτηση από τον πελάτη.

Πρώτα, χρησιμοποιούμε την useRuntimeConfig για να ανακτήσουμε τις ρυθμίσεις μας από τον διακομιστή. Στη συνέχεια, χρησιμοποιούμε την parseParamsAs για να αναλύσουμε τις παραμέτρους του αιτήματος και να επιβεβαιώσουμε ότι ανταποκρίνονται στο αναμενόμενο σχήμα των δεδομένων μας. Στη συνέχεια, χρησιμοποιούμε την $api για να κάνουμε την αίτηση στον διακομιστή μας και να λάβουμε την απόκριση.

Τέλος, χρησιμοποιούμε την parseDataAs για να αναλύσουμε τα δεδομένα από την απόκριση και να επιβεβαιώσουμε ότι ανταποκρίνονται στο αναμενόμενο σχήμα των δεδομένων μας. Αν όλα πάνε καλά, η συνάρτηση θα επιστρέψει τα δεδομένα του προϊόντος.

Με αυτόν τον τρόπο, διασφαλίζουμε ότι τα δεδομένα που λαμβάνουμε από τον διακομιστή μας είναι έγκυρα και σύμφωνα με τις προδιαγραφές μας, πριν τα παράσχουμε στον πελάτη μας.

5.2.4.2 Middleware (Ενδιάμεσο Λογισμικό)

```
import { useLogger } from '@nuxt/kit'

export default defineEventHandler((event) => {
 const requestUrl = getRequestURL(event)
 const logger = useLogger()

 if (process.env.NODE_ENV !== 'production') {
   logger.info(`New request: ${requestUrl}`)
 }
})
```

Στο παρακάτω παράδειγμα κώδικα, εμφανίζουμε ένα μεσολαβητικό λογισμικό (middleware) που χρησιμοποιεί το Nuxt API για να καταγράψει τις νέες αιτήσεις που λαμβάνονται από τον διακομιστή. Αυτό γίνεται με την χρήση της μεθόδου

defineEventHandler για να ορίσουμε έναν χειριστή συμβάντων που θα εκτελείται όταν λάβουμε ένα αίτημα.

Αρχικά, χρησιμοποιούμε την getRequestURL για να λάβουμε την διεύθυνση URL της αιτήσεως. Στη συνέχεια, χρησιμοποιούμε την useLogger για να λάβουμε έναν εγγραφέα συμβάντων.

Στον κώδικα μας, ελέγχουμε αν η εφαρμογή μας βρίσκεται σε περιβάλλον παραγωγής (production). Αν δεν βρισκόμαστε σε περιβάλλον παραγωγής, χρησιμοποιούμε τον εγγραφέα για να καταγράψουμε ένα μήνυμα που περιέχει τη διεύθυνση URL της νέας αιτήσεως.

Με αυτόν τον τρόπο, καταγράφουμε τις νέες αιτήσεις που λαμβάνουμε στον διακομιστή μας, επιτρέποντάς μας να έχουμε μια καλύτερη εποπτεία των αιτήσεων που διαχειριζόμαστε και να διαγνώσουμε πιο εύκολα τυχόν προβλήματα που μπορεί να προκύψουν.

5.2.4.3 Utils (Εργαλεία)

```
import type { EventHandler, EventHandlerRequest } from 'h3'

export const defineWrappedResponseHandler = <T extends
EventHandlerRequest, D>(
 handler: EventHandler<T, D>
): EventHandler<T, D> =>
 defineEventHandler<T>(async (event) => {
  const jwtAuth = getCookie(event, 'jwt_auth') || ''
  const cookie = getHeader(event, 'cookie') || ''

  if (jwtAuth) {
   appendResponseHeader(event, 'Authorization', 'Bearer ' +
jwtAuth)
  }
  appendResponseHeader(event, 'Cookie', cookie)

  try {
   // do something before the route handler
   const response = await handler(event)
   // do something after the route handler
   return response
  } catch (error) {
   await handleError(error)
  }
 })
```

Στο παρακάτω παράδειγμα κώδικα, έχουμε ορίσει ένα βοηθητικό εργαλείο που χρησιμοποιείται για τη δημιουργία ενός χειριστή αιτήσεων με τη χρήση της defineEventHandler. Το βοηθητικό εργαλείο defineWrappedResponseHandler λαμβάνει ως παράμετρο έναν χειριστή και επιστρέφει έναν νέο χειριστή που έχει τροποποιηθεί ώστε να εκτελεί κάποιες ενέργειες πριν και μετά την εκτέλεση του αρχικού χειριστή.

Συγκεκριμένα, το εργαλείο defineWrappedResponseHandler χρησιμοποιεί την getCookie για να λάβει το cookie jwt_auth από το αίτημα. Στη συνέχεια, χρησιμοποιεί την appendResponseHeader για να προσθέσει μια κεφαλίδα Authorization με την τιμή του jwt_auth στο αίτημα, αν αυτό υπάρχει.

Μετά την προσθήκη της κεφαλίδας, το εργαλείο καλεί τον αρχικό χειριστή και επιστρέφει την απόκριση που λαμβάνει από αυτόν. Εάν προκύψει κάποιο σφάλμα κατά την εκτέλεση του χειριστή, το εργαλείο καλεί την handleError για να χειριστεί το σφάλμα.

Με αυτόν τον τρόπο, το εργαλείο defineWrappedResponseHandler παρέχει έναν βολικό τρόπο για να προσθέσουμε κάποιες κοινές ενέργειες πριν και μετά την εκτέλεση ενός χειριστή, χωρίς να χρειάζεται να τροποποιήσουμε τον κώδικα του αρχικού χειριστή.

5.3 Media Stream Application (Nest.js):

5.3.1 Βασικές λειτουργίες

Ενα κομμάτι της πτυχιακής εργασίας είναι το microservice media stream το οποίο έχει δημιουργηθεί με τη χρήση του Nest.js.Οι βασικές λειτουργίες του Microservice περιλαμβάνουν την επεξεργασία των metadata της εικόνας, τη δυνατότητα αλλαγής των διαστάσεων της εικόνας "on the fly".

Επεξεργασία Metadata:
Το Microservice επεξεργάζεται τα παρακάτω metadata για κάθε εικόνα:

- **version**: Η έκδοση της εικόνας.
- **size**: Το μέγεθος της εικόνας.
- **format**: Το format της εικόνας (π.χ., jpg, png).
- **dateCreated**: Η ημερομηνία δημιουργίας της εικόνας.
- **privateTTL**: Ο χρόνος ζωής της εικόνας σε ιδιωτική μορφή.
- **publicTTL**: Ο χρόνος ζωής της εικόνας σε δημόσια μορφή.

Resize On The Fly:
Επιπλέον, το Microservice παρέχει τη δυνατότητα αλλαγής των διαστάσεων της εικόνας "on the fly" μέσω API endpoints.

Οι παράμετροι του API endpoint είναι οι εξής:

- **image**: Το όνομα της εικόνας.
- **width**: Το πλάτος της εικόνας.
- **height**: Το ύψος της εικόνας.
- **fit**: Ο τρόπος προσαρμογής της εικόνας.
- **position**: Η θέση της εικόνας.
- **background**: Το φόντο της εικόνας.
- **trimThreshold**: Το όριο αποκοπής της εικόνας.
- **format**: Το format της εικόνας.

5.3.2 Αποσπάσματα Κώδικα

Παρακάτω παρουσιάζονται αποσπάσματα από τον κώδικα του Microservice, τα οποία περιλαμβάνουν παραδείγματα από Controller, DTO, Exception, Job, Module, Operation και Rule.

5.3.3 Controller (Ο ρυθμιστής που διαχειρίζεται τις εισερχόμενες αιτήσεις)

```
@Get('static/images/:image/:width?/:height?/:fit?/:position?/:back
ground?/:trimThreshold?/:format?')
public async staticImage(
 @Param('image') image: string,
 @Param('width') width: number = null,
 @Param('height') height: number = null,
 @Param('fit') fit: FitOptions = FitOptions.contain,
 @Param('position') position = PositionOptions.entropy,
 @Param('background') background = BackgroundOptions.transparent,
 @Param('trimThreshold') trimThreshold = 5,
 @Param('format') format: SupportedResizeFormats =
SupportedResizeFormats.webp,
 @Res() res: Response
): Promise<void> {
 const djangoApiUrl = process.env.DJANGO_API_URL ||
'http://localhost:8000'
 const request = new CacheImageRequest({
  resourceTarget:
MediaStreamImageRESTController.resourceTargetPrepare(`${djangoApiUr
l}/static/images/${image}`),
  resizeOptions: new ResizeOptions({
   width,
   height,
   position,
   background,
   fit,
   trimThreshold,
   format
  })
 })
 await this.streamRequestedResource(request, res)
}
```

Στον παραπάνω κώδικα, ο controller ανταποκρίνεται στο API endpoint static/images/:image/:width?/:height?/:fit?/:position?/:background?/:trimThreshold?/:format? και λαμβάνει ως παραμέτρους:

- **image**: το όνομα της εικόνας.
- **width**: το πλάτος για το resizing (προαιρετικό).
- **height**: το ύψος για το resizing (προαιρετικό).

- **fit**: τον τρόπο προσαρμογής της εικόνας στις διαστάσεις (προαιρετικό, προκαθορισμένο contain).
- **position**: τη θέση της εικόνας (προαιρετικό, προκαθορισμένο entropy).
- **background**: το φόντο της εικόνας (προαιρετικό, προκαθορισμένο transparent)
- **trimThreshold**: το κατώφλι για το trimming (προαιρετικό, προκαθορισμένο 5).
- **format**: το format της εικόνας (προαιρετικό, προκαθορισμένο webp).

Με βάση αυτές τις παραμέτρους, κατασκευάζεται ένα αντικείμενο CacheImageRequest το οποίο περιέχει το URL της εικόνας και τις επιλογές για το resizing. Στη συνέχεια, καλείται η λειτουργία streamRequestedResource για να εκτελέσει την ενέργεια του resizing της εικόνας και να επιστρέψει το αποτέλεσμα στον χρήστη.

Επομένως, ο controller αναλαμβάνει τον συντονισμό των εισερχομένων αιτήσεων, την εξαγωγή και την επεξεργασία των παραμέτρων, καθώς και την κλήση των αντίστοιχων λειτουργιών για την εκτέλεση της ζητούμενης ενέργειας.

```
private async streamRequestedResource(request: CacheImageRequest,
res: Response): Promise<void> {
 await this.cacheImageResourceOperation.setup(request)
 if (this.cacheImageResourceOperation.resourceExists) {
  const headers = this.cacheImageResourceOperation.getHeaders
  res = MediaStreamImageRESTController.addHeadersToRequest(res,
headers)
  const stream =
createReadStream(this.cacheImageResourceOperation.getResourcePath)
.pipe(res)
  try {
   await new Promise((resolve, reject) => {
    stream.on('finish', () => resolve)
    stream.on('error', () => reject)
   })
  } catch (e) {
   // ignore failed stream to client for now
   this.logger.error(e)
  } finally {
   await this.cacheImageResourceOperation.execute()
  }
 } else {
  try {
   await this.cacheImageResourceOperation.execute()
   const headers = this.cacheImageResourceOperation.getHeaders
   res = MediaStreamImageRESTController.addHeadersToRequest(res,
headers)

createReadStream(this.cacheImageResourceOperation.getResourcePath)
.pipe(res)
  } catch (e) {
   this.logger.error(e)
```

```
    res.status(404).send()
   }
  }
}
```

Η συγκεκριμένη λειτουργία streamRequestedResource αναλαμβάνει να κατευθύνει το stream της εικόνας προς τον client. Αρχικά, καλεί τη λειτουργία setup του cacheImageResourceOperation για να προετοιμάσει τα απαιτούμενα δεδομένα. Στη συνέχεια, ελέγχει εάν το αρχείο υπάρχει ήδη στον cache, και αν ναι, προσθέτει τα απαιτούμενα headers και κατευθύνει το stream της εικόνας προς τον client. Εάν το αρχείο δεν υπάρχει στον cache, εκτελεί την απαιτούμενη λειτουργία για να το δημιουργήσει, προσθέτει τα headers και κατευθύνει το stream της εικόνας προς τον client.

5.3.4 DTO (Data Transfer Object, Το αντικείμενο που μεταφέρει δεδομένα μεταξύ των επιπέδων της εφαρμογής)

```
export default class ResourceMetaData {
 version: number
 size: string
 format: string
 dateCreated: number
 privateTTL: number
 publicTTL: number

 constructor(data?: Partial<ResourceMetaData>) {
  if (!data.version) this.version = resourceMetaVersion
  if (!data.publicTTL) this.publicTTL = defaultPublicTTL
  if (!data.privateTTL) this.privateTTL = defaultPrivateTTL
  Object.assign(this, data)
 }
}
```

Στον παραπάνω κώδικα, η κλάση ResourceMetaData είναι ένα DTO που χρησιμοποιείται για τη μεταφορά δεδομένων σχετικών με τα μεταδεδομένα μιας πόρου. Περιέχει τα πεδία version, size, format, dateCreated, privateTTL, και publicTTL, τα οποία αποθηκεύουν διάφορες πληροφορίες για τον πόρο. Η μέθοδος constructor δέχεται ένα αντικείμενο data και χρησιμοποιεί τη συνάρτηση Object.assign για να αντιγράψει τα δεδομένα από το αντικείμενο data στο νέο αντικείμενο ResourceMetaData. Επίσης, εάν δεν υπάρχουν συγκεκριμένα πεδία στο αντικείμενο data, τότε χρησιμοποιεί τις προεπιλεγμένες τιμές resourceMetaVersion, defaultPublicTTL, και defaultPrivateTTL.

5.3.5 Exception (Η κλάση που διαχειρίζεται τις εξαιρέσεις)

```
export default class RequestedResizeTargetTooLargeException
extends Error {
 constructor(resizeRequest: ResizeOptions, allowedPixelCount:
number) {
  super(
```

```
    `Requested resize target
(${resizeRequest.width}x${resizeRequest.height}) exceeded maximum
allowed size of ${allowedPixelCount} total pixels`
   )
  }
}
```

Στον παραπάνω κώδικα, η κλάση RequestedResizeTargetTooLargeException είναι μία εξαίρεση που εκτείνεται από την κλάση Error της JavaScript. Χρησιμοποιείται για τον χειρισμό των καταστάσεων όπου ένας αιτούμενος στόχος αλλαγής μεγέθους της εικόνας είναι μεγαλύτερος από το επιτρεπόμενο μέγιστο μέγεθος. Η μέθοδος constructor δέχεται ως παραμέτρους ένα αντικείμενο ResizeOptions και έναν αριθμό allowedPixelCount, και καλεί τη μέθοδο super για να δημιουργήσει ένα μήνυμα λάθους που περιγράφει το πρόβλημα.

5.3.6 Job (Η δουλειά που πρέπει να εκτελεστεί, συχνά σε background)

```
@Injectable({ scope: Scope.REQUEST })
export default class FetchResourceResponseJob {
 private readonly logger = new
Logger(FetchResourceResponseJob.name)
 constructor(private readonly httpService: HttpService) {}

 async handle(request: CacheImageRequest): Promise<AxiosResponse>
{
  try {
   return await this.httpService.axiosRef({
    url: request.resourceTarget,
    method: 'GET',
    responseType: 'stream'
   })
  } catch (error) {
   // Return a 404 Bad Request response
     this.logger.error(error)
   return {
    status: 404,
    statusText: 'Bad Request',
    headers: {},
    config: error.config,
    data: null
   }
  }
 }
}
```

Στον παραπάνω κώδικα, η κλάση FetchResourceResponseJob είναι ένας Job που χειρίζεται το κατέβασμα μιας απόκρισης πόρου από έναν απομακρυσμένο διακομιστή με την χρήση του HttpService. Η μέθοδος handle δέχεται ένα αντικείμενο CacheImageRequest και χρησιμοποιεί την axiosRef του HttpService για να κάνει μια αίτηση GET στον διακομιστή. Αν η αίτηση αποτύχει, η μέθοδος handle επιστρέφει μια απόκριση 404 Bad Request.

5.3.7 Module (Το επίπεδο που ορίζει τις εξαρτήσεις και τις λειτουργίες της εφαρμογής)

```
const controllers = [MediaStreamImageRESTController]

const operations = [CacheImageResourceOperation]

const jobs = [
 GenerateResourceIdentityFromRequestJob,
 FetchResourceResponseJob,
 StoreResourceResponseToFileJob,
 WebpImageManipulationJob
]

const rules = [ValidateCacheImageRequestRule,
ValidateCacheImageRequestResizeTargetRule]

@Module({
 imports: [HttpModule],
 controllers,
 providers: [...jobs, ...rules, ...operations]
})
export default class MediaStreamModule {}
```

Στον παραπάνω κώδικα, η κλάση MediaStreamModule είναι ένας Module που ορίζει τον τρόπο που τα διάφορα στοιχεία της εφαρμογής συνδέονται μεταξύ τους.

Ειδικότερα:

- Η ιδιότητα controllers ορίζει τους ελεγκτές που χρησιμοποιούνται στο module, στην περίπτωση αυτή το MediaStreamImageRESTController.
- Η ιδιότητα operations ορίζει τις λειτουργίες που χρησιμοποιούνται στο module, στην περίπτωση αυτή το CacheImageResourceOperation.
- Η ιδιότητα jobs ορίζει τις δουλειές που εκτελούνται στο module, στην περίπτωση αυτή τις GenerateResourceIdentityFromRequestJob, FetchResourceResponseJob, StoreResourceResponseToFileJob, WebpImageManipulationJob.
- Η ιδιότητα rules ορίζει τους κανόνες που εφαρμόζονται στο module, στην περίπτωση αυτή τους ValidateCacheImageRequestRule, ValidateCacheImageRequestResizeTargetRule.
- Η διακόσμηση @Module χρησιμοποιείται για να ορίσει τις εξαρτήσεις του module, στην περίπτωση αυτή το HttpModule, καθώς και τους ελεγκτές, τους παροχείς, και άλλα στοιχεία που είναι απαραίτητα για την λειτουργία του module.

5.3.8 Operation (Η λειτουργία που πρέπει να εκτελεστεί)

```
public async execute(): Promise<void> {
 if (this.resourceExists) {
```

```
  return
 }

 const response = await
this.fetchResourceResponseJob.handle(this.request)
 await
this.storeResourceResponseToFileJob.handle(this.request.resourceTa
rget, this.getResourceTempPath, response)
 const manipulationResult = await
this.webpImageManipulationJob.handle(
  this.getResourceTempPath,
  this.getResourcePath,
  this.request.resizeOptions
 )

 const resourceMetaDataOptions = {
  size: manipulationResult.size,
  format: manipulationResult.format,
  p: this.request.ttl,
  dateCreated: Date.now(),
  publicTTL: 24 * 60 * 60 * 1000
 } as unknown as ResourceMetaData
 if (this.request.ttl) {
  resourceMetaDataOptions['privateTTL'] = this.request.ttl
 }
 this.metaData = new ResourceMetaData(resourceMetaDataOptions)

 writeFileSync(this.getResourceMetaPath,
JSON.stringify(this.metaData))
 unlink(this.getResourceTempPath, (err) => {
  if (null !== err) {
      this.logger.error(err)
    }
 })
}
```

Στον παραπάνω κώδικα, η μέθοδος execute εκτελεί τις ακόλουθες ενέργειες:

- Ελέγχει αν το πόρος υπάρχει ήδη (this.resourceExists). Αν ναι, τότε δεν εκτελείται καμία περαιτέρω ενέργεια.
- Χρησιμοποιεί το fetchResourceResponseJob για να ανακτήσει την απόκριση του πόρου. Χρησιμοποιεί το storeResourceResponseToFileJob για να αποθηκεύσει την απόκριση του πόρου σε ένα αρχείο.
- Χρησιμοποιεί το webpImageManipulationJob για να επεξεργαστεί την εικόνα και να την αποθηκεύσει σε μορφή WebP.
- Δημιουργεί ένα νέο ResourceMetaData αντικείμενο με τα κατάλληλα δεδομένα.
- Αποθηκεύει τα μεταδεδομένα του πόρου σε ένα αρχείο JSON.
- Διαγράφει το προσωρινό αρχείο της εικόνας.

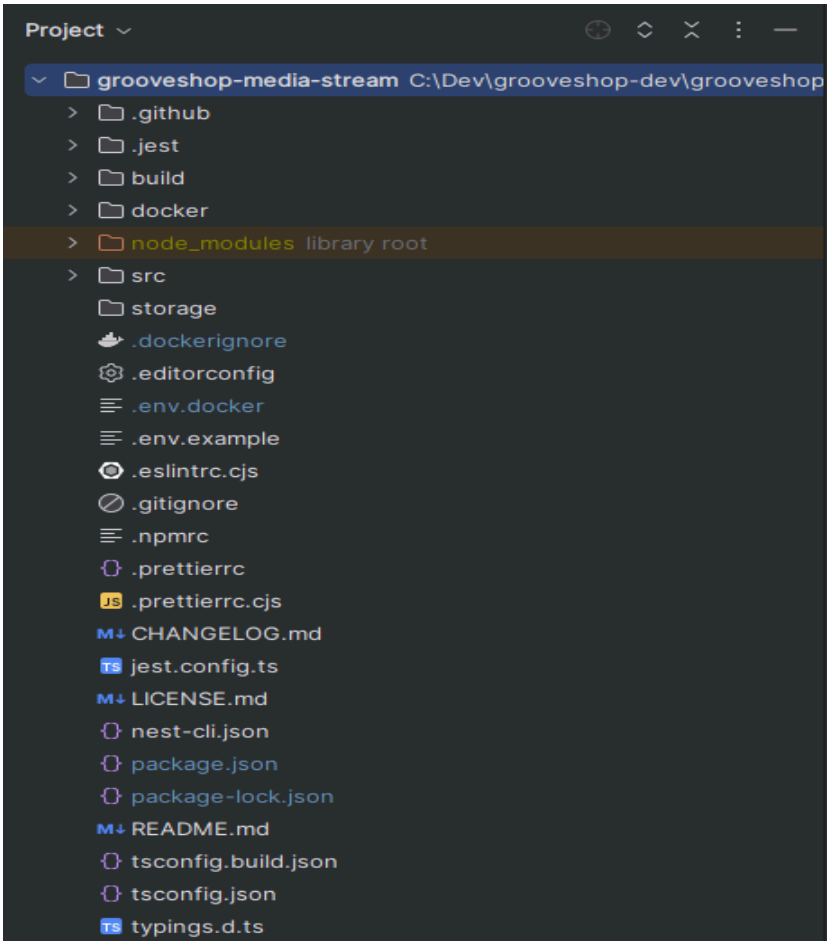
Κάθε βήμα είναι μια ξεχωριστή υποδιεργασία που διαχειρίζεται ένα συγκεκριμένο κομμάτι της συνολικής εργασίας, και το Operation τα συνδέει όλα μαζί για να επιτευχθεί ο τελικός στόχος.

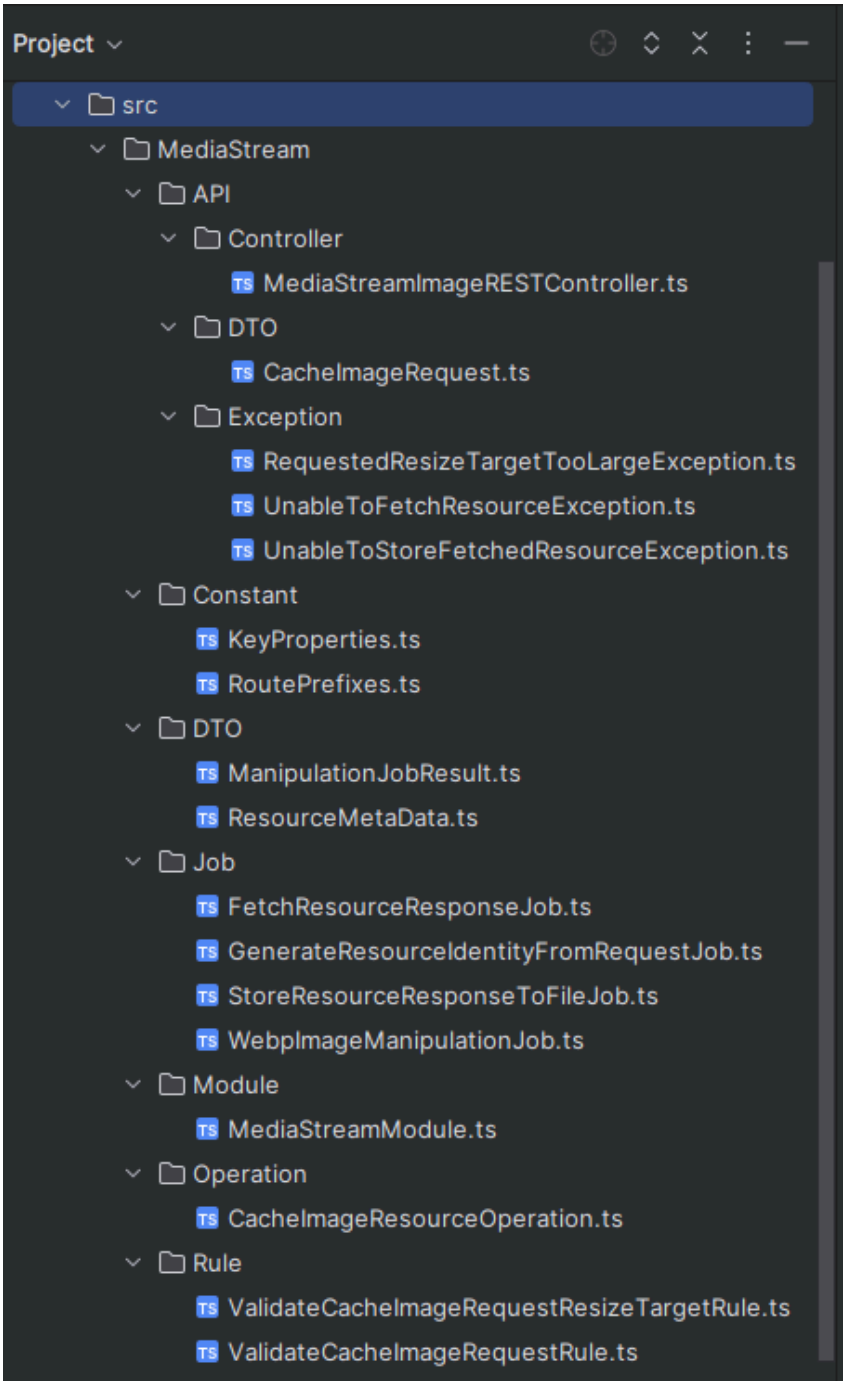
5.3.9 Rule (Ο κανόνας που πρέπει να ακολουθηθεί)

```
@Injectable({ scope: Scope.REQUEST })
export default class ValidateCacheImageRequestResizeTargetRule {
 allowedPixelCount = 2048 * 2048 //2K Squared

 request: CacheImageRequest = null

 public async setup(request: CacheImageRequest): Promise<void> {
  this.request = request
 }

 public async apply(): Promise<void> {
  const pixelCount = this.request.resizeOptions.width +
this.request.resizeOptions.height
  if (pixelCount > this.allowedPixelCount) {
   throw new
RequestedResizeTargetTooLargeException(this.request.resizeOptions,
this.allowedPixelCount)
  }
 }
}
```

Στον παραπάνω κώδικα ValidateCacheImageRequestResizeTargetRule, ο κανόνας ελέγχει αν η αίτηση για αλλαγή μεγέθους μιας εικόνας υπερβαίνει έναν καθορισμένο αριθμό pixel. Αν το μέγεθος της εικόνας που ζητείται για αλλαγή μεγέθους είναι μεγαλύτερο από το επιτρεπόμενο, τότε η λογική της εφαρμογής πετάει μια εξαίρεση RequestedResizeTargetTooLargeException για να ενημερώσει ότι το αίτημα δεν μπορεί να εκτελεστεί λόγω περιορισμού του μεγέθους.

5.3.8 Δομή φακέλου


Project
grooveshop-media-stream C:\Dev\grooveshop-dev\grooveshop
.github
.jest
build
docker
node_modules library root
src
storage
.dockerignore
.editorconfig
.env.docker
.env.example
.eslintrc.cjs
.gitignore
.npmrc
.prettierrc
.prettierrc.cjs
CHANGELOG.md
jest.config.ts
LICENSE.md
nest-cli.json
package.json
package-lock.json
README.md
tsconfig.build.json
tsconfig.json
typings.d.ts



Συμπεράσματα

Η εφαρμογή GrooveShop σχεδιάστηκε για να βοηθήσει στη διαχείριση μιας αποθήκης, καθώς και στη δημιουργία ενός εύχρηστου ηλεκτρονικού καταστήματος. Για να το κάνουμε αυτό, χρησιμοποιήσαμε το Django, το οποίο μας βοήθησε πολύ. Η τεκμηρίωση (Documentation) μας παρείχε όλες τις πληροφορίες που χρειαζόμασταν για να το χρησιμοποιήσουμε σωστά. Τα σχέδιά του είναι κατανοητά και εύκολα στην εφαρμογή. Πολλά μέρη της εφαρμογής, όπως το σύστημα ελέγχου ταυτότητας και ο χειρισμός του κωδικού πρόσβασης, ήταν έτοιμα για χρήση. Με αρκετά εύκολες οδηγίες, συνδέσαμε το πρόσθετο Django Rest στο Django, χρησιμοποιώντας τη λογική του για επικοινωνία API με τη βάση δεδομένων. Τέλος, προσθέσαμε το πακέτο διαχείρισης Django admin για να κάνουμε την εφαρμογή διαχείρισης αποθήκης πιο ολοκληρωμένη.

Η Vue διαθέτει επίσης εξαιρετική τεκμηρίωση (Documentation), συμπεριλαμβανομένου ενός μικρού σεμιναρίου (Tutorial). Η λογική σχεδιασμού συστατικών ήταν πολύ χρήσιμη και συνέβαλε σημαντικά ως άσκηση για την απόκτηση των απαραίτητων γνώσεων σε αυτόν τον τομέα. Ένα άλλο πλεονέκτημα είναι ότι ολόκληρη η διεπαφή μπορεί να αναλυθεί σε μικρότερα κομμάτια χωρίς να χρειάζεται να επεξεργαστεί τα κομμάτια, κάτι που δίνει τη δυνατότητα επαναχρησιμοποίησής τους, χωρίς ταυτόχρονα να γίνουν αλλαγές στην εφαρμογή. Η Vue δεν αποτέλεσε ποτέ πρόβλημα κατά τη διάρκεια της κατασκευής της εφαρμογής, αλλά μάλλον ενίσχυσε την ικανότητά μας να δημιουργούμε καλύτερες διεπαφές χρήστη (User Interface). Η Vue χρησιμοποιήθηκε για τη δημιουργία της διεπαφής για τα ηλεκτρονικά κατάστημα ηλεκτρονικών ειδών και τον πελάτη, δηλαδή το UI του ηλεκτρονικού καταστήματος.

Για τη βάση δεδομένων χρησιμοποιήσαμε την PostgreSQL η οποία είναι ένα ισχυρό, αξιόπιστο και ασφαλές σύστημα διαχείρισης βάσεων δεδομένων ανοικτού κώδικα που μπορεί να διαχειριστεί μεγάλα σύνολα δεδομένων και υποστηρίζει ένα ευρύ φάσμα τύπων δεδομένων και γλωσσών προγραμματισμού. Είναι οικονομικά αποδοτικό και διαθέτει μια μεγάλη κοινότητα προγραμματιστών και χρηστών που συμβάλλουν στην ανάπτυξη και την υποστήριξή του.

Θεωρούμε επίσης μείζον να αναφέρουμε δύο εργαλεία που μας βοήθησαν στον εντοπισμό σφαλμάτων στην εφαρμογή.

Το πρώτο είναι το Nuxt Dev Tools, στην πορεία ανάπτυξης της εφαρμογής μας, ανακαλύψαμε ότι τα Nuxt Dev Tools αποτελούν έναν πολύτιμο σύμμαχο. Τα εργαλεία αυτά προσφέρουν μια σειρά από λειτουργίες που επιτρέπουν την αποτελεσματική ανίχνευση και επίλυση προβλημάτων, καθώς και την βελτίωση της απόδοσης και της ποιότητας του κώδικα. Παρακάτω αναφέρουμε μερικά από τα κύρια οφέλη που αποκομίσαμε από τη χρήση των Nuxt Dev Tools:

1. **Ανίχνευση και Επίλυση Προβλημάτων**: Τα Nuxt Dev Tools μας παρείχαν τη δυνατότητα να εντοπίζουμε προβλήματα στον κώδικά μας σε πραγματικό χρόνο, προσφέροντας λεπτομερείς πληροφορίες σχετικά με τα λάθη και τις προειδοποιήσεις.

2. **Βελτίωση Απόδοσης**: Μέσω των εργαλείων ανάλυσης που περιλαμβάνονται, καταφέραμε να εντοπίσουμε και να βελτιώσουμε τμήματα του κώδικα που επηρέαζαν αρνητικά την απόδοση της εφαρμογής.

3. **Βελτίωση Ποιότητας Κώδικα**: Τα Nuxt Dev Tools μας βοήθησαν να διατηρήσουμε υψηλά επίπεδα ποιότητας κώδικα, αποκαλύπτοντας πρακτικές που θα μπορούσαν να οδηγήσουν σε προβλήματα στο μέλλον.

4. **Εξοικονόμηση Χρόνου**: Η αυτοματοποίηση πολλών διαδικασιών ελέγχου και ανάλυσης μας επέτρεψε να εξοικονομήσουμε σημαντικό χρόνο, επιτρέποντάς μας να επικεντρωθούμε στην υλοποίηση νέων χαρακτηριστικών.

5. **Ευελιξία και Ευκολία Χρήσης**: Τα Nuxt Dev Tools είναι σχεδιασμένα με τέτοιο τρόπο που επιτρέπουν εύκολη προσαρμογή και επέκταση, διευκολύνοντας την ένταξή τους στην υφιστάμενη ροή εργασίας μας.

Συνολικά, τα Nuxt Dev Tools αποδείχθηκαν αναντικατάστατα στην ενίσχυση της διαδικασίας ανάπτυξης, προσφέροντάς μας μια ισχυρή σουίτα εργαλείων για την ανίχνευση και επίλυση προβλημάτων, τη βελτίωση της απόδοσης και της ποιότητας του κώδικα, και την εξοικονόμηση πολύτιμου χρόνου κατά τη διάρκεια της ανάπτυξης της εφαρμογής μας.

Το δεύτερο είναι το Postman το οποίο είναι ένα δημοφιλές εργαλείο που χρησιμοποιείται από προγραμματιστές για δοκιμές και ανάπτυξη API. Το Postman φάνηκε ιδιαίτερα χρήσιμο για τους παρακάτω λόγους:

1. **Απλοποιεί τις δοκιμές API**: Το Postman παρέχει ένα εύχρηστο γραφικό περιβάλλον για την υποβολή αιτήσεων API, το οποίο διευκολύνει τους προγραμματιστές να δοκιμάζουν τα API τους χωρίς να γράφουν κώδικα.

2. **Εξοικονομεί χρόνο**: Με το Postman, οι προγραμματιστές μπορούν να αυτοματοποιήσουν επαναλαμβανόμενες δοκιμές API και να τις αποθηκεύσουν ως συλλογές, οι οποίες μπορούν εύκολα να μοιραστούν με άλλα μέλη της ομάδας. Αυτό εξοικονομεί χρόνο και διασφαλίζει τη συνέπεια στις δοκιμές.

3. **Υποστηρίζει διάφορα πρωτόκολλα**: Το Postman υποστηρίζει διάφορα πρωτόκολλα, συμπεριλαμβανομένων των REST, SOAP και GraphQL, καθιστώντας το ένα ευέλικτο εργαλείο για την ανάπτυξη API.

4. **Προσφέρει χαρακτηριστικά συνεργασίας**: Το Postman επιτρέπει στους προγραμματιστές να συνεργάζονται για την ανάπτυξη και τη δοκιμή API, μοιράζοντας συλλογές, περιβάλλοντα και τεκμηρίωση με τα μέλη της ομάδας.

5. **Παρέχει εργαλεία εντοπισμού σφαλμάτων**: Το Postman παρέχει εργαλεία εντοπισμού σφαλμάτων, όπως αρχεία καταγραφής κονσόλας και παρακολούθηση σφαλμάτων, που βοηθούν τους προγραμματιστές να εντοπίζουν και να διορθώνουν προβλήματα στα API τους.

6. **Υποστηρίζει μεταβλητές περιβάλλοντος**: Το Postman επιτρέπει στους προγραμματιστές να ορίζουν μεταβλητές περιβάλλοντος που μπορούν να χρησιμοποιηθούν σε διαφορετικά αιτήματα και συλλογές, διευκολύνοντας τη διαχείριση των ρυθμίσεων και των δοκιμών API.

Συνολικά, το Postman είναι ένα χρήσιμο εργαλείο για την ανάπτυξη και τη δοκιμή API που απλοποιεί τη διαδικασία δοκιμών, εξοικονομεί χρόνο και υποστηρίζει τη

συνεργασία και την αποσφαλμάτωση. Χρησιμοποιείται ευρέως από προγραμματιστές και διαθέτει μια μεγάλη κοινότητα χρηστών που συμβάλλουν στην ανάπτυξη και την υποστήριξή του.

Βελτιώσεις και Επιπλέον Προσθήκες

Η εφαρμογή αυτοματοποιημένων ειδοποιήσεων push θα απαιτούσε σημαντική ανάπτυξη και ενσωμάτωση με το backend της υπάρχουσας εφαρμογής. Οι ειδοποιήσεις push θα μπορούσαν να χρησιμοποιηθούν για την ενημέρωση των χρηστών σχετικά με σημαντικές ενημερώσεις, όπως νέα χαρακτηριστικά ή αλλαγές στη διεπαφή χρήστη. Επιπλέον, οι ειδοποιήσεις push θα μπορούσαν να χρησιμοποιηθούν για να ειδοποιούν τους χρήστες για τυχόν προβλήματα ή σφάλματα στην εφαρμογή, γεγονός που θα μπορούσε να συμβάλει στη βελτίωση της εμπειρίας του χρήστη και στη μείωση της απογοήτευσης.

Μια άλλη πιθανή λειτουργία που θα μπορούσε να προστεθεί στην εφαρμογή είναι η δυνατότητα προσαρμογής της διεπαφής χρήστη. Ενώ η τρέχουσα διεπαφή χρήστη είναι φιλική προς το χρήστη και εύκολη στην πλοήγηση, ορισμένοι χρήστες μπορεί να προτιμούν μια διαφορετική διάταξη ή ένα διαφορετικό χρωματικό σχήμα. Επιτρέποντας στους χρήστες να προσαρμόσουν τη διεπαφή στις προτιμήσεις τους, η εφαρμογή θα μπορούσε να γίνει ακόμη πιο εξατομικευμένη και ελκυστική σε ένα ευρύτερο φάσμα χρηστών.

Επιπλέον, η ενσωμάτωση λειτουργιών κοινής χρήσης κοινωνικών μέσων θα μπορούσε επίσης να αποτελέσει μια χρήσιμη προσθήκη στην εφαρμογή. Αυτό θα επέτρεπε στους χρήστες να μοιράζονται την πρόοδο, τα επιτεύγματα και τις εμπειρίες τους σε διάφορες πλατφόρμες κοινωνικής δικτύωσης, αυξάνοντας έτσι τη δέσμευση και την αναγνωρισιμότητα της εφαρμογής.

Εν κατακλείδι είναι σαφές οι λειτουργίες της εφαρμογής, όπως η διαχείριση αποθεμάτων, η παρακολούθηση παραγγελιών και η αλληλεπίδραση με τους πελάτες, είναι απαραίτητες για την ομαλή λειτουργία κάθε αποθήκης. Η λειτουργία ηλεκτρονικού καταστήματος είναι μια σπουδαία προσθήκη που επιτρέπει στους πελάτες να έχουν πρόσβαση σε όλες τις απαραίτητες πληροφορίες σχετικά με τα προϊόντα και τις τιμές τους, γεγονός που οδηγεί σε καλύτερη επικοινωνία και εμπιστοσύνη μεταξύ των πελατών και των αποθηκών.

Ωστόσο, υπάρχουν πάντα περιθώρια βελτίωσης. Για παράδειγμα, η διεπαφή χρήστη θα μπορούσε να είναι πιο διαισθητική και φιλική προς τον χρήστη, ώστε να ελαχιστοποιηθεί η καμπύλη εκμάθησης για τους νέους χρήστες. Επιπλέον, η εφαρμογή θα μπορούσε να επωφεληθεί από χαρακτηριστικά όπως η παρακολούθηση αποθεμάτων σε πραγματικό χρόνο και η αυτοματοποιημένη επεξεργασία παραγγελιών για τον περαιτέρω εξορθολογισμό της ροής εργασίας.

Ως δημιουργοί αυτής της εφαρμογής, αποκτήσαμε πολύτιμη εμπειρία και γνώσεις σχετικά με τις διαδικτυακές εφαρμογές και τα εργαλεία σχεδιασμού. Η διαδικασία δημιουργίας αυτής της εφαρμογής μας δίδαξε τη σημασία του σχεδιασμού με επίκεντρο τον χρήστη και την ανάγκη συνεχούς επανάληψης και βελτίωσης με βάση την ανατροφοδότηση των χρηστών.

Συνολικά, αυτή η εφαρμογή έχει τη δυνατότητα να φέρει επανάσταση στον κλάδο των αποθηκών και να βελτιώσει τον τρόπο λειτουργίας των αποθηκών. Με περαιτέρω ανάπτυξη και βελτιώσεις, θα μπορούσε να γίνει ένα απαραίτητο εργαλείο τόσο για τις αποθήκες όσο και για τους πελάτες τους.

Βιβλιογραφία, Αναφορές και Διαδικτυακές Πηγές

1. Django Software Foundation. (n.d.). Django. [Διαδίκτυο] Διαθέσιμο στο https://www.djangoproject.com [Πρόσβαση 12 Ιανουαρίου 2024].

2. Django Software Foundation. (2021). Django documentation (Version 4.1).[Διαδίκτυο] Διαθέσιμο στο https://docs.djangoproject.com/en/4.1 [Πρόσβαση 12 Ιανουαρίου 2024].

3. Wikipedia contributors. (n.d.). Django (web framework). In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/Django_(web_framework) [Πρόσβαση 12 Ιανουαρίου 2024].

4. Wikipedia contributors. (n.d.). Software framework. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/Software_framework [Πρόσβαση 14 Ιανουαρίου 2024].

5. Python Software Foundation. (n.d.). Python documentation. [Διαδίκτυο] Διαθέσιμο στο https://www.python.org/doc [Πρόσβαση 14 Ιανουαρίου 2024].

6. Wikipedia contributors. (n.d.). Python (programming language). In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/Python_(programming_language) [Πρόσβαση 14 Ιανουαρίου 2024].

7. Wikipedia contributors. (n.d.). JavaScript. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://el.wikipedia.org/wiki/JavaScript [Πρόσβαση 14 Ιανουαρίου 2024].

8. TypeScript. (n.d.). TypeScript. [Διαδίκτυο] Διαθέσιμο στο https://www.typescriptlang.org [Πρόσβαση 17 Ιανουαρίου 2024].

9. Wikipedia contributors. (n.d.). TypeScript. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/TypeScript [Πρόσβαση 17 Ιανουαρίου 2024].

10. W3Schools. (n.d.). HTML tutorial. [Διαδίκτυο] Διαθέσιμο στο https://www.w3schools.com/html [Πρόσβαση 17 Ιανουαρίου 2024].

11. W3Schools. (n.d.). CSS Tutorial. [Διαδίκτυο] Διαθέσιμο στο https://www.w3schools.com/css [Πρόσβαση 17 Ιανουαρίου 2024].

12. Wikipedia contributors. (n.d.). Cascading Style Sheets [CSS]. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://el.wikipedia.org/wiki/CSS [Πρόσβαση 17 Ιανουαρίου 2024].

13. Sass Lang. (n.d.). Sass: Syntactically Awesome Style Sheets - Guide. [Διαδίκτυο] Διαθέσιμο στο https://sass-lang.com/guide [Πρόσβαση 17 Ιανουαρίου 2024].

14. Wikipedia contributors. (n.d.). SCSS (Sassy CSS). In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/SCSS [Πρόσβαση 22 Ιανουαρίου 2024].

15. Wikipedia contributors. (n.d.). Sass (stylesheet language). In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/Sass_(stylesheet_language) [Πρόσβαση 22 Ιανουαρίου 2024].

16. Node.js. (n.d.). Node.js. [Διαδίκτυο] Διαθέσιμο στο https://nodejs.org [Πρόσβαση 5 Φεβρουαρίου 2024].

17. NuxtJS. (n.d.). Getting Started - Introduction to NuxtJS. [Διαδίκτυο] Διαθέσιμο στο https://nuxt.com/docs/getting-started/introduction [Πρόσβαση 5 Φεβρουαρίου 2024].

18. NuxtJS DevTools. (n.d.). Getting Started with NuxtJS DevTools. [Διαδίκτυο] Διαθέσιμο στο https://devtools.nuxt.com/guide/getting-started [Πρόσβαση 5 Φεβρουαρίου 2024].

19. Wikipedia contributors. (n.d.). Node.js. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://el.wikipedia.org/wiki/Nodejs [Πρόσβαση 5 Φεβρουαρίου 2024].

20. Wikipedia contributors. (n.d.). HTML5. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://el.wikipedia.org/wiki/HTML5 [Πρόσβαση 5 Φεβρουαρίου 2024].

21. GitHub. (n.d.). GitHub. [Διαδίκτυο] Διαθέσιμο στο https://github.com [Πρόσβαση 5 Φεβρουαρίου 2024].

22. Wikipedia contributors. (n.d.). GitHub. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/GitHub [Πρόσβαση 5 Φεβρουαρίου 2024].

23. PostgreSQL. (n.d.). PostgreSQL: The world's most advanced open source relational database. [Διαδίκτυο] Διαθέσιμο στο https://www.postgresql.org [Πρόσβαση 5 Φεβρουαρίου 2024].

24. Wikipedia contributors. (n.d.). PostgreSQL. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/PostgreSQL [Πρόσβαση 5 Φεβρουαρίου 2024].

25. Wikipedia contributors. (n.d.). Database. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/Database [Πρόσβαση 19 Φεβρουαρίου 2024].

26. Next.js. (n.d.). Learn | Next.js - What is Next.js? [Διαδίκτυο] Διαθέσιμο στο https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs [Πρόσβαση 19 Φεβρουαρίου 2024].

27. Marianods81. (n.d.). Svelte.js: Why not? [Blog post]. [Διαδίκτυο] Διαθέσιμο στο https://marianods81.medium.com/svelte-js-why-not-c61dc455df4d [Πρόσβαση 19 Φεβρουαρίου 2024].

28. DigitalOcean Community. (n.d.). What is Laravel? [Tutorial]. [Διαδίκτυο] Διαθέσιμο στο https://www.digitalocean.com/community/tutorials/what-is-laravel [Πρόσβαση 19 Φεβρουαρίου 2024].

29. Wikipedia contributors. (n.d.). Hypertext Transfer Protocol. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/HTTP [Πρόσβαση 19 Φεβρουαρίου 2024].

30. W3Schools. (n.d.). What is JSON? [Διαδίκτυο] Διαθέσιμο στο https://www.w3schools.com/whatis/whatis_json.asp [Πρόσβαση 19 Φεβρουαρίου 2024].

31. Turing. (n.d.). What is NestJS & Why Use It in 2022? [Blog post]. [Διαδίκτυο] Διαθέσιμο στο https://www.turing.com/blog/what-is-nest-js-why-use-it-in-2022 [Πρόσβαση 19 Φεβρουαρίου 2024].

32. Tiangolo. (n.d.). FastAPI. [Διαδίκτυο] Διαθέσιμο στο https://fastapi.tiangolo.com [Πρόσβαση 19 Φεβρουαρίου 2024].

33. Wikipedia contributors. (n.d.). FastAPI. In Wikipedia, The Free Encyclopedia. [Διαδίκτυο] Διαθέσιμο στο https://en.wikipedia.org/wiki/FastAPI [Πρόσβαση 20 Φεβρουαρίου 2024].

34. IBM. (n.d.). REST APIs. [Διαδίκτυο] Διαθέσιμο στο https://www.ibm.com/topics/rest-apis [Πρόσβαση 20 Φεβρουαρίου 2024].

35. NGINX. (n.d.). NGINX Glossary. [Διαδίκτυο] Διαθέσιμο στο https://www.nginx.com/resources/glossary/nginx [Πρόσβαση 20 Φεβρουαρίου 2024].

36. Vue.js. (n.d.). Introduction to Vue.js. [Διαδίκτυο] Διαθέσιμο στο https://vuejs.org/guide/introduction.html [Πρόσβαση 3 Μαρτίου 2024].

37. Built In. (n.d.). Why Vue.js Is Good for More Than Just Application Prototypes. [Διαδίκτυο] Διαθέσιμο στο https://builtin.com/software-engineering-perspectives/vue-js [Πρόσβαση 3 Μαρτίου 2024].

38. Testim.io. (n.d.). What is a Linter? Here's a Definition and Quick Start Guide [Blog post]. [Διαδίκτυο] Διαθέσιμο στο https://www.testim.io/blog/what-is-a-linter-heres-a-definition-and-quick-start-guide [Πρόσβαση 3 Μαρτίου 2024].

39. Prettier. (n.d.). Prettier Documentation. [Διαδίκτυο] Διαθέσιμο στο https://prettier.io/docs/en/index.html [Πρόσβαση 3 Μαρτίου 2024].

40. AlphaServe. (n.d.). What is a SourceTree and How to Use It? [Blog post]. [Διαδίκτυο] Διαθέσιμο στο https://www.alphaservesp.com/blog/what-is-a-sourcetree-and-how-to-use-it [Πρόσβαση 3 Μαρτίου 2024].