Trian.gr - eshop Documentation

# ΓΕΝΙΚΉ ΕΙΚΌΝΑ

## JAVASCRIPT

Χρησιμοποιούνται τα εξής πρόσθετα στο front-end

* To Firebase που αναλαμβάνει το authentication
* To alpine.js που είναι το front-end framework και αναλαμβάνει τη λογική
* Bootstrap (για το CSS)

## AUTHENTICATION

To authentication γίνεται μέσω «Email link Authentication». To Firebase έχει blocking functions, ώστε αν κάποιος νέος χρήστης προσπαθήσει να φτιάξει λογαριασμό, τότε ελέγχει real-time μέσω το API αν ο χρήστης είναι υφιστάμενος πελάτης. Αν είναι ήδη χρήστης (έχει ήδη συνδεθεί στο παρελθόν), τότε δεν γίνεται αυτός ο έλεγχος. Σε περίπτωση διαγραφής, χρήστη, αυτός θα πρέπει να απενεργοποιηθεί από το Firebase UI.

Σχετικά αρχεία: auth.js, verify.js

## ΚΑΛΑΘΙ

Το καλάθι αποθηκεύεται στο localstorage του browser στο alpine αντικείμενο Shop.Cart, με τη βοήθεια του alpine.js. Κάθε αντικείμενο προστίθεται εκεί, ώστε να είναι διαθέσιμο από οπουδήποτε στη σελίδα. Κάθε object του καλαθιού, περιέχει τα απαραίτητα στοιχεία: τον αντικείμενο «item» (σκελετός ή φακός ή ζευγάρι φακών) ολόκληρο με τα στοιχεία του, την ποσότητα «quantity», το είδος του αντικειμένου «type» (frame, lens, pair) καθώς και ένα «id» που χρησιμεύει για να μπορεί να αφαιρεθεί το αντικείμενο από το καλάθι.

Αν το αντικείμενο του καλαθιού είναι ζευγάρι φακών, τότε type==”pair”, και το αντικείμενο αυτό στη θέση του «item» περιέχει ένα αντικείμενο με properties: R, L, type==”pair”, retail, όπου R,L είναι τα αντικείμενα φακοί, και retail είναι το όνομα του πελάτη λιανικής. Σε αυτήν την περίπτωση του ζευγαριού, οι φακοί L, R έχουν επίσης και το property “Axis”.

Σχετικό αρχείο: shop.js

## ΣΚΕΛΕΤΟΙ

Οι σκελετοί έρχονται όλοι συνολικά από το API, και φιλτράρονται στον browser, σύμφωνα με τις επιλογές του χρήστη. Έρχονται, είτε με prefetch, είτε όταν ο χρήστης επισκέπτεται τη σελίδα των σκελετών.

<link rel="prefetch" href="https://api.trian.gr/api/frames.json" as="fetch">

## ΦΑΚΟΙ

Αντίθετα οι φακοί, δεν μπορούν να έρθουν όλοι λόγω του τεράστιου μεγέθους. Έτσι, έρχονται τα παρακάτω αρχεία (με prefetch, είτε όταν ο χρήστης επισκέπτεται κάποια σελίδα φακών). Τα αρχεία αυτά περιέχουν μόνο τα πρώτα 2 φίλτρα, δηλαδή τα διαθέσιμα Σφαιρώματα και, για κάθε Σφαίρωμα, τους διαθέσιμους Κυλίνδρους. Αυτό γίνεται για κάθε τύπο φακών (trian, tokai) και για κάθε Γραφή (main, alt). Αυτά ονομάζονται **trees** των φακών.

<link rel="prefetch" href="https://api.trian.gr/api/tree/lens/trian.json" as="fetch">

<link rel="prefetch" href="https://api.trian.gr/api/tree/lens/tokai.json" as="fetch">

<link rel="prefetch" href="https://api.trian.gr/api/tree/lens/trian-alt.json" as="fetch">

<link rel="prefetch" href="https://api.trian.gr/api/tree/lens/tokai-alt.json" as="fetch">

## ΓΡΑΦΕΣ ΦΑΚΩΝ

Υπάρχουν τέσσερα διαθέσιμα «πρόσημα» για τους φακούς οι οποίες αντιστοιχούν σε ένα από τα δύο «γραφές».

* Πρόσημο +/+, με κύρια (main) γραφή
* Πρόσημο +/-, με εναλλακτική (alt) γραφή
* Πρόσημο -/+, με κύρια (main) γραφή
* Πρόσημο -/-, με εναλλακτική (alt) γραφή

Κάθε γραφή, έχει δύο πρόσημα. Όμως, τα πρόσημα αποτελούν απλώς φίλτρα των φακών (Σφαίρωμα/Κύλινδρος) κατά την αναζήτηση του πελάτη. Αυτό που έχει σημασία και ουσιαστική διαφοροποίηση είναι οι δύο γραφές (όχι τα πρόσημα).

Κάθε φακός, έρχεται με δύο γραφές στα χαρακτηριστικά του: Σφαίρωμα, Κύλινδρος, τα οποία αναφέρονται στην κύρια γραφή, και Σφαίρωμα2, Κύλινδριος2, τα οποία αναφέρονται στην εναλλακτική γραφή. Οπότε, με βάση την επιλογή της γραφής από το χρήστη, εμφανίζονται τα 2 πεδία που αντιστοιχούν (και τα άλλα 2 πεδία κρύβονται). Με βάση το πρόσημο, απλώς φιλτράρονται οι επιλογές των πεδίων.

## ΦΟΡΜΕΣ ΑΝΑΖΗΤΗΣΗΣ

Λόγω των πολλών επαναλήψεων των φορμών που απαιτούνται (φακός trian, αριστερός φακός trian, δεξιός φακός trian, φακός tokai, αριστερός φακός tokai, δεξιός φακός tokai), έχουν δημιουργηθεί web components, τα οποία ορίζονται στο shop.js.

Η δομή των web components στην html είναι:

<lens-division>

< lens-search-form></ lens-search-form>

<lens-results></lens-results>

</ lens-division>

(μεταξύ τους περιέχονται και άλλα html tags)

# φιλτρα αναζητησης και αποτελεσματα

## ΣΚΕΛΕΤΟΙ

Βλέποντας τη δομή του x-data για τους σκελετούς παρατηρούμε τα εξής:

Όλοι οι φακοί είναι φορτωμένοι στον browser. Το πεδίο «Κατασκευαστής» είναι υποχρεωτικό. Με κάθε αλλαγή των πεδίων φίλτρου γίνονται τα εξής:

* Φιλτράρονται τα προϊόντα (σκελετοί)  
  Αυτό γίνεται μέσω της συνάρτησης multiFilter (dim.js)
* Φιλτράρονται τα υπόλοιπα φίλτρα  
  Αυτό γίνεται μέσω της συνάρτησης filteredKey (dim.js)

## ΦΑΚΟΙ

Εδώ τα πράγματα είναι πιο πολύπλοκα.

Καταρχάς, με βάση τη γραφή (που καθορίζεται από το πρόσημο), εμφανίζονται τα 2 πεδία που αντιστοιχούν σε αυτή τη γραφή (Σφαίρωμα, Κύλινδρος, για την κύρια γραφή, ή Σφαίρωμα2, Κύλινδρος2, για την εναλλακτική γραφή). Στην περίπτωση του ζευγαριού φακών, εμφανίζεται το επιπλέον πεδίο Axis, το οποίο όμως είναι ανεξάρτητο του φιλτραρίσματος και της αναζήτησης, αλλά αποτελεί πληροφορία για το ζευγάρι.

Με βάση τα 2 πεδία φιλτραρίσματος, γίνεται ένα request στο API φέρνοντας τα προϊόντα που του αντιστοιχούν. Αυτό γίνεται μέσω της συνάρτησης filterViaApi.

Παρόλο που τα trees έρχονται για κάθε γραφή, το request για το φίλτρο είναι κοινό για τις 2 γραφές. Διαφέρει μόνο το path του request ανάλογα με τον κατασκευαστή Trian/Tokai.

Μόλις έρθουν τα προϊόντα (φακοί) από το API:

* Παρουσιάζονται τα προϊόντα
* Συμπληρώνονται τα υπόλοιπα φίλτρα  
  Αυτό γίνεται μέσω της συνάρτησης filteredKey (dim.js)

Στη συνέχεια, τα προϊόντα αυτά είναι φορτωμένα στον browser. Βάζοντας ο χρήστης επιπλέον φίλτρα, αυτά φιλτράρονται στον browser, χωρίς επιπλέον request στο API, με τον ίδιο τρόπο που γίνεται και στους σκελετούς.

Στη περίπτωση του ζευγαριού, οι φακοί φιλτράρονται επιπλέον με τον εξής κανόνα. Οι δύο φακοί πρέπει να έχουν τα ίδια πεδία «Τιμολογιακή Κατηγορία» και «Χρώμα». Το φιλτράρισμα γίνεται στον browser (όχι στο API).

## ΣΥΝΑΡΤΗΣΕΙΣ ΦΙΛΤΡΑΡΙΣΜΑΤΟΣ

* uniqueOf  
  Λαμβάνει ένα array από αντικείμενα. Τα αντικείμενα πρέπει να έχουν τα ίδια properties. Επιστρέφει ένα αντικείμενο, όπου τα properties του οποίου είναι τα μοναδικά properties που βρίσκονται στα αντικείμενα. Χρησιμοποιείται για να συμπληρώσει τις επιλογές των φίλτρων με βάση ένα σύνολο αντικειμένων.
* multiFilter  
  Κάνει το φιλτράρισμα των αντικειμένων. Λαμβάνει ένα array από αντικείμενα καθώς κι ένα αντικείμενο το οποίο είναι το φίλτρο. Το αντικείμενο αυτό έχει ως properties τα φίλτρα με τις τιμές τους.
* array2Filter  
  Φτιάχνει το φίλτρο ώστε να περαστεί μέσα στο multiFilter. Λαμβάνει ένα array με τα ονόματα των φίλτρων (χωρίς τις τιμές τους) και φτιάχνει το φίλτρο βρίσκοντας τις τιμές των φίλτρων από το parent που θα του υποδειχθεί. Χρησιμοποιείται μέσα στο alpine.
* filteredKey  
  Χρησιμοποιείται για να φιλτράρει τα φίλτρα. Είναι σχεδόν το uniqueOf(multiFilter(\_)). Η διαφορά του είναι ότι εδώ, το ίδιο το πεδίο του φίλτρου δεν φιλτράρεται με βάση τον εαυτό του (γιατί θα έδειχνε μόνο την τρέχουσα τιμή του). Κάθε φίλτρο, δηλαδή, φιλτράρεται με βάση τις τιμές των υπόλοιπων φίλτρων.