ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Σχεδίαση & Ανάπτυξη Web App για Καλλιτεχνικό Portal»

Του φοιτητή	**Επιβλέπων**
Δημήτριου Τσιαντάρη	**Μιχαήλ Σαλαμπάσης**
Αρ. Μητρώου: 164761	**Καθηγητής**

Θεσσαλονίκη 2024

Τίτλος Δ.Ε.: Σχεδίαση & Ανάπτυξη Web App για Καλλιτεχνικό Portal
Κωδικός Δ.Ε.: 23150
Ονοματεπώνυμο φοιτητή:Δημήτριος Τσιαντάρης
Ονοματεπώνυμο εισηγητή: Μιχαήλ Σαλαμπάσης
Ημερομηνία ανάληψης Δ.Ε.: 16-03-2023
Ημερομηνία περάτωσης Δ.Ε.: 26-05-2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Δημητρίου Τσιαντάρη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Στις μέρες μας, ένας από τους καλύτερους τρόπους για να παρουσιάσεις κάτι είναι μέσω μιας ιστοσελίδας στο διαδίκτυο. Ως web developer, έχεις τη δυνατότητα να επιλέξεις πώς θα παρουσιάσεις στο χρήστη το αντικείμενο που θέλει πραγματικά να δει, να εξερευνήσει ή να διαβάσει. Η δομή του front end μοιάζει σαν το να διακοσμείς έναν χώρο, αλλά πολύ πιο πολύπλοκη, καθώς ο ‘χώρος’ δεν είναι ενιαίος αλλά βρίσκεται στην οθόνη του κάθε χρήστη.

Η αυξημένη ανάγκη για εξειδικευμένες ιστοσελίδες που ανταποκρίνονται αποτελεσματικά στις απαιτήσεις των χρηστών μας οδήγησε σε μια εξαιρετικά δυναμική εξέλιξη των γλωσσών και των τεχνολογιών του διαδικτύου. Ο σκοπός αυτών των τεχνολογιών είναι να επιτρέπουν την ταχεία δημιουργία εφαρμογών με ελάχιστο κόστος, παρέχοντας παράλληλα μια πιο φιλική διαδικασία ανάπτυξης για τους προγραμματιστές.

Ο λόγος λοιπόν που αποφάσισα να ασχοληθώ με αυτήν τη διπλωματική εργασία είναι ο ενθουσιασμός μου με το δημιουργικό αυτό κομμάτι της ανάπτυξης λογισμικού, έχοντας τώρα την ευκαιρία να παρουσιάσω ένας άκρως ενδιαφέρον θέμα, το ελληνικό θέατρο, με τον δικό μου τρόπο.

Περίληψη

Το θέατρο, ως ένα από τα παλαιότερα μέσα έκφρασης της τέχνης, βρίσκει τις ρίζες του βαθιά ενσωματωμένες στον πολιτιστικό ιστό της αρχαίας Ελλάδας. Λειτουργεί ως ένα ισχυρό μέσο με το οποίο οι άνθρωποι συνδέονται, επικοινωνούν και δημιουργούν δεσμούς. Στον κόσμο του θεάτρου, οι άνθρωποι ξεκινούν ένα συλλογικό ταξίδι που όχι μόνο ψυχαγωγεί αλλά και αναδεικνύει τις πνευματικές και ηθικές τους διαστάσεις. Τρέφει τόσο τον εσωτερικό τους κόσμο όσο και την ευαισθησία τους στα κοινωνικά ζητήματα, προσφέροντας έναν ανακουφιστικό χώρο, μακριά από την ρουτίνα και τις προκλήσεις της καθημερινότητας.
Στην εποχή μας, όπου η τηλεόραση και οι μεγάλες κινηματογραφικές παραγωγές κυριαρχούν στον χώρο της ψυχαγωγίας, δεν πρέπει να ξεχνάμε την τέχνη από την οποία προέρχονται αυτοί οι σύγχρονοι γίγαντες. Είναι καθήκον μας να αξιοποιούμε τα εργαλεία που η τεχνολογία μας έχει προσφέρει για να φωτίσουμε και να γιορτάσουμε τον πολιτισμικό μας πλούτο.
Αυτή η διπλωματική εργασία προσπαθεί να φωτίσει την ανάπτυξη μιας ιστοσελίδας, η οποία θα επιτρέπει στους χρήστες να γνωρίζουν τον πλούτο του ελληνικού θεάτρου: θεατρικά έργα, θεατρικούς χώρους, ηθοποιούς, σεναριογράφους, σκηνοθέτες και όλους τους επαγγελματίες του κλάδου.
Επιπλέον, η εφαρμογή επιτρέπει στους χρήστες να ανακαλύπτουν θεατρικά έργα με βάση την γεωγραφική τους θέση, να έχουν πρόσβαση σε στατιστικά στοιχεία και να πραγματοποιούν συγκρίσεις μεταξύ διαφορετικών χρονικών περιόδων.
Για να υλοποιήσουμε αυτήν την καινοτόμα εφαρμογή, αξιοποιήσαμε τη δύναμη των προηγμένων τεχνολογιών και τεχνικών του διαδικτύου. Η παρούσα διπλωματική εργασία προσφέρει μια εκτενή εξερεύνηση και ανάλυση νέων τεχνικών, με πάτημα την εργασία του πρώην φοιτητή Δημητρίου Αναστασιάδη, με τίτλο `Web-Based Σύστημα Θεατρικών Παραστάσεων - Front End`.
Ειδικότερα θα αναλύσουμε τις τεχνολογίες που χρησιμοποιήσαμε για να υλοποιήσουμε μια πιο εξελιξημη εφαρμογή και να αυξήσουμε τις προοπτικές της ως ένα ολοκληρωμένο web app σύστημα

«Web-App design and development for Theatrical Portal - Front End»

«Dimitrios Tsiantaris»

Abstract

Theatre, as one of the oldest forms of artistic expression, finds its roots deeply embedded in the cultural tapestry of Ancient Greece. It serves as a powerful medium through which people connect, communicate, and forge bonds. In the realm of the theatre, individuals embark on a collective journey that not only entertains but also elevates their spiritual and ethical dimensions. It nourishes both their inner worlds and their awareness of societal issues, offering a respite from the monotony and challenges of daily existence.

In today's era, where television and grand cinematic spectacles dominate our entertainment landscape, we must not lose sight of the art that birthed these modern giants. It is our responsibility to leverage the tools that technology has bestowed upon us to illuminate and celebrate our cultural treasures. This thesis endeavours to shed light on the development of a web application designed to acquaint users with the rich world of Greek theatrical plays, venues, actors, playwrights, directors, and the myriad of individuals who shape this dynamic industry. Furthermore, the application empowers users to discover plays based on their geographical location, access statistical insights, and engage in thoughtful comparisons across different time periods.

To materialise this innovative application, we harnessed the power of cutting-edge web technologies and techniques. This thesis offers an extensive exploration and deconstruction of these technologies, based on former student’s Dimitrios Anastasiadis thesis, named `Web-Based System for Theatrical Plays - Front End`.

Moreover, we will analyse the technologies we used in order to create a more expandable app and increase its potential for a complete web app system.

Περιεχόμενα

Πρόλογος 3
Περίληψη 4
Abstract 5
Περιεχόμενα 6
Κατάλογος Σχημάτων 10
Συντομογραφίες 13
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Εισαγωγή 1
1.2 Ιστορική αναδρομή web εφαρμογών 1
1.3 Οι Πρώτες Ιστοσελίδες έναντι των Σύγχρονων Frameworks 1
1.4 Ιστορική αναδρομή των web frameworks 2
1.5 Στόχοι της εργασίας 4
1.6 Δομή εργασίας 5
1.7 Επίλογος 5
Κεφάλαιο 2ο: Web apps και APIs 6
2.1 Εισαγωγή 6
2.2 Τι είναι τα APIs 6
2.2.1 Ιστορική Εξέλιξη APIs 6
2.2.2 Γενικές κατηγορίες APIs: 7
2.3 Client Side APIs 7
2.4 Server side APIs 8
2.5 HTTP Requests 9
2.6 Stateless API and HTTP 10
2.7 Promise-Based HTTP Clients 11
2.7.1 Χρησιμότητα των Promise-Based Clients: 11
2.7.2 Axios 12
2.8 JSON 12
2.9 Περιγραφή του δικού μας API 13
2.9.1 API tables 14
2.9.2 Πίνακας User 15
2.9.2.1 Authentication 15
2.9.2.2 User Context 15
2.9.2.3 User relations 16
2.10 Επίλογος 16
Κεφάλαιο 3ο: Τεχνολογίες εργασίας 17
3.1 Εισαγωγή 17
3.2 Package Management 17
3.2.1 Το npm ως Package Manager 17
3.2.2 Γιατί Χρησιμοποιείται το npm; 17

3.3 Web scripting....19
3.3.1 Javascript....20
3.3.2 Typescript....20
3.3.2.1 Εγκατάσταση TypeScript....21
3.3.3 Javascript vs Typescript....22
3.4 JWT Authentication....22
3.4.1 Προέλευση και εξέλιξη:....22
3.4.2 Πως λειτουργεί το JWT....22
3.5 Axios Interceptors....23
3.5.1 Κατηγορίες Interceptors....23
3.5.2 Παραδείγματα Χρήσης....24
3.5.3 Πότε Χρησιμοποιείται ο Axios Interceptor....24
3.5.4 Θετικά της Χρήσης Axios Interceptors....25
3.5 React framework....25
3.5.1 JSX....26
3.5.2 Components....26
3.5.3 State....28
3.5.4 Hooks....28
3.5.4.1 useState....28
3.5.4.2 useEffect....29
3.5.5 Lifecycle methods....30
3.5.6 React Context....30
3.5.6.1 Λόγοι χρήσης του Context....31
3.6 Material UI....33
3.6.1 Material UI Components....34
3.6.2 Material UI Theme και ThemeProvider....35
3.7 Tailwind....37
3.8 Next JS....39
3.8.1 Δημιουργία σελίδων....39
3.8.2 Server-Side vs Static Site Generation....40
3.8.3 API Routes για Backend Λειτουργίες....41
3.9 Git....42
3.10 Dependencies lifecycle....43
3.11 Επίλογος....44
Κεφάλαιο 4ο: Περιγραφή εργασίας....46
4.1 Εισαγωγή....46
4.2 Refactoring για χρήση Typescript....46
4.2.1 Λόγοι προσθήκης Typescript....46
4.2.2 Προσθήκη Typescript....49
4.3 Προσθήκη Tailwind....52
4.4 Προσθήκη λογικής JWT....52
4.4.1 Χρήση interceptors....53
4.5 React Toast....54
4.5.1 Χρήση του Toastify....55

4.5.2 Generic-use Toasts........ 56
4.5.2.1 Success Toasts........ 56
4.5.2.2 Error Toasts........ 57
4.6 User Context........ 58
4.6.1 Δημιουργία UserContext και UserContextProvider........ 58
4.6.2 Δημιουργία user custom hooks........ 62
4.6.2.1 useUserQueries........ 62
4.6.2.2 useUserMutations........ 63
4.7 User page και user actions........ 71
4.7.1 User page........ 71
4.7.2 Άλλα User actions........ 78
4.8 Επιπλέον αλλαγές........ 81
4.8.1 Προσθήκη αναζήτησης........ 81
4.8.2 Breadcrumbs........ 82
4.8.3 MediaViewer Component........ 82
4.8.4 Reusable Card Components........ 83
4.8.5 Login page........ 85
4.8.6 Φίλτρα αναζήτησης........ 86
4.9 Επίλογος........ 86
Κεφάλαιο 5ο: Προτάσεις για βελτίωση........ 87
5.1 Outdated dependencies........ 87
5.2 Documentation........ 87
5.3 OAUTH2........ 87
5.4 Επικοινωνιακό........ 88
5.4.1 Jira Issues........ 89
ΒΙΒΛΙΟΓΡΑΦΙΑ........ 91

Κατάλογος Σχημάτων

Εικόνα 1.1: Πρώτη Ιστοσελίδα(World Wide Web)....2
Εικόνα 1.2: Χρήση δημοφιλών JS framework....4
Εικόνα 2.1: Τι είναι ένα API....6
Εικόνα 2.2: Χρήση HTTP Requests....10
Εικόνα 2.3: JS Promise....11
Εικόνα 2.4: Παράδειγμα χρήσης Axios....12
Εικόνα 2.5: Παράδειγμα αντικειμένου JSON....13
Εικόνα 2.6: Ο πίνακας ενός User....15
Εικόνα 3.1: Χρήση του NPM....18
Εικόνα 3.2: Παράδειγμα ενός package.json αρχείου....19
Εικόνα 3.3: Παράδειγμα tsconfig.json αρχείου....21
Εικόνα 3.4: Πως λειτουργεί το JWT....23
Εικόνα 3.5: Παράδειγμα χρήσης request interceptor....24
Εικόνα 3.6: Παράδειγμα χρήσης response interceptor....24
Εικόνα 3.7: Η React στη λίστα των frameworks....25
Εικόνα 3.8: React Class Component....27
Εικόνα 3.9: React Function Component....27
Εικόνα 3.10: React Arrow Function Component....27
Εικόνα 3.11: Σύνταξη του useState....28
Εικόνα 3.12: Παράδειγμα χρήση; του useState....29
Εικόνα 3.13: Σύνταξη του useEffect....29
Εικόνα 3.14: Παράδειγμα χρήσης useEffect....30
Εικόνα 3.15: Χρήση του createContext....30
Εικόνα 3.16: Expose context με το Context.Provider....31
Εικόνα 3.17: Χρήση του Context με Context.Consumer ή useContext....31
Εικόνα 3.18: Παράδειγμα global ThemeContext....32
Εικόνα 3.19: Δημοφιλείς εφαρμογές σε React....33
Εικόνα 3.20: Χρήση Button του MUI....34
Εικόνα 3.21: Χρήση props ενός MUI component....34
Εικόνα 3.22: Παράδειγμα Button MUI....34
Εικόνα 3.23: Παράδειγμα createStyles....35
Εικόνα 3.24: Παράδειγμα χρήσης useStyles....36
Εικόνα 3.25: Παράδειγμα χρήσης του ThemeContext....36
Εικόνα 3.26: Εγκατάσταση tailwind CSS....37
Εικόνα 3.27: Ενσωμάτωση tailwind CSS....37
Εικόνα 3.28: Το αρχείο tailwind.config.js....38
Εικόνα 3.29: Παράδειγμα χρήσης Tailwind CSS....38
Εικόνα 3.30: Παράδειγμα χωρίς Tailwind....39
Εικόνα 3.31: Δημιουργία σελίδας στο Next JS....40

Εικόνα 3.32: Παράδειγμα getServerSideProps..40
Εικόνα 3.33: Παράδειγμα getStaticProps.. 41
Εικόνα 3.34: API endpoint στη Next JS.. 42
Εικόνα 3.35: Ποσοστό χρήσης Next JS.. 42
Εικόνα 4.1: API call χωρίς TS.. 47
Εικόνα 4.2: Παράδειγμα χωρίς χρήση TS..47
Εικόνα 4.3: API call με TS..48
Εικόνα 4.4: Παράδειγμα με χρήση TS..48
Εικόνα 4.5: Autocomplete με TS.. 49
Εικόνα 4.6: Χρήση TS τα τελευταία χρόνια.. 49
Εικόνα 4.7: Εγκατάσταση Typescript με npm... 50
Εικόνα 4.8: Παράδειγμα ενός TS interface..50
Εικόνα 4.9: Αρχικός κώδικας σε JSX... 51
Εικόνα 4.10: Μετατροπή σε TSX.. 51
Εικόνα 4.11: React component με JSX.. 52
Εικόνα 4.12: React component με TSX.. 52
Εικόνα 4.13: Δημιουργία mainAxios instance..53
Εικόνα 4.14: Εισαγωγή JWT.. 53
Εικόνα 4.15: Εξαγωγή του JWT.. 54
Εικόνα 4.16: Χρήση του ToastContainer.. 55
Εικόνα 4.17: Χρήση του toast.success σε interceptor...56
Εικόνα 4.18: Παράδειγμα toast.success..56
Εικόνα 4.19: Χρήση του toast.error σε interceptor...57
Εικόνα 4.20: Παράδειγμα toast.error.. 58
Εικόνα 4.21: Δημιουργία UserContext και UserContextProvider.. 58
Εικόνα 4.22: Διαχείριση πεδίων του UserContextProvider.. 59
Εικόνα 4.23: Συνάρτηση handleLogout...59
Εικόνα 4.24: Κώδικας για email verification toast.. 60
Εικόνα 4.25: Verify email toast..60
Εικόνα 4.26: Επιστροφή των values του Provider..61
Εικόνα 4.27: Χρήση UserContextProvider στο _app.tsx...61
Εικόνα 4.28: Export του useUserContex hook.. 61
Εικόνα 4.29: To useUserQueries hook...62
Εικόνα 4.30: To useUserMutations hook... 63
Εικόνα 4.31: registerUser hook...64
Εικόνα 4.32: loginUser hook... 65
Εικόνα 4.33: addRole hook..65
Εικόνα 4.34: toggle2FA hook.. 66
Εικόνα 4.35: addBio hook..66
Εικόνα 4.36: claimAccount hook...67
Εικόνα 4.37: claimVenue hook.. 67
Εικόνα 4.38: claimProduction hook...68
Εικόνα 4.39: updateSocial hook...68
Εικόνα 4.40: uploadUserPhoto hook... 69

Εικόνα 4.41: deleteUserPhoto hook....69
Εικόνα 4.42: requestPhoneVerification hook....70
Εικόνα 4.43: confirmPhoneVerification hook....70
Εικόνα 4.44: General Tab....71
Εικόνα 4.45: Theatrical Tab....71
Εικόνα 4.46: Πολυμέσα Tab....72
Εικόνα 4.47: Theme Tab....72
Εικόνα 4.48: To SelectProfilePhotoDialog....73
Εικόνα 4.49: Κώδικας για δημιουργία checkout session....74
Εικόνα 4.50: Checkout session popup window....74
Εικόνα 4.51: Χρηση payment overlay....75
Εικόνα 4.52: Κώδικας για Backdrop....75
Εικόνα 4.53: To AddBioDialog component....76
Εικόνα 4.54: To AddPhoneDIalog component....76
Εικόνα 4.53: To AddRolesDialog component....77
Εικόνα 4.54: Κώδικας του AddRolesDialog....77
Εικόνα 4.55: Upload Photo prompt....78
Εικόνα 4.56: Delete Photo prompt....78
Εικόνα 4.57: User modal....79
Εικόνα 4.58: Κουμπί claim για artists....79
Εικόνα 4.59: To component ClaimPersonDialog....80
Εικόνα 4.60: Claim Venue....80
Εικόνα 4.61: Edit Venue....81
Εικόνα 4.62: To EditVenueDialog component....81
Εικόνα 4.63: Δυναμική αναζήτηση....82
Εικόνα 4.64: Breadcrumbs....82
Εικόνα 4.65: MediaViewer Component....83
Εικόνα 4.66: News Card....83
Εικόνα 4.67: Artist Card....84
Εικόνα 4.68: Show Card....84
Εικόνα 4.69: Venue Card....85
Εικόνα 4.70: Login page....85
Εικόνα 4.71: Φίλτρα αναζήτησης....86
Εικόνα 5.1: Λειτουργία του OAUTH2....88
Εικόνα 5.2: Διαδικτυακό meeting 31-10-2023....89
Εικόνα 5.3: Παράδειγμα Jira issue....90

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
API	Application Programming Interface
ASP	Active Server Pages
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IDE	Integrated Development Environment
ISR	Incremental Static Regeneration
JS	JavaScript
JSON	JavaScript Object Notation
JSP	JavaServer Page
JSX	JavaScript XML
JWT	JSON Web Token
MUI	Material User Interface
NPM	Node Package Manager
PHP	Hypertext Preprocessor (Personal Home Page)
SEO	Search Engine Optimization
SPA	Single Page Application
SSL	Secure Sockets Layer
TS	Typescript
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
WCAG	Web Content Accessibility Guidelines
XML	Extensible Markup Language

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Στο πρώτο κεφάλαιο της πτυχιακής εργασίας θα παρουσιάσουμε τη γενικότερη δομή του τελικού web app. Επίσης θα περιγράψουμε τα δεδομένα τα οποία χρησιμοποιήσαμε και θα γίνει μια αναφορά στα διάφορα προβλήματα τα οποία αντιμετωπίσαμε και ξεπεράσαμε σε συνεργασία με τα άτομα που εργάστηκαν στο backend της εφαρμογής, καθώς και με τον επιβλέποντα κο Σαλαμπάση, αλλά και τους φοιτητές των οποίων τις εργασίες χρησιμοποιήσαμε ως βάση.

1.2 Ιστορική αναδρομή web εφαρμογών

Οι web εφαρμογές είναι προγράμματα που εκτελούνται σε έναν web server και προσφέρονται στον χρήστη μέσω ενός web browser. Σε αντίθεση με τις παραδοσιακές εφαρμογές που εγκαθίστανται τοπικά στον υπολογιστή, οι web εφαρμογές είναι προσβάσιμες από οπουδήποτε με έναν browser.

Από την εμφάνιση του πρώτου web browser στις αρχές της δεκαετίας του '90, το web έχει δει μια τεράστια εξέλιξη. Οι αρχικές ιστοσελίδες ήταν στατικές, γραμμένες σε απλή HTML. Με την εισαγωγή της JavaScript, οι σελίδες άρχισαν να γίνονται δυναμικές, προσφέροντας περισσότερη διαδραστικότητα.

Καθώς το web εξελίσσεται, η ανάγκη για πιο πολύπλοκες εφαρμογές αυξάνεται. Γι' αυτό, εμφανίζονται τα frameworks, όπως η React. Τα frameworks αυτά προσφέρουν έναν πιο οργανωμένο τρόπο για την ανάπτυξη web εφαρμογών, βελτιώνοντας την απόδοση και τη διαδραστικότητα.

1.3 Οι Πρώτες Ιστοσελίδες έναντι των Σύγχρονων Frameworks

Οι πρώτες ιστοσελίδες ήταν απλές και στατικές. Γραμμένες καθαρά σε HTML, ίσως με λίγο CSS για το styling, αυτές οι σελίδες εξυπηρετούσαν τον σκοπό τους, αλλά χωρίς πολλή διαδραστικότητα ή δυναμικές λειτουργίες.[1]

Με την εμφάνιση της JavaScript, οι ιστοσελίδες άρχισαν να γίνονται πιο δυναμικές. Ωστόσο, καθώς οι εφαρμογές γίνονταν πιο πολύπλοκες, ο κώδικας έγινε δυσκολότερος στη διαχείριση. Τα frameworks, όπως η React, ανταποκρίθηκαν σε αυτήν την ανάγκη, προσφέροντας μια δομημένη προσέγγιση στην ανάπτυξη εφαρμογών. Πέρα από την οργάνωση, τα frameworks βελτιώνουν την απόδοση, την ασφάλεια και τη δυνατότητα επεκτασιμότητας των εφαρμογών. [2]

Συνολικά, ενώ οι πρώτες ιστοσελίδες εξυπηρέτησαν τον σκοπό τους για την εποχή τους, τα σύγχρονα frameworks έχουν επαναπροσδιορίσει το τι είναι δυνατόν στον χώρο του web development, επιτρέποντας τη δημιουργία πολύπλοκων, ασφαλών και υψηλής απόδοσης εφαρμογών, όπως η εργασία αυτή.

Εικόνα 1.1: Πρώτη Ιστοσελίδα(World Wide Web)

1.4 Ιστορική αναδρομή των web frameworks.

Όταν οι ιστοσελίδες άρχισαν να εξελίσσονται σε πιο πολύπλοκες web εφαρμογές, τα frameworks προέκυψαν ως λύση για την ανάπτυξη και διαχείριση του μεγαλύτερου όγκου κώδικα. Αυτά τα εργαλεία πρόσφεραν μια δομημένη βάση για τον σχεδιασμό, τη δημιουργία και τη συντήρηση των εφαρμογών.

- Πρώτα Δημοφιλή Frameworks:

Τα πρώτα frameworks που έγιναν δημοφιλή ήταν το jQuery, το AngularJS και το Backbone.js. Το jQuery, που εμφανίστηκε το 2006, απλοποίησε τη διαδραστικότητα και τις αλληλεπιδράσεις μέσα σε ιστοσελίδες. Το AngularJS, από την άλλη πλευρά, ήταν ένα από τα πρώτα frameworks που προσέφεραν μια πλήρη λύση για τη δημιουργία single-page applications (SPAs). [3]

- Στατιστικά για τα Σημερινά Frameworks:

Τα frameworks όπως React, Angular και Vue.js έχουν κατακτήσει τεράστια δημοτικότητα τα τελευταία χρόνια. Σύμφωνα με έρευνες το 2022, πάνω από 2 εκατομμύρια προγραμματιστές έχουν εμπειρία στην ανάπτυξη με React, και περίπου 1.5 εκατομμύριο για το Angular. Εκατοντάδες χιλιάδες ιστοσελίδες, συμπεριλαμβανομένων πολλών από τις 10.000 πιο δημοφιλείς, είναι γραμμένες με αυτά τα frameworks.

- Ιδιότητες και Χαρακτηριστικά των Frameworks:

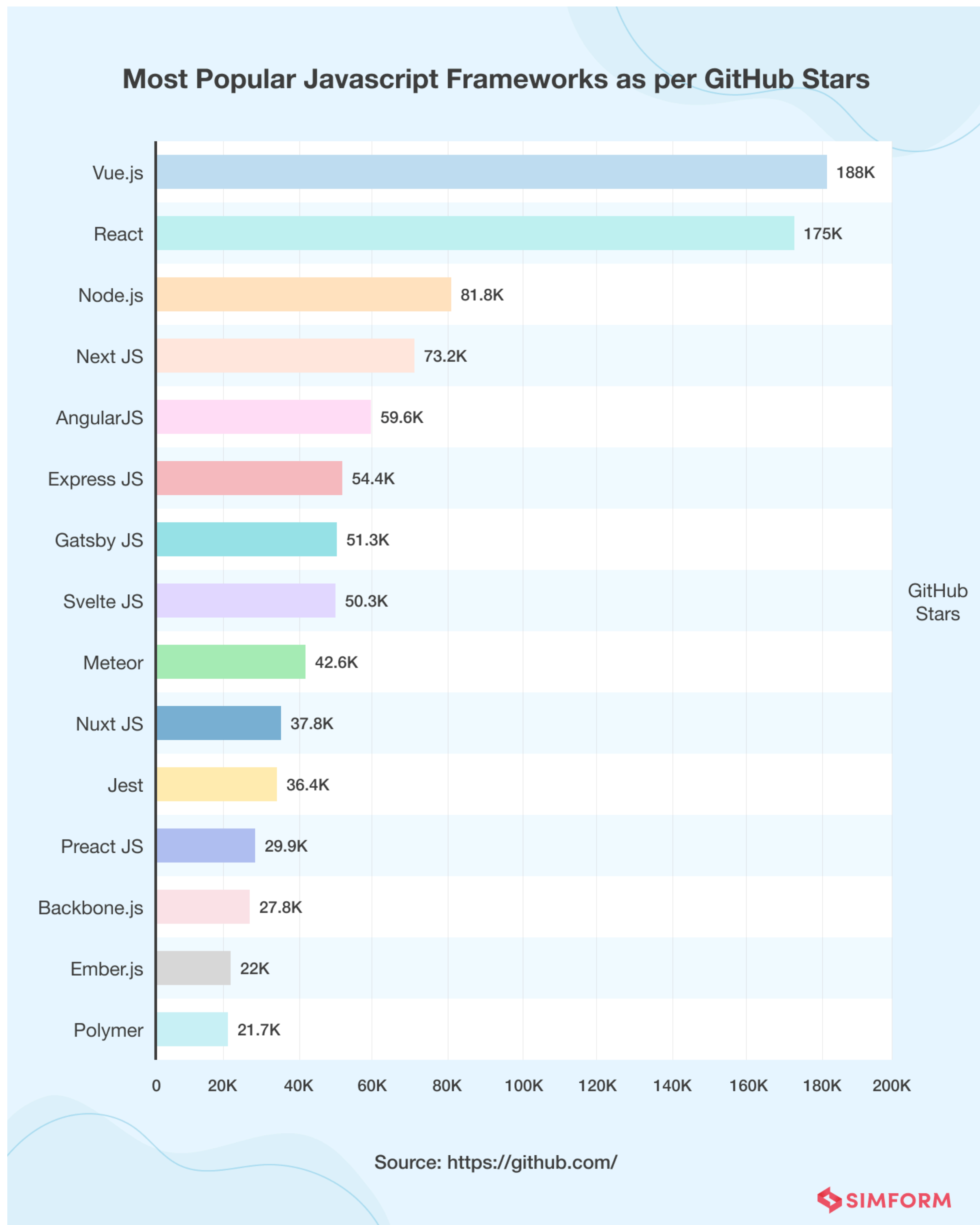
1. **Οργάνωση**: Διαρθρωτική οργάνωση του κώδικα με patterns όπως MVC (Model-View-Controller).
2. **Απόδοση**: Βελτιστοποίηση των εφαρμογών για γρήγορη απόκριση και χαμηλότερη κατανάλωση πόρων.

3. **Επεκτασιμότητα**: Δυνατότητα προσθήκης λειτουργιών χωρίς προβλήματα στο υπάρχον σύστημα.
4. **Ασφάλεια**: Παροχή εργαλείων για την προστασία από διάφορες επιθέσεις.

Κατηγορίες των Web Frameworks:
Τα frameworks χωρίζονται σε διάφορες κατηγορίες, ανάλογα με το είδος των εφαρμογών που υποστηρίζουν:

1. **Client-Side (ή Front-end) Frameworks:** όπως React, Vue.js, Angular, που διαχειρίζονται την παρουσίαση και την αλληλεπίδραση στον πελάτη.
2. **Server-Side (ή Back-end) Frameworks:** όπως Express.js (για Node.js), Django (για Python), Ruby on Rails (για Ruby).
3. **Full Stack Frameworks**: Συνδυάζουν τόσο το front-end όσο και το back-end, όπως το Meteor.
4. **Server Side Rendering (SSR):** Στο SSR, η σελίδα παράγεται από τον server πριν αποσταλεί στον πελάτη. Αυτό συνήθως παρέχει γρηγορότερη αρχική φόρτωση σελίδας σε σύγκριση με το client-side rendering, καθώς και καλύτερη δεικτοδότηση από τις μηχανές αναζήτησης. Frameworks όπως το Next.js (βασισμένο στο React) προσφέρουν SSR, καθιστώντας το δυνατό να συνδυάζουν τα οφέλη των SPA με τα πλεονεκτήματα του SSR.

Η διάδοση των frameworks και η συνεχής εξέλιξή τους είναι αναπόσπαστο μέρος της εξέλιξης του web. Καθώς οι εφαρμογές γίνονται όλο και πιο πολύπλοκες, τα εργαλεία αυτά παρέχουν τα μέσα για να διαχειριστούμε αυτήν την πολυπλοκότητα αποτελεσματικά.



Εικόνα 1.2: Χρήση δημοφιλών JS framework

1.5 Στόχοι της εργασίας

Η εφαρμογή που είχε ως αντικείμενο την παροχή πληροφοριών για ηθοποιούς, σκηνοθέτες, θέατρα και θεατρικές παραστάσεις αποτέλεσε ένα σύνθετο εγχείρημα που ζητούσε λεπτομερή προσοχή στην ανάπτυξη και τη βελτίωσή της. Οι βασικοί στόχοι της εργασίας, με βάση τις πληροφορίες που παρασχέθηκαν, περιλαμβάνουν τα εξής:

1. **Επεκτασιμότητα της Εφαρμογής:** Ο κύριος και πλέον προτεραιότητας στόχος της εργασίας ήταν να διασφαλίσει ότι η εφαρμογή θα μπορούσε να επεκταθεί εύκολα στο μέλλον. Καθώς το θέατρο είναι ένας δυναμικός χώρος και οι πληροφορίες αλλάζουν και ανανεώνονται συνεχώς, ήταν απαραίτητο η εφαρμογή να είναι σχεδιασμένη με τρόπο που να επιτρέπει εύκολες προσθήκες, τροποποιήσεις και βελτιώσεις χωρίς να επηρεάζεται η σταθερότητα της.
2. **Ενσωμάτωση του Typescript:** Για να εξασφαλίσει την σταθερότητα, την ασφάλεια και την επεκτασιμότητα της εφαρμογής, ενσωματώθηκε το Typescript. Το Typescript προσφέρει έναν πιο ξεκάθαρο και δομημένο τρόπο ανάπτυξης, κάνοντας τον κώδικα πιο κατανοητό και διαχειρίσιμο.
3. **Refactor της λογικής ύπαρξης 'χρήστη΄:** Ο τρόπος που ένας επισκέπτης της σελίδας δημιουργεί και διαχειρίζεται έναν λογαριασμό, καθώς και οι ενέργειες που θα μπορεί να κάνει, αλλάζουν ριζικά και επεκτείνονται.

Συνοψίζοντας, η εργασία επικεντρώθηκε στη δημιουργία μιας εφαρμογής που θα ήταν ταυτόχρονα σταθερή, ασφαλής, εύκολη στη χρήση και πάνω απ' όλα επεκτάσιμη, ώστε να μπορεί να προσαρμόζεται στις συνεχώς αλλαζόμενες ανάγκες του κοινού και της καλλιτεχνικής κοινότητας.

1.6 Δομή εργασίας

Η εργασία χωρίστηκε σε 5 κεφάλαια με βάση το πως δομήθηκε εξαρχής ως project. Στόχος της δομής αυτής ειναι ο αναγνώστης να κατανοήσει τον τρόπο σκέψης μου καθόλη της διάρκεια της ανάπτυξης της ΔΕ, τα προβλήματα που αντιμετώπισα και οι πληροφορίες που είχα διαθέσιμες για να τα λύσω. Οπότε εν τάχει, η δομή που θα ακολουθήσει είναι η εξής:

1. Πρώτο και τρέχον κεφάλαιο γενικής εισαγωγής. Παρουσίαση του τι θα ακολουθήσει.
2. Δεύτερο κεφάλαιο που ακολουθεί, η ανάλυση της αρχιτεκτονικής των σύγχρονων web apps και η επικοινωνία τους με remote APIs. Θα αναλύσουμε επίσης σύντομα τη δομή του δικού μας API.
3. Τρίτο κεφάλαιο, η λεπτομερής ανάλυση τεχνολογιών και τεχνικών που χρησιμοποιήθηκαν και τα προβλήματα που επίλυσαν. Μικρή ιστορική αναδρομή και ανάλυση των θετικών και αρνητικών κάθε τεχνολογίας.
4. Στο τέταρτο κεφάλαιο θα αναλύσουμε τη σχεδίαση και το refactor της εφαρμογής που παρέλαβα. Θα μιλήσουμε για το τι αντικαταστάθηκε, τι ήταν χρήσιμο και τι επεκτάθηκε ή προστέθηκε.
5. Στο κεφάλαιο 5, θα γίνει αναφορά στο επικοινωνιακό κομμάτι της εργασίας αυτής, η οποία αποτελεί κομμάτι ενός μεγαλύτερου project. Θα παρουσιαστούν τα συλλογικά προβλήματα που είχαμε και πως αντιμετωπίστηκαν μέσω καλής επικοινωνίας.

1.7 Επίλογος

Στο κεφάλαιο αυτό, κάναμε μια μικρή περιήγηση στο τι θα αναπτύξουμε παρακάτω με λεπτομέρεια. Εξηγήσαμε τι προσπαθεί να επιτύχει και αναφέραμε τεχνολογίες και ιδέες που χρησιμοποιεί ή ευελπιστούμε να χρησιμοποιήσει στο μέλλον.

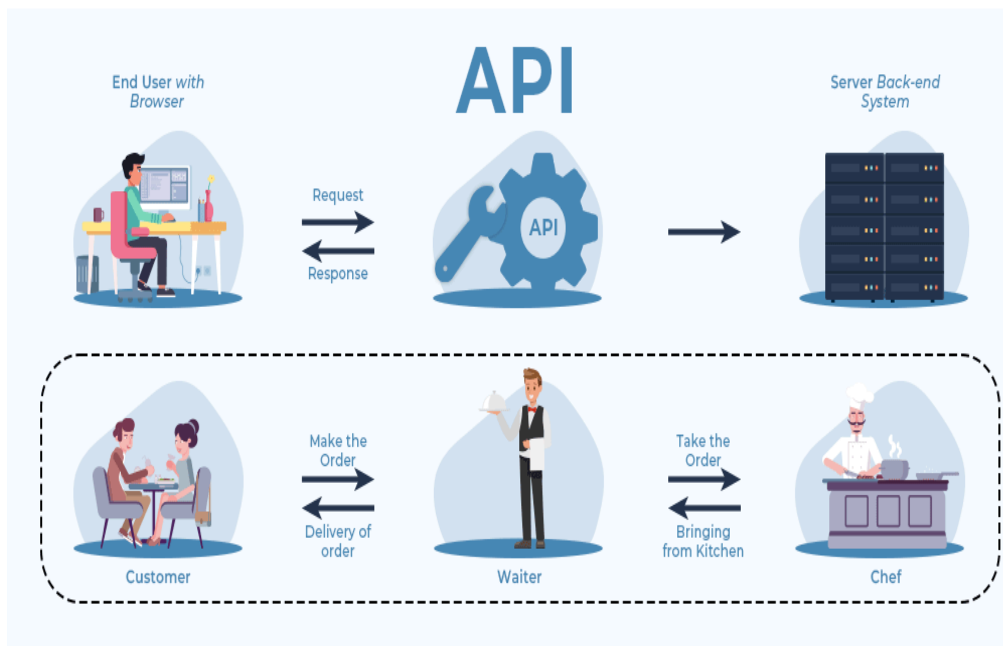
Κεφάλαιο 2ο: Web apps και APIs

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε πόσο σημαντική είναι η σωστή επικοινωνία με τον εξωτερικό server. Θα παρουσιάσουμε τι είναι τα APIs και γιατί είναι ο καλύτερος τρόπος. Επίσης θα εξηγήσουμε με λεπτομέρεια τα δεδομένα που χειριστήκαμε σε αυτήν την εφαρμογή, σχετικές τεχνολογίες για την επικοινωνία της εφαρμογής μας με το remote API και κάποιες σκέψεις για το μέλλον.

2.2 Τι είναι τα APIs

Στο σημερινό κόσμο της τεχνολογίας, η αποτελεσματική και ασφαλής επικοινωνία μεταξύ των web εφαρμογών και των εξωτερικών servers είναι κρίσιμη. Τα Application Programming Interfaces (APIs) παίζουν κεντρικό ρόλο στην επικοινωνία αυτή, παρέχοντας μια σταθερή και ευέλικτη διασύνδεση για την ανταλλαγή δεδομένων και πληροφοριών.
Τα APIs ή Application Programming Interfaces είναι σύνολα οδηγιών, πρωτοκόλλων και εργαλείων για τη δημιουργία λογισμικού και εφαρμογών. Αποτελούν τον "γλωσσικό μεσολαβητή" μεταξύ διαφόρων εφαρμογών και πλατφορμών, επιτρέποντας τους να επικοινωνούν μεταξύ τους χωρίς να απαιτείται να γνωρίζουν το εσωτερικό τους σχεδιασμό. [4]



Εικόνα 2.1: Τι είναι ένα API

2.2.1 Ιστορική Εξέλιξη APIs

Τα Πρώτα Βήματα: Τα APIs υπήρχαν σε διάφορες μορφές από τις αρχές της πληροφορικής, αλλά η εμφάνιση του διαδικτύου και των web εφαρμογών στα τέλη της δεκαετίας του '90 και του 2000, έφερε μια νέα εποχή για τα APIs.
Η Εποχή των Web APIs: Με την ανάδυση των web services και SOA (Service-Oriented Architecture) στις αρχές του 2000, τα APIs άρχισαν να γίνονται πιο προσβάσιμα και συνδεδεμένα με το διαδίκτυο.

2.2.2 Γενικές κατηγορίες APIs:

- Δημόσια APIs (Public APIs): Προσφέρονται ανοικτά για τους προγραμματιστές και επιχειρήσεις.
- Ιδιωτικά APIs (Private APIs): Χρησιμοποιούνται εσωτερικά σε μια επιχείρηση.
- Εταιρικά APIs (Partner APIs): Παρέχονται σε συγκεκριμένους, εξωτερικούς συνεργάτες.
- Σύνθετα APIs (Composite APIs): Συνδυάζουν διάφορα APIs για να προσφέρουν πιο σύνθετες λειτουργίες.[6]

Σημασία των APIs στα Σύγχρονα Web Apps:
Στη σύγχρονη εποχή, τα APIs είναι ζωτικής σημασίας για τις web εφαρμογές. Παρέχουν μια σταθερή και ασφαλή μέθοδο για την ανταλλαγή δεδομένων, επιτρέποντας στις εφαρμογές να επεκτείνουν τις λειτουργίες τους και να ενσωματώσουν εξωτερικές υπηρεσίες με ευκολία. Αυτό είναι ιδιαίτερα σημαντικό στην εποχή του cloud computing και της μεγάλης διασποράς των δεδομένων, όπου η δυνατότητα απρόσκοπτης επικοινωνίας και συνεργασίας μεταξύ διαφόρων πλατφορμών και εφαρμογών καθίσταται απαραίτητη.

2.3 Client Side APIs.

Τα Client-Side APIs είναι διεπαφές προγραμματισμού που εκτελούνται στην πλευρά του πελάτη, δηλαδή στον web browser του χρήστη. Αυτά τα APIs επιτρέπουν τη διαδραστικότητα μέσα σε μια ιστοσελίδα, ενισχύοντας την εμπειρία του χρήστη μέσω διαδραστικών στοιχείων όπως φόρμες, animations και παιχνίδια.[7]

Χαρακτηριστικά των Client-Side APIs:

- Εκτέλεση στον Browser: Το κώδικας τρέχει στον browser του χρήστη και όχι στον server.
- Απόκριση στην Αλληλεπίδραση του Χρήστη: Αυτά τα APIs ανταποκρίνονται σε ενέργειες του χρήστη, όπως κλικ και κινήσεις ποντικιού.
- Περιορισμοί Ασφαλείας: Λόγω ασφαλειακών περιορισμών, ορισμένες λειτουργίες είναι περιορισμένες στα client-side APIs για να αποφευχθεί η εκτέλεση επιβλαβούς κώδικα.

Παραδείγματα Δημοφιλών Client-Side APIs:

1. DOM API (Document Object Model API): Παρέχει δυνατότητες για τη διαχείριση και τροποποίηση του περιεχομένου της ιστοσελίδας, της δομής και των στυλ της. Επιτρέπει στο JavaScript να αλλάζει τα στοιχεία, τα στυλ και τη δομή του HTML.

2. Canvas API: Χρησιμοποιείται για το σχεδιασμό γραφικών και animations απευθείας σε ένα στοιχείο <canvas> της HTML. Ιδανικό για παιχνίδια, γραφικές αναπαραστάσεις και άλλες οπτικές εφαρμογές.
3. WebGL API: Μια JavaScript API που επιτρέπει την αναπαράσταση 3D γραφικών μέσα στον browser χωρίς τη χρήση plugins. Χρησιμοποιείται στην ανάπτυξη παιχνιδιών και οπτικά εντυπωσιακών web εφαρμογών.
4. Web Audio API: Παρέχει ένα πλούσιο σύστημα για τη δημιουργία και επεξεργασία ήχου στον web browser. Ιδανική για παιχνίδια, μουσικές εφαρμογές και ήχους σε web εφαρμογές.

2.4 Server side APIs

Τα Server-Side APIs είναι πιο πολύπλοκα και ισχυρά, καθώς επιτρέπουν την επεξεργασία δεδομένων και την εκτέλεση εργασιών στον server. Αυτό επιτρέπει τη δημιουργία πιο ασφαλών και δυναμικών εφαρμογών.

Χαρακτηριστικά των Server-Side APIs:

- Επεξεργασία στον Server: Ο κώδικας εκτελείται στον server, και μόνο τα αποτελέσματα στέλνονται πίσω στον client.
- Πρόσβαση στη Βάση Δεδομένων: Επιτρέπουν την ανάγνωση, ενημέρωση και διαγραφή δεδομένων από τη βάση δεδομένων.
- Ευελιξία στη Διαχείριση Δεδομένων: Προσφέρουν περισσότερες δυνατότητες για τη διαχείριση και επεξεργασία δεδομένων, συμπεριλαμβανομένων σύνθετων ερωτημάτων και τροποποιήσεων.
- Ασφάλεια: Παρέχουν αυξημένα μέτρα ασφαλείας, καθώς είναι δυνατόν να περιορίσουν την πρόσβαση και να ελέγχουν τις ενέργειες που μπορούν να εκτελεστούν από τον χρήστη.

Σημασία των Server-Side APIs στις Σύγχρονες Εφαρμογές:
Στη σύγχρονη ανάπτυξη web εφαρμογών, τα server-side APIs είναι θεμελιώδη για τη δημιουργία ευέλικτων, ασφαλών και αποδοτικών εφαρμογών. Προσφέρουν τη δυνατότητα επεξεργασίας μεγάλων όγκων δεδομένων, ενσωματώνουν λειτουργίες ασφαλείας και επιτρέπουν την απρόσκοπτη ενσωμάτωση διαφόρων υπηρεσιών και εφαρμογών. Η ικανότητα ενός web app να συνδεθεί αποτελεσματικά με ένα καλά δομημένο server-side API είναι κρίσιμη για την κλιμακωσιμότητα, την απόδοση και τη συνολική επιτυχία της εφαρμογής.

Παραδείγματα Δημοφιλών Server-Side APIs:

1. **RESTful APIs**: Οι RESTful APIs (Representational State Transfer) χρησιμοποιούν HTTP requests για την ανάκτηση, δημιουργία, ενημέρωση και διαγραφή δεδομένων. Είναι ευέλικτες και μπορούν να υποστηρίξουν πολλαπλές αναπαραστάσεις δεδομένων όπως JSON, XML κ.α.
2. **SOAP APIs**: Οι SOAP (Simple Object Access Protocol) APIs είναι πιο αυστηροί σε σχέση με τους RESTful APIs και βασίζονται σε XML για την ανταλλαγή δεδομένων. Προσφέρουν υψηλά επίπεδα ασφαλείας και είναι κατάλληλοι για επιχειρηματικές εφαρμογές.

3. **GraphQL**: Ένα πιο σύγχρονο API που επιτρέπει στους πελάτες να ορίζουν ακριβώς τι δεδομένα χρειάζονται, μειώνοντας τον περιττό όγκο δεδομένων και αυξάνοντας την απόδοση των εφαρμογών.
4. **OAuth**: Ένα πρωτόκολλο που επιτρέπει την ασφαλή εξουσιοδότηση σε web εφαρμογές, κυρίως χρησιμοποιείται για την επικύρωση χρηστών και την ασφαλή ανταλλαγή διαπιστευτηρίων.

Τα server-side APIs είναι θεμελιώδη για τη δημιουργία δυναμικών και σύγχρονων web εφαρμογών, παρέχοντας τη δυνατότητα στις εφαρμογές να επεξεργάζονται δεδομένα και να εκτελούν λειτουργίες που δεν θα ήταν δυνατές μόνο στην πλευρά του πελάτη.

Στη δική μας εργασία θα αναλύσουμε πως χρησιμοποιήσαμε ένα **RESTful** API για να επικοινωνούμε με τη βάση δεδομένων, έτσι ώστε να αντλούμε, να προσθέτουμε ή να μετατρέπουμε δεδομένα. Αρχικά θα μελετήσουμε λίγο τα HTTP Requests τα οποία είναι θεμελιώδη για τη λειτουργία ενός τέτοιου API.

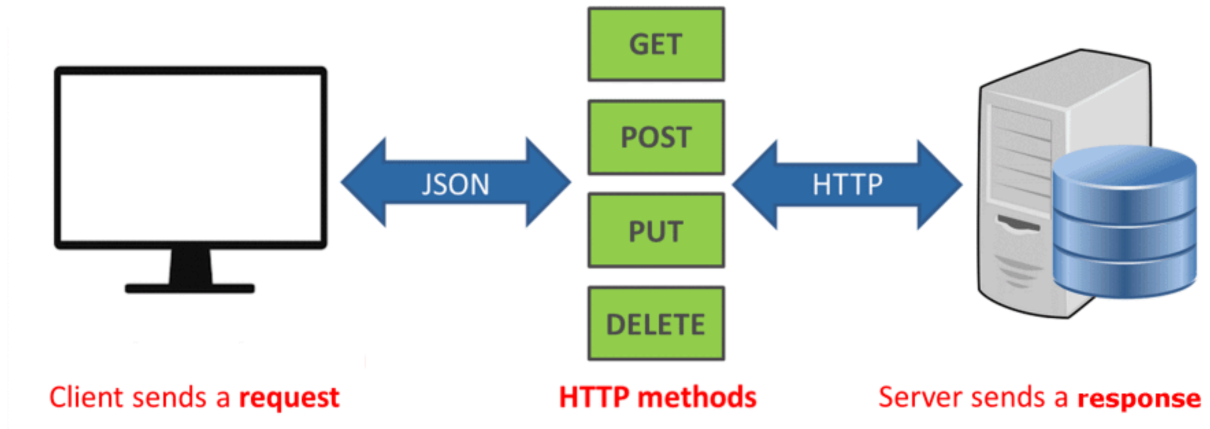
2.5 HTTP Requests

Στην εργασία σας, χρησιμοποιείται ένα remote stateless API. Ένα "stateless" API σημαίνει ότι κάθε αίτημα (request) προς το API είναι ανεξάρτητο από τα άλλα και ο server δεν διατηρεί κατάσταση (state) ανάμεσα στα αιτήματα. Αυτό είναι ένα βασικό χαρακτηριστικό των RESTful APIs και βοηθά στην απλοποίηση της σχεδίασης και της υλοποίησης του API.

Τα HTTP requests είναι οι αιτήσεις που στέλνονται από έναν client (π.χ., έναν web browser) προς έναν server για την ανάκτηση ή τροποποίηση δεδομένων. Υπάρχουν διάφοροι τύποι HTTP requests, οι οποίοι καθορίζουν το είδος της επικοινωνίας και της ενέργειας που ζητείται από τον server[8]:

- **GET**: Χρησιμοποιείται για την ανάκτηση δεδομένων από τον server. Είναι ένα αδρανές αίτημα που δεν πρέπει να προκαλεί καμία αλλαγή στην κατάσταση του server.
- **POST**: Χρησιμοποιείται για την αποστολή δεδομένων προς τον server για δημιουργία ή ενημέρωση. Συχνά χρησιμοποιείται για την υποβολή φορμών.
- **PUT**: Χρησιμοποιείται για την αντικατάσταση όλων των τρεχουσών αναπαραστάσεων του στόχου πόρου με το φορτίο αιτήματος (payload).
- **DELETE**: Χρησιμοποιείται για τη διαγραφή των πόρων που καθορίζονται από την URI.
- **PATCH**: Χρησιμοποιείται για την εφαρμογή μερικών τροποποιήσεων σε έναν πόρο.

Η σωστή χρήση αυτών των τύπων αιτήσεων είναι κρίσιμη για την αποτελεσματική και ασφαλή λειτουργία ενός web app που επικοινωνεί με ένα API. Η επιλογή του κατάλληλου τύπου αιτήματος βοηθά στην διατήρηση της συνέπειας, της διαφάνειας και της ασφάλειας των δεδομένων που ανταλλάσσονται μεταξύ της εφαρμογής και του server.



Εικόνα 2.2: Χρήση HTTP Requests

2.6 Stateless API and HTTP

Στα stateless APIs, κάθε αίτημα (request) που αποστέλλεται από τον client προς τον server είναι αυτοτελές και περιέχει όλες τις απαραίτητες πληροφορίες για την εκτέλεση του. Δεν υπάρχει διατήρηση κατάστασης (state) μεταξύ των αιτημάτων, κάτι που επιφέρει αυξημένη ασφάλεια και απλότητα στην διαχείριση των αιτημάτων.[9]

1. Δημιουργία του Αιτήματος:
    - Ο client δημιουργεί ένα HTTP αίτημα, το οποίο περιλαμβάνει την μέθοδο (GET, POST, PUT, DELETE, PATCH), την URI (Uniform Resource Identifier) του πόρου που στοχεύει, και μπορεί να περιλαμβάνει δεδομένα στο σώμα του αιτήματος (body), ειδικά για τις μεθόδους POST, PUT και PATCH.
2. Κεφαλίδες Αιτήματος (Headers):
    - Το αίτημα περιλαμβάνει επίσης κεφαλίδες που μπορούν να περιέχουν πληροφορίες όπως τον τύπο περιεχομένου (Content-Type), πληροφορίες για τον αποστολέα, cookies, πληροφορίες εξουσιοδότησης και άλλα.
3. Αποστολή του Αιτήματος:
    - Ο client αποστέλλει το αίτημα στον server μέσω του διαδικτύου.
4. Επεξεργασία Αιτήματος από τον Server:
    - Ο server λαμβάνει το αίτημα και το επεξεργάζεται βάσει της καθορισμένης λογικής. Δεν διατηρεί καμία κατάσταση μεταξύ των αιτημάτων.
5. Απάντηση του Server (Response):
    - Μετά την επεξεργασία, ο server απαντάει με ένα HTTP αποκριτικό μήνυμα που περιλαμβάνει τον κωδικό κατάστασης (status code), κεφαλίδες απόκρισης και πιθανώς δεδομένα στο σώμα της απάντησης.
6. Επεξεργασία Απόκρισης από τον Client:
    - Ο client λαμβάνει την απάντηση και την επεξεργάζεται ανάλογα, είτε εμφανίζοντας τα δεδομένα στον χρήστη, είτε προβαίνοντας σε περαιτέρω ενέργειες βάσει των ληφθέντων δεδομένων.

Κρίσιμα Τεχνολογικά Στοιχεία:

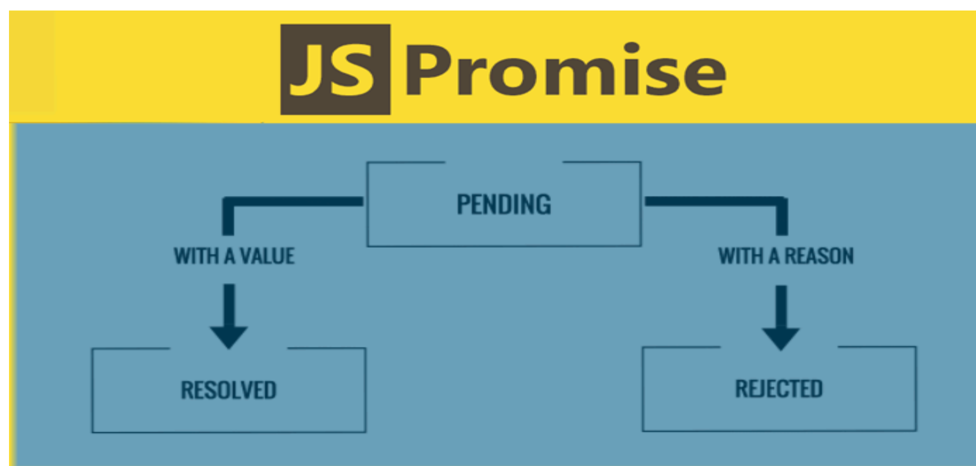
- Content Negotiation: Η διαδικασία όπου client και server συμφωνούν στον τύπο των δεδομένων που θα ανταλλάσσονται, όπως JSON ή XML.
- Εξουσιοδότηση και Ασφάλεια: Συχνά, τα APIs απαιτούν τεχνικές εξουσιοδότησης όπως το OAuth για να εξασφαλίσουν ότι μόνο εξουσιοδοτημένοι χρήστες ή συστήματα έχουν πρόσβαση στα δεδομένα.
- Ποσοστώσεις και Περιορισμοί: Πολλά APIs επιβάλλουν περιορισμούς στον αριθμό των αιτημάτων που μπορούν να γίνουν σε ένα συγκεκριμένο χρονικό διάστημα για να προστατεύσουν τον server από υπερφόρτωση.

2.7 Promise-Based HTTP Clients

Ένα Promise στη JavaScript είναι ένας αποκλειστικός μηχανισμός για τον χειρισμό ασύγχρονων εργασιών, όπως οι αιτήσεις προς έναν web server. Ένα Promise είναι ουσιαστικά ένα αντικείμενο που εκπροσωπεί την εκκρεμή ολοκλήρωση (ή την αποτυχία) μιας ασύγχρονης λειτουργίας, και το αποτέλεσμά της.[10]

2.7.1 Χρησιμότητα των Promise-Based Clients:

- Απλοποίηση Ασύγχρονου Κώδικα: Τα Promises απλοποιούν τη διαχείριση ασύγχρονων εργασιών, όπως οι HTTP αιτήσεις, αποφεύγοντας την ανάγκη για σύνθετες δομές callback.
- Επιτυχία ή Αποτυχία: Ένα Promise μπορεί να ολοκληρωθεί με "επιτυχία" (resolved) ή "αποτυχία" (rejected), επιτρέποντας τη διαχείριση και των δύο περιπτώσεων.
- Καλύτερη Διαχείριση Κώδικα: Τα Promises βοηθούν στη δημιουργία πιο καθαρού και διαχειρίσιμου κώδικα.



Εικόνα 2.3: JS Promise

2.7.2 Axios

Το Axios είναι ένας δημοφιλής HTTP client βασισμένος σε Promises[11]. Ξεκίνησε να γίνεται γνωστό γύρω στο 2016 και από τότε έχει αποκτήσει μεγάλη δημοτικότητα λόγω της ευελιξίας και της ευκολίας χρήσης του. Το Axios μεταξύ άλλων προσφέρει:

1. Εύκολη Διαχείριση Αιτημάτων και Αποκρίσεων: Το Axios διευκολύνει την αποστολή HTTP αιτημάτων και την επεξεργασία των αποκρίσεων μέσω Promises.
2. Δυνατότητες Ενσωμάτωσης και Παραμετροποίησης: Προσφέρει εκτεταμένες δυνατότητες παραμετροποίησης για αιτήματα, όπως headers, timeout, responseType και άλλα.
3. Αυτόματη Μετατροπή σε JSON: Μετατρέπει αυτόματα τις αποκρίσεις JSON σε JavaScript objects.
4. Διαχείριση Σφαλμάτων: Επιτρέπει την αποτελεσματική διαχείριση σφαλμάτων μέσω της απόρριψης Promises.

Η Άνοδος του Axios: Από την κυκλοφορία του το 2016, το Axios έχει γίνει ένας από τους πιο δημοφιλείς HTTP clients στην κοινότητα JavaScript. Σύμφωνα με πηγές όπως το GitHub και npm, το Axios έχει εκατομμύρια λήψεις κάθε μήνα.
Ευρεία Χρήση: Εκτιμάται ότι εκατοντάδες χιλιάδες έργα και εφαρμογές χρησιμοποιούν το Axios για την αποστολή HTTP αιτημάτων, κάνοντας το έναν από τους πλέον προτιμώμενους επιλογές ανάμεσα στους προγραμματιστές.
Η ανάγκη για έναν αξιόπιστο και ευέλικτο HTTP client όπως το Axios είναι ιδιαίτερα σημαντική στη σύγχρονη web ανάπτυξη, καθώς επιτρέπει την αποτελεσματική διασύνδεση μεταξύ των front-end και back-end συστημάτων με μεγάλη ευκολία και ευελιξία.

```
import axios from "axios";

export const mainAxios = axios.create({
  baseURL,
});

const login = async (username: string, password: string) => {
  setLoading(true);
  try {
    const res = await mainAxios.post("/User/login", { username, password });
  } catch (error) {
    console.log("Error in user login", error);
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 2.4: Παράδειγμα χρήσης Axios

2.8 JSON

JSON (JavaScript Object Notation) είναι ένα ελαφρύ μορφότυπο ανταλλαγής δεδομένων, το οποίο είναι εύκολο στο διάβασμα και την εγγραφή από τους ανθρώπους, καθώς και εύκολο στην ανάλυση

και την παραγωγή από τους υπολογιστές. Επιτρέπει την ανταλλαγή δεδομένων μεταξύ ενός server και ενός web client και χρησιμοποιείται ευρέως σε APIs και web εφαρμογές[12].

Χρησιμότητα των JSON στις Επικοινωνίες API-Web App

1. Συμβατότητα με τη JavaScript: Επειδή το JSON βασίζεται στη JavaScript, τα δεδομένα JSON μπορούν εύκολα να μετατραπούν σε JavaScript objects και αντίστροφα. Αυτό καθιστά το JSON ιδανικό για web εφαρμογές που χρησιμοποιούν JavaScript.
2. Ευκολία στην Ανάγνωση και Εγγραφή: Το JSON έχει μια σαφή και κατανοητή δομή, κάτι που διευκολύνει τους προγραμματιστές να το διαβάσουν και να το γράψουν, αυξάνοντας την παραγωγικότητα και μειώνοντας τα σφάλματα.
3. Ευκολία Ανάλυσης και Σειριοποίησης: Τα περισσότερα προγραμματιστικά περιβάλλοντα και γλώσσες προγραμματισμού παρέχουν ενσωματωμένες λειτουργίες για την ανάλυση (parsing) και τη σειριοποίηση (serialization) των δεδομένων JSON, επιτρέποντας την εύκολη μεταφορά δεδομένων μεταξύ client και server.
4. Αποδοτικότητα στην Μεταφορά Δεδομένων: Το JSON, λόγω της ελαφρότητας του, βελτιστοποιεί την απόδοση στην μεταφορά δεδομένων μέσω δικτύου, καθιστώντας το ιδανικό για web εφαρμογές που χρειάζονται γρήγορη φόρτωση και επεξεργασία δεδομένων.
5. Η χρήση των JSON στην επικοινωνία μεταξύ APIs και web εφαρμογών είναι ζωτικής σημασίας στη σύγχρονη ανάπτυξη λογισμικού. Το JSON παρέχει μια απλή, αποδοτική και ευέλικτη μέθοδο για την ανταλλαγή δεδομένων, που είναι απαραίτητη για την κατασκευή διαδραστικών και αποδοτικών web εφαρμογών.

```
{} sample.json > [ ] data
1  {
2      "data": [
3          {
4              "type": "articles",
5              "id": "1",
6              "attributes": {
7                  "title": "Working with JSON Data in python",
8                  "description": "This article explains the various ways to work with JSON data in python.",
9                  "created": "2020-12-28T14:56:29.000Z",
10                 "updated": "2020-12-28T14:56:28.000Z"
11             },
12             "author": {
13                 "id": "1",
14                 "name": "Aveek Das"
15             }
16         }
17     ]
18 }
```

Εικόνα 2.5: Παράδειγμα αντικειμένου JSON

2.9 Περιγραφή του δικού μας API

Για τη δική μας περίπτωση θα χρησιμοποιήσουμε ένα stateless API. Παρακάτω θα περιγράψουμε σύντομα την αρχιτεκτονική του.

2.9.1 API tables

Η βάση μας είναι χωρισμένη σε διάφορα tables. Έτσι ουσιαστικά διαφοροποιούνται τα entities της εφαρμογής μας. Οι πίνακες που χρησιμοποιούμε είναι οι εξείς:

1. **People:** Περιγράφει τα φυσικά πρόσωπα της εφαρμογής. Οποιοδήποτε φυσικό πρόσωπο αναφέρεται υπάρχει στη βάση, υπάρχει αυτόματα και σε αυτόν τον πίνακα.
2. **Venues:** Στον πίνακα αυτό ανήκουν όλοι οι θεατρικοί χώροι. Στον πίνακα αυτό υπάρχουν τα συνδεδεμένα με τον χώρο **events.**
3. **Productions:** Μια συγκεκριμένη θεατρική παράσταση. Πληροφορίες όπως τίτλος, περιγραφή, διάρκεια, φωτογραφίες και συντελεστές ανήκουν σε αυτόν τον πίνακα. Επίσης περιέχει διαθέσιμα events για αυτήν την παράσταση.
4. **Contribution:** Λεπτομέρειες για τη συμμετοχή ενός **Person,** σε ένα συγκεκριμένο **Production.** Αναφορά στο ρόλο του ατόμου.
5. **Organizer:** Στον πίνακα αυτόν περιλαμβάνονται οι διοργανωτές των **events.** Μπορούμε να αντλήσουμε διάφορες πληροφορίες όπως, την επωνυμία της εταιρείας, στοιχεία διεύθυνσης και επικοινωνίας, ακόμη και το ΑΦΜ.
6. **Roles:** Οι διαθέσιμοι ρόλοι που μπορεί να έχει κάποιος που ανήκει στον πίνακα **People.** Για παράδειγμα ηθοποιός, σκηνοθέτης, τεχνικός ήχου, μακιγιάζ κλπ.
7. **Transactions:** Ο πίνακας αυτός κρατάει στοιχεία για μια συναλλαγή ανάμεσα στο σύστημα και έναν χρήστη της εφαρμογής. Πληροφορίες όπως η αξία της συναλλαγής, η ημερομηνία και ο λόγος για τον οποίο έλαβε θέση η εκάστοτε συναλλαγή.
8. **AccountRequests:** Εδώ αποθηκεύονται διάφορες πληροφορίες όταν ένας χρήστης κάνει την κατάλληλη διαδικασία για να πραγματοποιήσει ένα **claim,** για ένα συγκεκριμένο προφίλ**(person).** Δεδομένα όπως το εν λόγω προφίλ, την κατάσταση της αίτησης και των απαιτούμενων εγγράφων, καθώς και ποιος **admin** είναι υπεύθυνος για την αίτηση αυτή.
9. **User:** Τελευταίος και πιο σημαντικός πίνακας για την εργασία αυτή. Εδώ υπάρχουν πολλές πληροφορίες για τον εκάστοτε χρήστη της εφαρμογής, αφού δημιουργήσει λογαριασμό και μετά.

```
export interface User {
  id: number;
  email: string;
  emailVerified: boolean;
  _2FA_enabled: boolean;
  role: string;
  performerRoles: string[];
  transactions?: Transaction[];
  balance: number;
  facebook?: string;
  instagram?: string;
  youtube?: string;
  profilePhoto?: UserPhoto;
  bioPdfLocation?: string;
  userImages: UserPhoto[];
  phoneNumber?: string;
  phoneNumberVerified?: boolean;
  claimedPerson: Person;
  claimedVenues: Venue[];
  claimedEvents: Event[];
}
```

Εικόνα 2.6: Ο πίνακας ενός User

2.9.2 Πίνακας User

Ο τελευταίος πίνακας που αναφέραμε είναι αυτός με τον οποίο ασχοληθήκαμε κατά κύριο λόγο στην εργασία αυτή. Τεράστιο κομμάτι της εργασίας ήταν να μετατρέψουμε την εφαρμογή από μία απλή ενημερωτική σελίδα στην οποία ένας επισκέπτης μπορούσε να δει πληροφορίες για παραστάσεις ηθοποιούς και θέατρα, σε ένα σύγχρονο web app με το οποίο μπορεί να αλληλεπιδράσει σε αληθινό χρόνο. Σε αυτό το υποκεφάλαιο θα μιλήσουμε κυρίως για τις μεθόδους που χρειάστηκε να προσθέσουμε και θα αναλύσουμε σε επόμενο κεφάλαιο πως υλοποιήθηκαν στην εφαρμογή μας.

2.9.2.1 Authentication

Πρώτο βήμα για να γίνει μια εφαρμογή user-related, είναι το να υπάρχει ασφαλής επικοινωνία ανάμεσα στην εφαρμογή μας και το API που χρησιμοποιούμε. Για να το καταφέρουμε αυτό πρέπει το API να γνωρίζει ότι το αίτημα έρχεται από έναν επαληθευμένο χρήστη. Στην περίπτωσή μας, που το API μας είναι stateless, έχουμε επιλέξει το JWT token exchange ως τρόπο επαλήθευσης των απαιτούμενων calls στο API.
Δεν είναι όλα τα calls μας απαραίτητο να είναι επαληθευμένα, καθώς κάποιος χρήστης μπορεί απλά να επισκεφτεί τη σελίδα μας και να δει γενικές πληροφορίες χωρίς να θέλει να κάνει κάποια ενέργεια που είναι user-related. Περισσότερες λεπτομέρειες για τη διαδικασία του JWT θα δούμε στο **κεφάλαιο 3**.

2.9.2.2 User Context

Μια επίσης μεγάλη αλλαγή στην εφαρμογή μας είναι η χρήση ενός νέου context που καλύπτει ουσιαστικά ολόκληρο το app. Όπως θα δείτε και παρακάτω για τη χρήση του React Context Provider

πιο αναλυτικά, θέλουμε πλέον η εφαρμογή μας να είναι δυναμική και να αντιδρά σε διάφορα actions του χρήστη. Για το λόγο αυτό με τη χρήση του User context, ουσιαστικά παρακολουθούμε αλλαγές σχετικές με το χρήστη σε κάθε βήμα της εφαρμογής. Αν για παράδειγμα ο χρήστης πραγματοποιεί μια συναλλαγή, πρέπει να είμαστε συνεχώς συγχρονισμένοι με τη βάση δεδομένων μας για υπόλοιπο του χρήστη στο λογαριασμό του και άλλες διάφορες πολύ ευαίσθητες πληροφορίες. Καθώς το API μας ειναι stateless, πρεπει να φροντίζουμε για όλη αυτή τη διαδικασία αυτόματα.

2.9.2.3 User relations

Πολλοί από τους πίνακες που είδαμε πριν, έχουν πλέον ενημερωθεί και στο κομμάτι του frontend ώστε να αντικατοπτρίζουν την πιθανή σχέση τους με έναν χρήστη της εφαρμογής. Για παράδειγμα, ένας χρήστης μπορεί να κάνει claim έναν θεατρικό χώρο και αργότερα να αποφασίσει να αλλάξει το όνομα του. Είτε αντίστοιχα σε μια θεατρική παράσταση να προσθέσει πληροφορίες που λείπουν ή να αλλάξει υπάρχουσες. Η ίδια λογική ισχύει και για τον πίνακα **People.**

2.10 Επίλογος

Στο κεφάλαιο αυτό περιγράψαμε τη σημασία που υπάρχει ανάμεσα στην επικοινωνια ενός web app με το backend και τη βάση δεδομένων με τη χρήση των APIs. Αναλύσαμε σύντομα απαραίτητες τεχνολογίες καθώς και τη δομή των δικών μας δεδομένων που θα χρησιμοποιήσουμε στην ανάπτυξη της εφαρμογής μας.

Κεφάλαιο 3ο: Τεχνολογίες εργασίας

3.1 Εισαγωγή

Στο τρίτο κεφάλαιο θα γίνει μια αναλυτική προσέγγιση στις τεχνολογίες που χρησιμοποιήσαμε στην υλοποίηση της ΔΕ. Θα εξηγήσουμε ποιες επιλογές έγιναν και για ποιον λόγο και θα προσπαθήσουμε να ρίξουμε μια ματιά στο τι θα μπορούσε να γίνει καλύτερα ή να βελτιωθεί από τεχνολογικής άποψης στο μέλλον.

3.2 Package Management

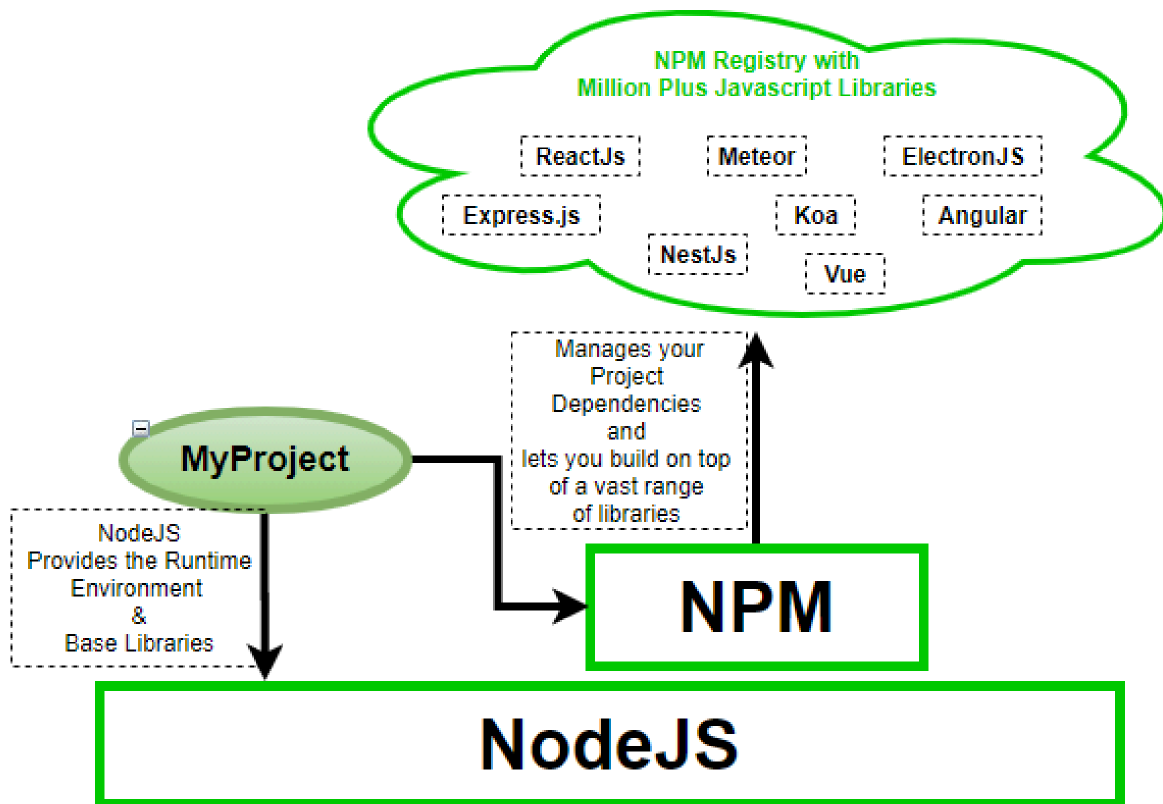
Το package management είναι μια κρίσιμη έννοια στην ανάπτυξη λογισμικού, που αφορά τη διαχείριση των λεγόμενων "πακέτων" ή "modules" - συλλογές κώδικα που εκτελούν συγκεκριμένες λειτουργίες. Ένας package manager είναι ένα εργαλείο που διευκολύνει την εγκατάσταση, αναβάθμιση, διαμόρφωση και απεγκατάσταση πακέτων από ένα λογισμικό. Στον κόσμο της προγραμματιστικής, αυτό συνήθως αφορά την ανάπτυξη και τη διαχείριση βιβλιοθηκών κώδικα ή εξαρτήσεων.

3.2.1 Το npm ως Package Manager

Το npm (Node Package Manager) είναι ένας από τους πιο δημοφιλείς package managers στον κόσμο της JavaScript. Αρχικά σχεδιασμένο για τη διαχείριση εξαρτήσεων στο Node.js, το npm έχει εξελιχθεί σε ένα ολοκληρωμένο εργαλείο για τη διαχείριση JavaScript πακέτων και σεναρίων ανάπτυξης.[13]

3.2.2 Γιατί Χρησιμοποιείται το npm;

1. Διαχείριση Εξαρτήσεων: Το npm καθιστά εύκολη την εγκατάσταση, αναβάθμιση και απεγκατάσταση βιβλιοθηκών και πακέτων, διασφαλίζοντας την εύκολη διαχείριση εξαρτήσεων σε ένα project.
2. Ευρύ Οικοσύστημα: Με χιλιάδες διαθέσιμα πακέτα, το npm παρέχει ένα τεράστιο οικοσύστημα βιβλιοθηκών και εργαλείων για κάθε σχεδόν ανάγκη ανάπτυξης.
3. Συμβατότητα και Πρότυπα: Το npm διασφαλίζει τη συμβατότητα των πακέτων και υποστηρίζει σύγχρονα πρότυπα ανάπτυξης.
4. Αυτοματοποίηση: Παρέχει δυνατότητες αυτοματοποίησης και διαμόρφωσης σεναρίων ανάπτυξης, βελτιώνοντας τη διαδικασία ανάπτυξης.
5. Βασικές Οδηγίες Χρήσης του npm:
6. Εγκατάσταση πακέτων: Μέσω της εντολής ***npm install <package-name>***, μπορείτε να εγκαταστήσετε τις εξαρτήσεις που χρειάζεστε.
7. Διαχείριση εξαρτήσεων: Με το αρχείο package.json, μπορείτε να διαχειρίζεστε και να παρακολουθείτε τις εξαρτήσεις του project σας.
8. Σενάρια και Scripts: Το npm επιτρέπει τη δημιουργία custom scripts για την αυτοματοποίηση συχνών εργασιών όπως η δοκιμαστική εκτέλεση ή η δημιουργία του build.



Εικόνα 3.1: Χρήση του NPM

Στο σύγχρονο περιβάλλον ανάπτυξης λογισμικού, το npm αποτελεί μια θεμελιώδη εργαλειοθήκη για κάθε JavaScript developer, παρέχοντας τη δυνατότητα αποδοτικής διαχείρισης και ανάπτυξης των project τους.
Η διατήρηση όλων των dependencies και των εξωτερικών libraries που χρησιμοποιούνται σε μια εφαρμογή, καθώς και διάφορα scripts, όπως για παράδειγμα το **build, dev, serve** κλπ, διατηρούνται και ρυθμίζονται σε ένα **package.json** αρχείο. Το αρχείο αυτό έχει τη μορφή ενός απλού json αντικειμένου και συνεργάζεται με το εκάστοτε package manager που χρησιμοποιούμε για την εγκατάσταση ή και την ενημέρωση των dependencies.

TODO: maybe add more info, about versioning, peer dependencies etc.

```
package.json ×
package.json > {} dependencies
You, last month | 2 authors (Dima and others)
1   {
2     "name": "theatrica-frontend",
3     "version": "0.1.0",
4     "private": true,
5     "dependencies": {
6       "@date-io/date-fns": "^1.3.13",
7       "@emotion/react": "^11.11.1",
8       "@emotion/styled": "^11.11.0",
9       "@material-ui/core": "^4.11.3",
10      "@material-ui/icons": "^4.11.2",
11      "@material-ui/lab": "^4.0.0-alpha.57",
12      "@material-ui/pickers": "^3.3.10",
13      "@mui/icons-material": "^5.14.16",
14      "@nivo/calendar": "^0.79.1",
15      "@nivo/colors": "^0.79.1",
16      "@nivo/pie": "^0.79.1",
17      "@testing-library/jest-dom": "^5.11.9",
18      "@testing-library/react": "^11.2.5",
19      "@testing-library/user-event": "^12.8.1",
20      "algoliasearch": "^4.12.0",
21      "autoprefixer": "^10.4.16",
22      "axios": "^0.21.1",
```

Εικόνα 3.2: Παράδειγμα ενός **package.json** αρχείου

3.3 Web scripting.

Το Web-Scripting αφορά τη διαδικασία δημιουργίας και ενσωμάτωσης scripts σε μια ιστοσελίδα. Ένα script είναι μια λίστα εντολών που εκτελείται από ένα συγκεκριμένο πρόγραμμα ή scripting engine.[14]

Τα scripts γράφονται σε γλώσσες προγραμματισμού γνωστές ως scripting languages, όπως JavaScript, VBScript, PHP, ASP, PERL και JSP.

Τα scripts διακρίνονται σε δύο κατηγορίες:

1. **Client-Side Scripts:**
   a. Ευθύνονται για την αλληλεπίδραση εντός μιας ιστοσελίδας.
   b. Εκτελούνται στην πλευρά του πελάτη (client) και από τον browser του.
   c. Είναι εξαρτημένα από τον browser και χρησιμοποιούνται για διάφορες λειτουργίες, όπως τη συλλογή δεδομένων από τον χρήστη ή τον προσαρμογή της παρουσίασης της σελίδας.
   d. Παραδείγματα: JavaScript, VBScript.
2. **Server-Side Scripts:**

a. Ευθύνονται για εργασίες που πραγματοποιούνται στον server και στη συνέχεια στέλνουν τα αποτελέσματα στον πελάτη.
b. Είναι ανεξάρτητα από τον browser του πελάτη, καθώς ο server αναλαμβάνει την εργασία.
c. Χρησιμοποιούνται για λειτουργίες όπως η προστασία με κωδικό ή η επεξεργασία φορμών.
d. Παραδείγματα: PHP, ASP, JSP.

3.3.1 Javascript

Το 1995, ο Brendan Eich, ενώ εργαζόταν στη Netscape, δημιούργησε τη γλώσσα προγραμματισμού JavaScript μέσα σε μόλις δέκα ημέρες. Από τότε, έχει γίνει μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού για τον ιστό.[3]

- **Dynamic**: Στο JavaScript, ο τύπος της μεταβλητής μπορεί να αλλάξει κατά τη διάρκεια της εκτέλεσης. Π.χ., μια μεταβλητή που αρχικά ήταν αριθμός μπορεί να γίνει αργότερα συμβολοσειρά.
- **Interactive**: Επιτρέπει τη διαδραστικότητα στις ιστοσελίδες, όπως pop-ups ή animations.
- **First-Class Functions:** Στο JavaScript, οι συναρτήσεις είναι "first-class citizens", πράγμα που σημαίνει ότι μπορούν να αντιμετωπίζονται όπως οι μεταβλητές. Για παράδειγμα, μια συνάρτηση μπορεί να περαστεί ως όρισμα σε μια άλλη συνάρτηση.

3.3.2 Typescript

Η **TypeScript** δημιουργήθηκε από τη Microsoft το 2012 έως ένα υπερσύνολο της JavaScript.[15]

Βασικά Χαρακτηριστικά του TypeScript:

1. **Static Typing:** Όπως αναφέρατε, οι μεταβλητές στη TypeScript δηλώνονται με συγκεκριμένο τύπο. Αυτό επιτρέπει την ανίχνευση σφαλμάτων σχετικών με τύπους κατά τη φάση της συγγραφής κώδικα.
2. **Classes & Interfaces:** Η TypeScript εισάγει την έννοια των classes και interfaces, προσθέτοντας ένα στρώμα τυποποίησης και δομής που βοηθά στην οργάνωση του κώδικα και στην υλοποίηση σχεδιαστικών προτύπων.
3. **Development Tools:** Με τη χρήση του TypeScript compiler, οι προγραμματιστές μπορούν να εντοπίζουν λάθη πριν ακόμα τρέξουν τον κώδικα, βελτιώνοντας την αξιοπιστία και την ποιότητα του λογισμικού.
4. **Συμβατότητα με JavaScript:** Κώδικας JavaScript μπορεί να ενσωματωθεί απευθείας σε ένα TypeScript project, καθώς ο κώδικας TypeScript μεταγλωττίζεται σε JavaScript. Αυτό διευκολύνει την μετάβαση από JavaScript σε TypeScript.
5. Ενίσχυση Της Ανάπτυξης: Η TypeScript βοηθά στην ανάπτυξη μεγάλων εφαρμογών και συστημάτων, παρέχοντας ένα πιο αυστηρό και δομημένο πλαίσιο.

3.3.2.1 Εγκατάσταση TypeScript

Η εγκατάσταση της TypeScript σε ένα υπάρχον project είναι μια απλή διαδικασία:

1. Εγκατάσταση μέσω npm:
   - Χρησιμοποιήστε την εντολή npm install typescript --save-dev για να προσθέσετε τη TypeScript ως μια εξάρτηση ανάπτυξης στο project σας.
2. Δημιουργία ή Ενημέρωση του tsconfig.json:
   - Αυτό το αρχείο ρυθμίσεων χρησιμοποιείται για να καθορίσει πώς ο TypeScript compiler θα μεταγλωττίζει τον κώδικα σε JavaScript. Μπορείτε να δημιουργήσετε αυτό το αρχείο χρησιμοποιώντας την εντολή tsc --init.

```
tsconfig.json ×
tsconfig.json > ...
You, 5 months ago | 1 author (You)
1  {
2    "compilerOptions": {
3      "target": "esnext",
4      "lib": ["dom", "dom.iterable", "esnext"],
5      "module": "esnext",
6      "strict": true,
7      "esModuleInterop": true,
8      "skipLibCheck": true,
9      "forceConsistentCasingInFileNames": true,
10     "moduleResolution": "node",
11     "resolveJsonModule": true,
12     "isolatedModules": true,
13     "jsx": "preserve",
14     "allowJs": true,
15     "noEmit": true,
16     "incremental": true
17   },
18   "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx"],
19   "exclude": ["node_modules", ".next"]
20 }
```

Εικόνα 3.3: Παράδειγμα **tsconfig.json** αρχείου

3. Μετατροπή .js αρχείων σε .ts:
   - Για να αρχίσετε να χρησιμοποιείτε τη TypeScript, μετονομάστε τα αρχεία JavaScript (.js) σας σε TypeScript (.ts). Στη συνέχεια, αρχίστε να προσθέτετε τύπους στις μεταβλητές και τις συναρτήσεις σας.
4. Κατασκευή (Build) του Έργου:
   - Χρησιμοποιήστε την εντολή tsc για να μεταγλωττίσετε τον κώδικα TypeScript σε JavaScript. Οι ρυθμίσεις της μεταγλώττισης ορίζονται στο αρχείο tsconfig.json.

3.3.3 Javascript vs Typescript

Το TypeScript επεκτείνει τη JavaScript, προσθέτοντας δυνατότητες πληκτρολόγησης και άλλες λειτουργίες.

Πλεονεκτήματα της TypeScript έναντι της JavaScript:

1. Type Safety: Στο TypeScript, τα λάθη σχετικά με τους τύπους μεταβλητών ανακαλύπτονται κατά την ανάπτυξη. Π.χ., αν δοκιμάσετε να ορίσετε μια μεταβλητή number ως συμβολοσειρά, ο compiler θα σας ενημερώσει για το λάθος.
2. Code Management: Η προσθήκη τύπων και διεπαφών βοηθά στην οργάνωση και διαχείριση του κώδικα, ειδικά σε μεγάλες εφαρμογές.
3. Enhanced Documentation: Οι τύποι μεταβλητών προσφέρουν αυτόματη τεκμηρίωση για τον κώδικα.
4. Ωστόσο, η επιλογή μεταξύ των δύο γλωσσών εξαρτάται από τις ανάγκες του έργου και τις προτιμήσεις του προγραμματιστή.

3.4 JWT Authentication

Στον τομέα της ανάπτυξης ιστού και της ανταλλαγής δεδομένων, η πιστοποίηση JSON (JavaScript Object Notation) διαδραματίζει κεντρικό ρόλο στη διασφάλιση της επικοινωνίας μεταξύ συστημάτων πελάτη και εξυπηρετητή. Αυτό το κεφάλαιο έχει ως στόχο να παρέχει μια επισκόπηση της πιστοποίησης JSON, της προέλευσής της, των βασικών αρχών της και μιας κριτικής εξέτασης των θετικών και αρνητικών πτυχών της.[16]

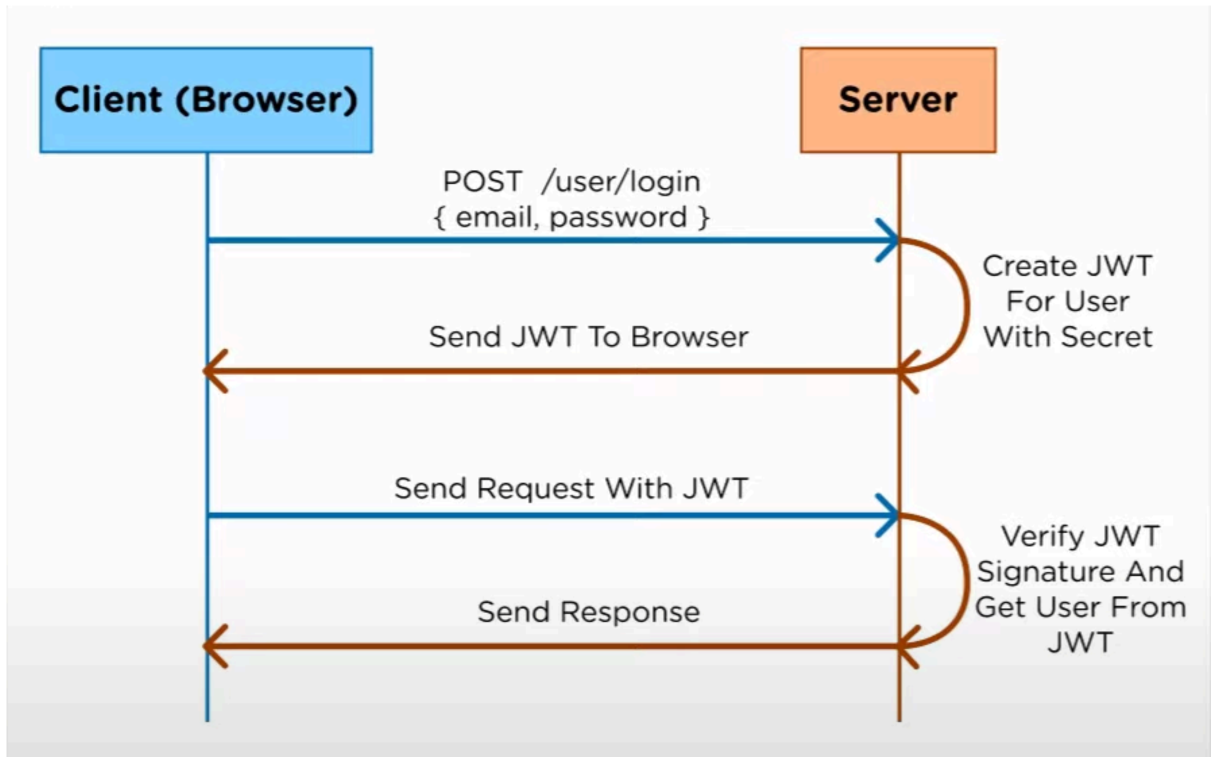
3.4.1 Προέλευση και εξέλιξη:

Η πιστοποίηση JSON προέκυψε ως απάντηση στην αυξανόμενη ανάγκη για ένα ελαφρύ, ευανάγνωστο από ανθρώπους και εύκολο στον αναλυτικό προγραμματισμό μορφή ανταλλαγής δεδομένων. Ο Douglas Crockford παρουσίασε το JSON στις αρχές της δεκαετίας του 2000, βελτιώνοντας τη σύνταξή του και την απλότητά του ως υποσύνολο της JavaScript. Η υιοθέτησή του έχει εδραιωθεί από τις αρχές της ανάπτυξης ιστού για να γίνει ένα αποδεκτό πρότυπο για την ανταλλαγή δεδομένων. Η απλότητα της σύνταξης του JSON καθιστά εύκολη την ανάγνωση και την εγγραφή τόσο από ανθρώπους όσο και από μηχανές. Η ελαφρότητά του μειώνει το πρόσθετο βάρος δεδομένων κατά τη μετάδοση, συντελώντας στην αποτελεσματική επικοινωνία.

3.4.2 Πως λειτουργεί το JWT

1. Παραγωγή διακριτικού: Κατά την πιστοποίηση, ένας εξυπηρετητής δημιουργεί ένα JSON Web Token (JWT) που περιέχει πληροφορίες όπως το αναγνωριστικό χρήστη, ο χρόνος λήξης και μια ψηφιακή υπογραφή. Το διακριτικό αυτό στέλνεται στον πελάτη.

2. Επιβεβαίωση διακριτικού: Ο client συμπεριλαμβάνει το ληφθέν διακριτικό σε μελλοντικά αιτήματα. Οι εξυπηρετητές επιβεβαιώνουν την αυθεντικότητα και την ακεραιότητα του διακριτικού ελέγχοντας την ψηφιακή υπογραφή, διασφαλίζοντας ότι τα δεδομένα δεν έχουν παραβιαστεί.
3. Stateless: Η πιστοποίηση JSON λειτουργεί βασιζόμενη σε ανυπαρξία κατάστασης, πράγμα που σημαίνει ότι κάθε αίτημα από τον πελάτη πρέπει να περιλαμβάνει όλες τις απαραίτητες πληροφορίες για τον εξυπηρετητή προκειμένου να πιστοποιήσει και να εξουσιοδοτήσει τον χρήστη, μειώνοντας την εξάρτηση από δεδομένα συνεδρίας και ενισχύοντας την κλιμακωσιμότητα.



Εικόνα 3.4: Πως λειτουργεί το JWT

Στην ανάλυση της υλοποίησης της εφαρμογής μας στο Κεφάλαιο 4 θα εξηγήσουμε πως χρησιμοποιήσαμε την αυθεντικοποίηση μέσω JWT για την επικοινωνία με το API μας σε user-related requests.

3.5 Axios Interceptors

Οι Axios Interceptors είναι ένα ισχυρό χαρακτηριστικό που παρέχεται από τη βιβλιοθήκη Axios, ένα δημοφιλές HTTP client για JavaScript. Οι interceptors επιτρέπουν την παρεμβολή πριν την αποστολή ενός αιτήματος ή πριν τη λήψη μιας απάντησης από τον server, προσφέροντας έτσι τη δυνατότητα προσαρμογής των αιτημάτων, των αποκρίσεων και της διαχείρισης σφαλμάτων.[17]

3.5.1 Κατηγορίες Interceptors

Οι interceptors χωρίζονται σε δύο κατηγορίες τις οποίες θα αναλύσουμε παρακάτω:

1. **Request Interceptors**
    - Χρησιμοποιούνται για την τροποποίηση του αιτήματος πριν αυτό φτάσει στον server.

   ○ Μπορείτε να προσθέσετε headers, να τροποποιήσετε τα δεδομένα του αιτήματος, ή να εκτελέσετε άλλες λειτουργίες.
2. **Response Interceptors**
   ○ Χρησιμοποιούνται για την επεξεργασία της απόκρισης από τον server πριν την επιστροφή στον κώδικα που την κάλεσε.
   ○ Μπορείτε να τροποποιήσετε την απόκριση, να διαχειριστείτε σφάλματα, ή να εκτελέσετε άλλες λειτουργίες.

3.5.2 Παραδείγματα Χρήσης

Παρακάτω θα δείξουμε σύντομα πως δημιουργούνται και χρησιμοποιούνται με πολύ απλό τρόπο οι axios interceptors:

1. **Δημιουργία ενός Request Interceptor:**

```
axios.interceptors.request.use(request => {
  console.log('Starting Request', JSON.stringify(request, null, 2))
  return request
})
```

Εικόνα 3.5: Παράδειγμα χρήσης request interceptor

Στο παράδειγμα αυτό, προστίθεται ένας interceptor που εμφανίζει τα στοιχεία ενός εξερχόμενου αιτήματος στην κονσόλα πριν αυτό αποσταλεί. Σε μια πραγματική χρήση όπως θα δούμε στο κεφάλαιο 4, μπορούμε να ελέγξουμε πλήρως ένα request προτού αυτό αποσταλεί, συμπεριλαμβανομένου και των κεφαλίδων του.

2. **Δημιουργία ενός Response Interceptor:**

```
axios.interceptors.response.use(response => {
  console.log('Receiving Response', JSON.stringify(response, null, 2))
  return response;
}, error => {
  // Διαχείριση σφαλμάτων
  return Promise.reject(error);
});
```

Εικόνα 3.6: Παράδειγμα χρήσης response interceptor

Σε αυτό το παράδειγμα, ο response interceptor καταγράφει την απόκριση στην κονσόλα. Επίσης, παρέχει μια βασική διαχείριση σφαλμάτων. Στο κεφάλαιο 4 θα δείξουμε πως μπορούμε να χρησιμοποιήσουμε ένα τέτοιο interceptor για τη συνεχή ενημέρωση του χρήστη(επιτυχής ενέργεια, σφάλμα σύνδεσης κλπ).

3.5.3 Πότε Χρησιμοποιείται ο Axios Interceptor

Οι Axios Interceptors είναι χρήσιμοι σε περιπτώσεις όπως:

- Η αυτόματη ενημέρωση των headers αιτήματος (π.χ., τοποθέτηση των JWT tokens).

- Η προσαρμογή των δεδομένων της απόκρισης πριν φτάσουν στον κώδικα της εφαρμογής.
- Η κεντρική διαχείριση σφαλμάτων και ανακατευθύνσεων.

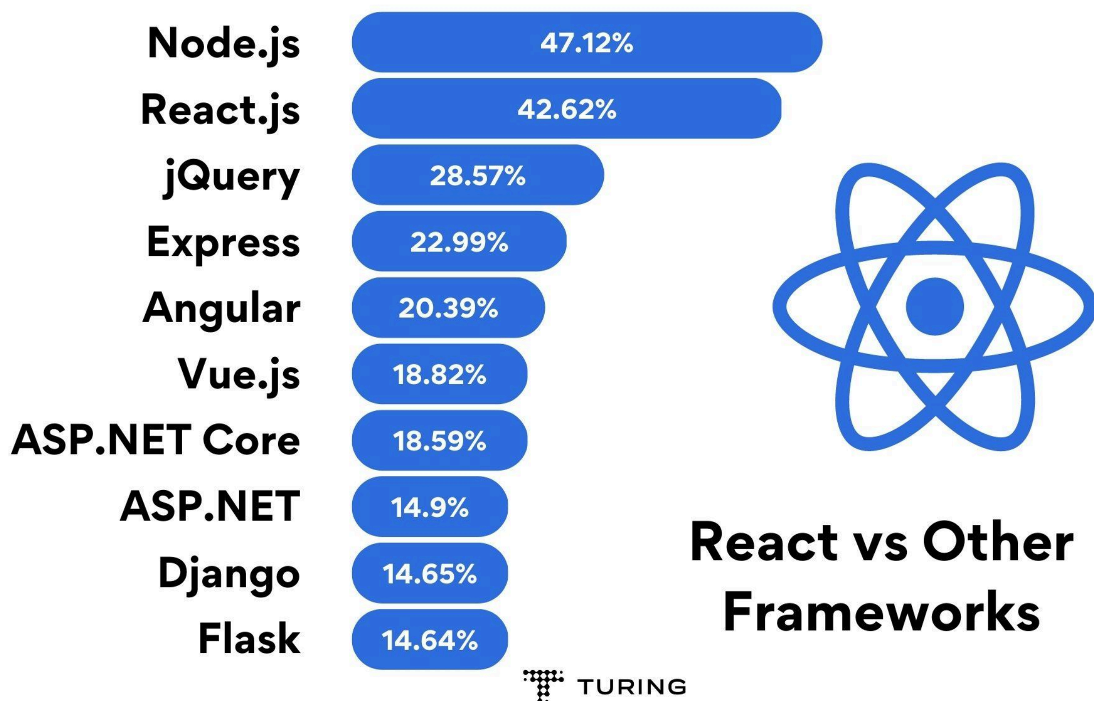
3.5.4 Θετικά της Χρήσης Axios Interceptors

- Κεντρική Διαχείριση: Επιτρέπει την κεντρική διαχείριση των αιτημάτων και των αποκρίσεων, μειώνοντας την επανάληψη κώδικα.
- Ευελιξία: Παρέχει τη δυνατότητα προσαρμογής των αιτημάτων και των αποκρίσεων σε πολλαπλά σημεία της εφαρμογής.
- Καλύτερη Διαχείριση Σφαλμάτων: Διευκολύνει την κεντρική και συνεπή διαχείριση σφαλμάτων.

Συνοψίζοντας, οι Axios Interceptors αποτελούν ένα πολύτιμο εργαλείο στη διαχείριση HTTP αιτημάτων και αποκρίσεων, προσφέροντας ευελιξία, κεντρική διαχείριση και βελτιωμένη διαχείριση σφαλμάτων.

3.5 React framework

Όπως αναφέραμε το web app αυτό είναι γραμμένο σε React. Η React είναι ένα σύγχρονο framework, τρομερά διαδεδομένο. Οι περισσότερες σύγχρονες ιστοσελίδες και web εφαρμογές είναι γραμμένες σε React. [18]



Εικόνα 3.7: Η React στη λίστα των frameworks

Η React είναι ένα δημοφιλές JavaScript library που αναπτύχθηκε από την Facebook για την κατασκευή διεπαφών χρήστη (user interfaces) σε web εφαρμογές. Επιτρέπει την ανάπτυξη επαναχρησιμοποιήσιμων UI components, παρέχοντας μια δυναμική και αποδοτική εμπειρία ανάπτυξης. Βασικά χαρακτηριστικά της React:

1. **JSX (JavaScript XML):** Το React χρησιμοποιεί JSX, μια σύνταξη που επιτρέπει την εγγραφή HTML μέσα σε JavaScript. Αυτό διευκολύνει την ανάπτυξη διεπαφών και την οπτική κατανόηση του κώδικα.[19]
2. **Components:** Το React βασίζεται στην ιδέα των επαναχρησιμοποιήσιμων components, που είναι ανεξάρτητα και αυτοτελή τμήματα κώδικα. Υπάρχουν δύο τύποι components: Class Components και Function Components. Περιλαμβάνονται και τα **Arrow Function Components.** Αυτά τα components είναι μια συντομότερη σύνταξη για τα Function Components. Χρησιμοποιούν την arrow function (=>) της ES6 και είναι λιγότερο περίπλοκα από τα Class Components.[20]
3. **State:** Το state στο React είναι ένας τρόπος αποθήκευσης πληροφοριών για τα components. Μπορεί να αλλάζει με τον καιρό και όταν αλλάζει, το component επαναφορτώνεται.[21]
4. **Hooks:** Εισαγωγή με την έκδοση 16.8, οι Hooks επιτρέπουν τη χρήση state και άλλων React features σε Function Components. Τα πιο συνηθισμένα είναι τα useState και useEffect.[22]
5. **Κύκλος Ζωής του Component (Lifecycle):** Τα Class Components του React έχουν διάφορες "φάσεις" ζωής, όπως το mounting, updating και unmounting. Κάθε φάση έχει διαφορετικές μεθόδους που μπορούν να εκτελεστούν.[23]
6. **Context και ContextProvider**: Χρησιμοποιούνται για να περνάμε δεδομένα σε global επίπεδο ή γενικότερα σε ένα επίπεδο που δεν είναι δυνατή ή επιθυμητή, η διαμοίραση δεδομένων μέσω των components και των props.[24]

3.5.1 JSX

- Το JSX είναι μια επέκταση σύνταξης για JavaScript, που επιτρέπει την γραφή HTML στον JavaScript κώδικα. Είναι ουσιαστικά ένας συνδυασμός JavaScript και HTML.
- Χρησιμοποιείται για να διευκολύνει τη δημιουργία των React components, καθιστώντας τον κώδικα πιο ευανάγνωστο και διαχειρίσιμο.
- Το JSX μετατρέπεται σε JavaScript κατά τη διάρκεια της εκτέλεσης με τη βοήθεια ενός compiler (όπως το Babel).

3.5.2 Components

Τα Components αποτελούν τον πυρήνα της βιβλιοθήκης React και είναι τα βασικά κτίρια μπλοκ για τη δημιουργία διεπαφών χρήστη. Ένα React component είναι μια ανεξάρτητη, επαναχρησιμοποιήσιμη μονάδα που περιέχει HTML και JavaScript και ελέγχει ένα κομμάτι της διεπαφής χρήστη (UI).[20]

1. Class Components
    - Τα Class Components είναι πιο παραδοσιακά στο React και δημιουργούνται χρησιμοποιώντας την ES6 class syntax.
    - Παρέχουν πρόσβαση σε lifecycle methods και επιτρέπουν τη διαχείριση του state.

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Εικόνα 3.8: React Class Component

2. Function Components
   - Τα Function Components είναι πιο απλά και συντομότερα από τα Class Components.
   - Δεν έχουν πρόσβαση στα lifecycle methods και δεν μπορούν να χρησιμοποιήσουν state χωρίς τη χρήση hooks.

```
function TabPanel(props: TabPanelProps) {        You, 4 mont
  return (
    <div hidden={props.value !== props.index} {...props}>
      {props.value === props.index && props.children}
    </div>
  );
}
```

Εικόνα 3.9: React Function Component

3. Arrow Function Components
   - Μια συντομευμένη μορφή των Function Components χρησιμοποιώντας την arrow function της ES6.
   - Παρέχουν μια πιο συμπαγή σύνταξη.

```
const ArtistCard: React.FC<ArtistCardProps> = ({
  id,
  fullname,
  image,
  isClaimed,
  isDetails,
}) => {
  const classes = useStyles();
  const theme = useTheme();
  return (···
  );
};
```

Εικόνα 3.10: React Arrow Function Component

Σημασία των Components:

- Επαναχρησιμοποίηση και Οργάνωση: Τα Components επιτρέπουν την επαναχρησιμοποίηση του κώδικα, καθιστώντας την ανάπτυξη πιο αποδοτική και τον κώδικα πιο οργανωμένο.
- Διαχωρισμός Ανησυχιών: Κάθε component ελέγχει ένα συγκεκριμένο κομμάτι της UI, βοηθώντας στο διαχωρισμό των ανησυχιών και στην απλοποίηση της ανάπτυξης.
- Ευελιξία: Τα Components μπορούν να δεχτούν δεδομένα ως props, επιτρέποντας τους να είναι δυναμικά και να ανταποκρίνονται σε αλλαγές στα δεδομένα.

Τα React Components αποτελούν τη βάση για τη δημιουργία διαδραστικών και αποδοτικών web εφαρμογών, παρέχοντας την ευελιξία και τη δυνατότητα ανάπτυξης σύνθετων UI δομών με απλό και καθαρό τρόπο.

3.5.3 State

- Το State είναι ένα αντικείμενο που αποθηκεύει τις πληροφορίες σχετικά με την κατάσταση ενός component.
- Επιτρέπει την αποθήκευση και την παρακολούθηση δεδομένων που μπορούν να αλλάξουν με τον καιρό. Όταν το state αλλάζει, το component ανανεώνεται αυτόματα.

3.5.4 Hooks

- Τα Hooks είναι μια νέα προσθήκη στο React 16.8, που επιτρέπουν τη χρήση state και άλλων React χαρακτηριστικών σε function components.
- Τα **useState** και **useEffect** είναι δύο βασικά hooks. Το useState επιτρέπει την προσθήκη state σε function components, ενώ το useEffect είναι για τη διαχείριση παρενεργειών (side-effects) στα components.

3.5.4.1 useState

Το useState είναι ένα hook που επιτρέπει σε function components να έχουν δικό τους state.

```
const [state, setState] = useState(initialState);
```

Εικόνα 3.11: Σύνταξη του useState

- **state** είναι η τρέχουσα κατάσταση.
- **setState** είναι η συνάρτηση που αλλάζει την κατάσταση.
- **initialState** είναι η αρχική κατάσταση του state.

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Μετρητής: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Πρόσθεσε
      </button>
    </div>
  );
}
```

Εικόνα 3.12: Παράδειγμα χρήση; του useState

Στο παράδειγμα αυτό, έχουμε έναν μετρητή. Κάθε φορά που ο χρήστης πατά το κουμπί, η συνάρτηση **setCount** αλλάζει την τιμή του **count**.

3.5.4.2 useEffect

Το **useEffect** είναι ένα hook που επιτρέπει την εκτέλεση παρενεργειών (side effects) σε function components.

```
useEffect(() => {
  // Κώδικας που εκτελείται μετά την κάθε ανανέωση του component
}, [dependencies]);
```

Εικόνα 3.13: Σύνταξη του useEffect

- Ο κώδικας μέσα στην συνάρτηση του **useEffect** εκτελείται μετά από κάθε completed render του component.
- Η λίστα **dependencies** είναι προαιρετική και καθορίζει πότε θα επανεκτελείται ο κώδικας (π.χ., μόνο όταν αλλάξει μια συγκεκριμένη μεταβλητή).

```
function UserStatus({ userId }) {
  const [user, setUser] = useState(null);

  useEffect(() => {
    fetchUser(userId).then(setUser);
  }, [userId]);

  return (
    <div>
      Όνομα χρήστη: {user ? user.name : 'Φορτώνει...'}
    </div>
  );
}
```

Εικόνα 3.14: Παράδειγμα χρήσης useEffect

Στο παράδειγμα αυτό, το **useEffect** χρησιμοποιείται για να φορτώσει δεδομένα χρήστη από ένα API. Το **useEffect** εκτελείται κάθε φορά που αλλάζει το **userId**.

3.5.5 Lifecycle methods

- Στα class components του React, τα lifecycle methods είναι ειδικές μέθοδοι που εκτελούνται σε διάφορα σημεία του κύκλου ζωής ενός component (π.χ., κατά τη δημιουργία, ενημέρωση, και καταστροφή του).
- Περιλαμβάνουν μεθόδους όπως componentDidMount, componentDidUpdate, και componentWillUnmount.

3.5.6 React Context

Το Context στη React είναι ένας τρόπος για την πέραση δεδομένων "κάτω" από την ιεραρχία των components, χωρίς να χρειάζεται να τα περνάμε σε κάθε επίπεδο μέσω props. Αυτό καθιστά το Context ιδανικό για την κοινοποίηση δεδομένων που θεωρούνται global για ένα δέντρο του React, όπως τα themes, οι προτιμήσεις του χρήστη, η πληροφορία πιστοποίησης, κτλ.

1. **Δημιουργία Context:**
    - Χρησιμοποιούμε το **createContext()** hook για να δημιουργήσουμε ένα καινούργιο Context object.

```
const MyContext = createContext(defaultValue);
```

Εικόνα 3.15: Χρήση του createContext

2. **Context Provider:**
    - Ο Context Provider είναι ένας component που παρέχει τα δεδομένα στα υπόλοιπα components.

- Κάθε component που χρειάζεται πρόσβαση στο Context, πρέπει να βρίσκεται μέσα στο δέντρο του Provider.

```
<MyContext.Provider value={/* κάποια τιμή */}>
  {/* υποκείμενα components */}
</MyContext.Provider>
```

Εικόνα 3.16: Expose context με το Context.Provider

3. **Χρήση του Context:**
    - Τα components μπορούν να προσπελάσουν το Context μέσω του **useContext()** hook ή μέσω του **Context.Consumer**.

```
// Χρήση του Context.Consumer
<MyContext.Consumer>
  {value => /* κάντε κάτι με την τιμή του context */}
</MyContext.Consumer>

// ή χρήση της useContext Hook
const value = useContext(MyContext);
```

Εικόνα 3.17: Χρήση του Context με Context.Consumer ή useContext

3.5.6.1 Λόγοι χρήσης του Context

Αναλύοντας περισσότερο την αρχιτεκτονική του context και συγκρίνοντάς την με τις υπόλοιπες επιλογές μας στη διαχείριση ουσιαστικά ενός **global state,** καταλήγουμε στα εξής θετικά χαρακτηριστικά:

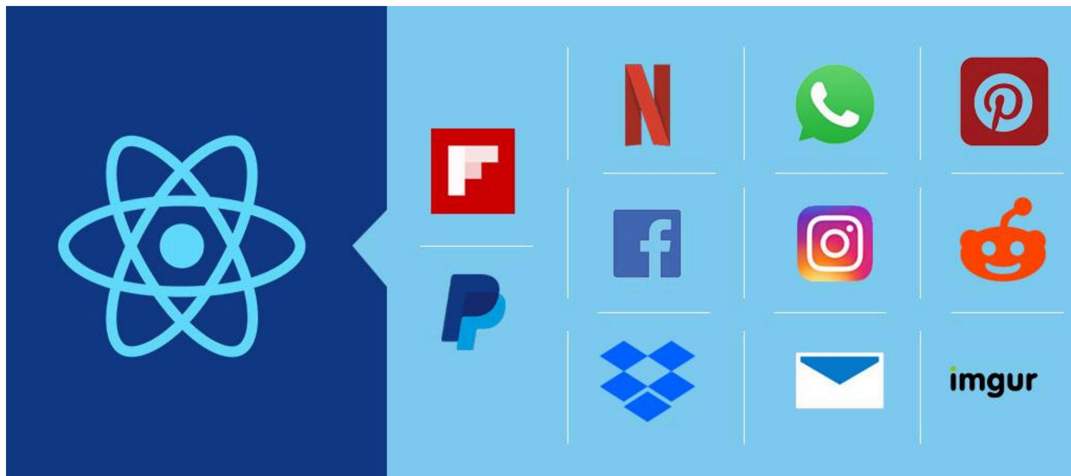
1. **Διαχείριση Κατάστασης σε Βάθος:** Το Context είναι χρήσιμο όταν οι πληροφορίες πρέπει να είναι προσβάσιμες από πολλά components σε διάφορα επίπεδα.
2. **Αποφυγή του "Prop Drilling":** Αποφεύγει την ανάγκη να περνάτε props μέσα από πολλά επίπεδα components για να φτάσετε στα βαθύτερα επίπεδα.
3. **Διαχείριση global ρυθμίσεων**: Χρήσιμο για την εφαρμογή παγκόσμιων ρυθμίσεων όπως θέματα ή γλώσσες. Άλλο παγκόσμιο state μπορεί να χαρακτηριστεί το state του υπάρχον χρήστη, ειδικά σε περιπτώσει σαν τη δική μας που ο χρήστης έχει την επιλογή να αλληλεπιδρά συνεχώς με την εφαρμογή, αλλάζοντας δεδομένα που επηρεάζουν και το δικό του state.
4. **Ευκολία Χρήσης:** Παρέχει μια απλή διεπαφή για την διαχείριση και την μεταφορά δεδομένων.
5. **Καθαρότερος Κώδικας:** Βοηθά στην διατήρηση ενός πιο καθαρού και οργανωμένου κώδικα. Τα Components μας παραμένουν πιο επαναχρησιμοποιήσιμα, καθώς αποφεύγοντας να τα κάνουμε να χειρίζονται εσωτερικά states, μπορούμε να τα χρησιμοποιούμε σε πολλά περισσότερα σημεία του κώδικά μας και με μεγαλύτερη ελευθερία.

```
const ThemeContext = createContext('light');

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <Button theme={theme} />;
}
```

Εικόνα 3.18: Παράδειγμα global ThemeContext

Στο παράδειγμα αυτό, το ThemeContext χρησιμοποιείται για να περάσει την προτίμηση θέματος στον **ThemedButton** component. Αυτό επιτρέπει την αλλαγή της θεματικής εμφάνισης των components χωρίς την ανάγκη για props σε κάθε επίπεδο.

Συνοψίζοντας, η React είναι το πιο διαδεδομένο σύγχρονο web framework τη σημερινή εποχή. Όταν ένας προγραμματιστής διαλέγει τη React ως το framework επιλογής του για την εφαρμογή του, μπορεί να νιώθει ασφάλεια γνωρίζοντας πως υπάρχουν αμέτρητες λύσεις για οποιοδήποτε πρόβλημα αντιμετωπίσει κατα τη διάρκεια ανάπτυξης του project του. Οι αμέτρητες εφαρμογές αξίας εκατομμυρίων που χρησιμοποιούν τη React, διασφαλίζουν πως σε ένα κλάδο του προγραμματισμού που κινείται με τεράστιες ταχύτητες και πολλά πράγματα αλλάζουν συνεχώς, ένα project που είναι γραμμένο σε React θα συνεχίσει να απολαμβάνει υποστήριξη και συνεχείς ενημερώσεις για το άμεσο μέλλον.



Εικόνα 3.19: Δημοφιλείς εφαρμογές σε React

3.6 Material UI

Το Material-UI είναι μία δημοφιλής βιβλιοθήκη σχεδίασης για React, βασισμένη στις αρχές του Material Design της Google. Αυτή η βιβλιοθήκη παρέχει μια σειρά από επαναχρησιμοποιήσιμα UI components και εργαλεία styling, που βοηθούν τους προγραμματιστές να δημιουργήσουν γρήγορα και αποτελεσματικά διεπαφές χρήστη που είναι όχι μόνο λειτουργικές αλλά και αισθητικά ελκυστικές.[25]

Βασικά χαρακτηριστικά MUI:

1. **Επαναχρησιμοποιήσιμα Components:**
   - Περιλαμβάνει μια ευρεία γκάμα από προκαθορισμένα components, όπως κουμπιά, inputs, κάρτες, λίστες και πολλά άλλα.
   - Τα components είναι εύκολα προσαρμόσιμα μέσω props.
2. **Συνολικό Σύστημα Σχεδίασης:**
   - Παρέχει ένα ολοκληρωμένο σύστημα θεμάτων, με τη δυνατότητα εύκολης προσαρμογής χρωμάτων, τυπογραφίας και στοιχείων layout.
3. **Responsive Layout:**
   - Υποστηρίζει τη δημιουργία responsive layouts, καθιστώντας τις διεπαφές φιλικές προς κάθε τύπο συσκευής.

3.6.1 Material UI Components

```
import React from 'react';
import Button from '@material-ui/core/Button';

const MyButton = () => (
  <Button color="primary">
    Πατήστε Εδώ
  </Button>
);
```

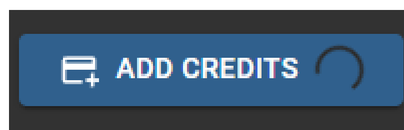
Εικόνα 3.20: Χρήση **Button** του MUI

Στην παραπάνω εικόνα βλέπουμε πόσο απλή είναι χρήση έτοιμων component του Material UI σε μια οποιαδήποτε εφαρμογή React.

```
<Button
  variant="contained"
  color="secondary"
  startIcon={<AddCard />}
  onClick={handleOpenPaymentDialog}
  endIcon={
    (loading || loadingMutation) && (
      <CircularProgress size={24} />
    )
  }
  className="mt-4 w-[12rem]"
>
  Add Credits
</Button>
```

Εικόνα 3.21: Χρήση **props** ενός MUI component

Εδώ βλέπουμε ένα αντίστοιχο παράδειγμα στο οποίο φαίνεται πόσο εύκολο είναι να παραμετροποιούμε και να χειριζόμαστε τα components του Material UI. Μέσα σε λίγες γραμμές κώδικα **jsx,** δώσαμε ουσιαστικά ένα **loading icon,** αλλάξαμε το styling του κουμπιού, χειριστήκαμε το **click event** και είναι το ίδιο εύκολο να κάνουμε αντίστοιχα πράγματα σε όλα τα διαθέσιμα components που υπάρχουν διαθέσιμα.



Εικόνα 3.22: Παράδειγμα Button MUI

Στην τελευταία εικόνα φαίνεται το παράδειγμα κώδικα που είδαμε στην εικόνα 3.15.

3.6.2 Material UI Theme και ThemeProvider

Το Material UI έχει ένα μοναδικό τρόπο ώστε να ομαδοποιούμε το styling της web εφαρμογής μας. Είτε αυτό πρόκειται για ένα component, ένα page ή ακόμη και ολόκληρη την εφαρμογή, το **Theme** είναι το κύριο εργαλείο που χρησιμοποιούμε σε τέτοιες περιπτώσεις, σε συνδυασμό με κάποιο **hooks** και styling functions όπως θα δούμε παρακάτω.[26]

- Χρήση **createStyles:**
  - Η συνάρτηση createStyles χρησιμοποιείται για να δημιουργήσουμε δικά μας **StylingRules.** Αυτός είναι και ο ενδεδειγμένος τρόπος για να κάνουμε override κλάσεις ή styles που υπάρχουν αυτόματα σε ενα Material UI component και δεν υπάρχει τρόπος για αλλαγή ή διαχείρηση τους μέσω των props.

```
import { Theme, createStyles } from "@material-ui/core";

const showDetailsStyle = (theme: Theme) =>
  createStyles({
    button: {
      // Define your button styles here
      backgroundColor: "blue",
      color: "white",
      padding: "10px 20px",
      borderRadius: "5px",
    },
    overview: {
      marginBottom: "6em",
    },
```

Εικόνα 3.23: Παράδειγμα createStyles

- Χρήση **useStyles** και **makeStyles:**
  - Οι δύο αυτές συναρτήσεις χρησιμοποιούνται μαζί. Ουσιαστικά η συνάρτηση makeStyles παίρνει ως παράμετρο τα **StylingRules** που φτιάξαμε στο προηγούμενο βήμα. Με τη συνάρτηση αυτή λοιπόν, και περνώντας την στο hook **useStyles,** το οποίο δημιουργεί εσωτερικά CSS κλάσεις για τα styles που θέσαμε πριν.

```
import style from "../assets/jss/components/artistCardStyle";
const useStyles = makeStyles(style);
//`style` is created by `createStyles`
const ArtistCard: React.FC<ArtistCardProps> = ({
  id,
  fullname,
  image,
  isClaimed,
  isDetails,
}) => {
  const classes = useStyles();
  return (
    <React.Fragment>
      <div className={classes.container}>
        <Avatar
          className={`${classes.avatar} ${
```

Εικόνα 3.24: Παράδειγμα χρήσης useStyles

- Χρήση **Theme** και **ThemeProvider**
    - Το Material-UI προσφέρει ένα πλούσιο και ευέλικτο σύστημα θεμάτων (theming), το οποίο επιτρέπει την προσαρμογή της εμφάνισης των UI components. Μέσω του θεμάτων, οι προγραμματιστές μπορούν να ελέγξουν την αισθητική των εφαρμογών τους, αλλάζοντας παραμέτρους όπως τα χρώματα, η τυπογραφία, οι σκιές, και άλλες ιδιότητες του στυλ.
    - Ο ThemeProvider είναι ένας component που επιτρέπει την εφαρμογή του θέματος σε όλα τα υποκείμενα components στην εφαρμογή. Παρέχει έναν τρόπο για να περνάτε το ορισμένο θέμα κάτω στη δέντρο του React component, ώστε κάθε component να μπορεί να έχει πρόσβαση και να χρησιμοποιεί τις παραμέτρους του θέματος.

```
const context: ThemeContextData = { secondaryColor, setSecondaryColor };
const ThemeProvider = dynamic(
  () => import("@material-ui/core/styles").then((mod) => mod.ThemeProvider),
  { ssr: false }
);
return (
  <ThemeProvider theme={DarkTheme(secondaryColor)}>
    <ThemeContext.Provider value={context}>
      {props.children}
    </ThemeContext.Provider>
  </ThemeProvider>
);
```

Εικόνα 3.25: Παράδειγμα χρήσης του **ThemeContext**

3.7 Tailwind

Το Tailwind CSS είναι ένα utility-first CSS framework που παρέχει μια σειρά από ενσωματωμένες κλάσεις για γρήγορη και ευέλικτη ανάπτυξη διεπαφών χρήστη. Δημιουργήθηκε από τον Adam Wathan και κυκλοφόρησε το 2017. Αντίθετα με τις παραδοσιακές CSS βιβλιοθήκες, το Tailwind επιτρέπει στους προγραμματιστές να δημιουργούν custom designs χωρίς να φεύγουν από το HTML. Έχει κερδίσει μεγάλη δημοτικότητα λόγω της ευελιξίας και της δυνατότητάς του να παράγει συνοπτικό, ευανάγνωστο κώδικα.[27]

Εγκατάσταση και Ρύθμιση του Tailwind CSS

- Μπορείτε να εγκαταστήσετε το Tailwind CSS χρησιμοποιώντας npm ή yarn:

```
npm install tailwindcss
// ή
yarn add tailwindcss
```

Εικόνα 3.26: Εγκατάσταση tailwind CSS

- Στη συνέχεια, ενσωματώστε το Tailwind στα stylesheets σας, συνήθως σε ένα globally διαθέσιμο css αρχείο:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Εικόνα 3.27: Ενσωμάτωση tailwind CSS

- Το επόμενο βήμα είναι να δημιουργήσουμε ένα αρχείο υπεύθυνο για τις ρυθμίσεις της tailwind. Χρησιμοποιούμε το αρχείο **tailwind.config.js** για να προσαρμόσετε το Tailwind. Μπορούμε να επεκτείνουμε ή να τροποποιήσουμε τις προεπιλεγμένες ρυθμίσεις:

```
module.exports = {
  content: [
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
    "./pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./components/**/*.{js,ts,jsx,tsx,mdx}",

    // Or if using `src` directory:
    "./src/**/*.{js,ts,jsx,tsx,mdx}",
  ],
  theme: {
    extend: {
      keyframes: {
        shake: {
          "0%, 100%": { transform: "translateX(0)" },
          "25%": { transform: "translateX(-5px)" },
          "50%": { transform: "translateX(5px)" },
          "75%": { transform: "translateX(-5px)" },
        },
      },
      animation: {
        shake: "shake 0.5s ease-in-out",
      },
    },        You, 2 months ago • Add hook for claiming ac
  },
  plugins: [],
};
```

Εικόνα 3.28: Το αρχείο **tailwind.config.js**

Στην παραπάνω εικόνα βλέπουμε πως έχουμε επεκτείνει τις διαθέσιμες κλάσεις με απλό και γνώριμο τρόπο, όπως ένα απλό Javascript Object. Το μόνο που έχουμε πλέον να κάνουμε είναι να διαλέξουμε που θέλουμε να χρησιμοποιήσουμε τις κλάσεις μας. Στο runtime οι κλάσεις αυτές μεταφράζονται σε απλό CSS.

```
<Dialog
  fullWidth
  maxWidth="sm"
  open={isOpen}
  onClose={onClose}
  className={`!text-black ${isShaking ? "animate-shake" : ""}`}
>
```

Εικόνα 3.29: Παράδειγμα χρήσης Tailwind CSS

Το **!text-black** είναι αντίστοιχο το να γράφαμε το ίδιο με παρακάτω:

```css
.classWithoutTailwind {
  color: black !important;
}
```

Εικόνα 3.30: Παράδειγμα χωρίς Tailwind

Θετικά του Tailwind CSS:

1. **Utility-First Approach:** Το Tailwind προσφέρει μια utility-first προσέγγιση που καθιστά την ανάπτυξη γρήγορη και ευέλικτη.
2. **Ευελιξία στον Σχεδιασμό:** Επιτρέπει τη δημιουργία προσαρμοσμένων designs χωρίς την ανάγκη για επιπλέον CSS.
3. **Συντομία και Απόδοση:** Ο κώδικας γίνεται πιο συμπαγής και ευανάγνωστος, καθώς μειώνεται η ανάγκη για επαναλαμβανόμενες CSS κλάσεις.

Σύγκριση με το Styling του Material-UI:

Ενώ το Material-UI προσφέρει ένα ολοκληρωμένο σύνολο components με ενσωματωμένα στυλ, το Tailwind CSS παρέχει μια πιο ευέλικτη βάση για custom σχεδιασμό. Το Tailwind δεν επιβάλλει μια συγκεκριμένη αισθητική ή δομή στοιχείων, επιτρέποντας στον προγραμματιστή να κατασκευάσει μοναδικές διεπαφές χρήστη από την αρχή. Αντίθετα, το Material-UI είναι ιδανικό για περιπτώσεις όπου οι προεπιλεγμένες διεπαφές και τα components που ακολουθούν τις γραμμές του Material Design είναι προτιμητέα.

Το Tailwind CSS είναι κατάλληλο για προγραμματιστές που θέλουν πλήρη έλεγχο στον σχεδιασμό και προτιμούν να χτίζουν το στυλ τους από την αρχή, ενώ το Material-UI εξυπηρετεί όσους αναζητούν ένα γρήγορο, συνεπές και συστηματικό τρόπο για την ανάπτυξη της διεπαφής τους.

3.8 Next JS

Το Next.js, ανεπτυγμένο από την Vercel, κυκλοφόρησε το 2016 και είναι ένα προηγμένο framework για τη δημιουργία web εφαρμογών με React. Προσφέρει λειτουργίες όπως το routing, server-side rendering (SSR), και static site generation (SSG), βελτιστοποιώντας την απόδοση και την ταχύτητα των εφαρμογών. Είναι αρκετά δυναμικό και όπως και στη δική μας περίπτωση, είναι πολύ καλό στο να συνδυάζει την ταχύτητα του Server Side Rendering αλλά και τη δημιουργία dynamic client side σελίδων.[28]

3.8.1 Δημιουργία σελίδων

Στην Next.js, κάθε σελίδα είναι ένα React component που προέρχεται από τον φάκελο **'pages'**. Η διαδρομή κάθε σελίδας προσδιορίζεται από το όνομα του αρχείου. Για παράδειγμα, ένα αρχείο about.js στον φάκελο **'pages'** δημιουργεί τη διαδρομή **'/about'**.
Οι σελίδες στην Next.js υπόκεινται σε pre-rendering, δηλαδή ο κώδικας HTML και τα δεδομένα φορτώνονται στον server πριν φτάσουν στον πελάτη. Αυτό βελτιώνει τόσο τις επιδόσεις όσο και το SEO της εφαρμογής.
Για να δημιουργήσουμε μια νέα σελίδα στην Next.js, απλά προσθέτουμε ένα νέο αρχείο JavaScript στον φάκελο **pages**. Κάθε αρχείο αντιστοιχεί σε μια διαδρομή στην εφαρμογή.

```
// pages/about.js
function About() {
  return <div>Σχετικά με μας</div>;
}

export default About;
```

Εικόνα 3.31: Δημιουργία σελίδας στο Next JS

Στο παράδειγμα αυτό, δημιουργήσαμε μια σελίδα "Σχετικά με μας" που είναι προσβάσιμη στη διαδρομή **/about**.

3.8.2 Server-Side vs Static Site Generation

Η Next.js προσφέρει δύο τρόπους pre-rendering:

1. **Server-Side Rendering (SSR):** Παράγει HTML κατά την εκτέλεση της εφαρμογής. Χρησιμοποιεί τη μέθοδο **getServerSideProps** για την ανάκτηση δεδομένων, η οποία εκτελείται στον server. Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια σελίδα προφίλ χρήστη η οποία φορτώνει δεδομένα από έναν εξωτερικό API. Θα χρησιμοποιήσουμε την getServerSideProps για να φορτώσουμε τα δεδομένα κατά την πρόσβαση του χρήστη στη σελίδα.

```
// pages/profile.js
export async function getServerSideProps(context) {
  const res = await fetch(`https://example.com/api/user/${context.params.id}`);
  const data = await res.json();

  return {
    props: { user: data }, // θα περάσει τα δεδομένα στο Profile component ως props
  };
}

function Profile({ user }) {
  return <div>Όνομα χρήστη: {user.name}</div>;
}

export default Profile;
```

Εικόνα 3.32: Παράδειγμα getServerSideProps

Στο παραπάνω παράδειγμα, η getServerSideProps εκτελείται στον server κάθε φορά που ένας χρήστης ανοίγει τη σελίδα profile, φορτώνοντας τα δεδομένα του χρήστη από τον εξωτερικό API.

2. **Static Site Generation (SSG):** Δημιουργεί στατικό HTML κατά τη φάση χτισίματος της εφαρμογής. Με τη μέθοδο **getStaticProps**, η σελίδα παράγεται μία φορά και χρησιμοποιείται επανειλημμένα. Για τη δημιουργία μιας στατικής σελίδας, για παράδειγμα, μια σελίδα blog, με δεδομένα που δεν θα αλλάξουν, θα χρησιμοποιήσουμε την getStaticProps.

```
// pages/blog.js
export async function getStaticProps() {
  const res = await fetch('https://example.com/api/posts');
  const posts = await res.json();

  return {
    props: {
      posts,
    },
    revalidate: 10, // Ανανεώνει την σελίδα κάθε 10 δευτερόλεπτα
  };
}

function Blog({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
}

export default Blog;
```

Εικόνα 3.33: Παράδειγμα getStaticProps

Εδώ, η getStaticProps φορτώνει τα δεδομένα της σελίδας blog κατά τη διάρκεια του build time. Η σελίδα ανανεώνεται αυτόματα κάθε 10 δευτερόλεπτα.

3.8.3 API Routes για Backend Λειτουργίες

Η Next.js επιτρέπει τη δημιουργία API endpoints μέσω των API routes. Αυτά βρίσκονται στον φάκελο pages/api και εκτελούνται στον server, διατηρώντας ευαίσθητα δεδομένα μακριά από τον πελάτη. Τα API routes στην Next.js επιτρέπουν τη δημιουργία server-side λειτουργιών και endpoints εντός της ίδιας της εφαρμογής Next.js. Αυτό επιτρέπει την αποκρυπτογράφηση της λογικής του backend και την ανάπτυξη API endpoints χωρίς την ανάγκη για ξεχωριστό server.
Για παράδειγμα, μια απλή λειτουργία API route μπορεί να είναι η επεξεργασία μιας φόρμας επικοινωνίας ή η παροχή δεδομένων από μια βάση δεδομένων. Αυτές οι διαδρομές είναι προσβάσιμες μέσω URL του τύπου /api/endpoint, όπου endpoint είναι το όνομα του αρχείου route.
Η χρήση των API routes συμβάλλει στην απλοποίηση της αρχιτεκτονικής της εφαρμογής, καθώς και στην κεντρική διαχείριση τόσο του frontend όσο και του backend κώδικα μέσα στο ίδιο πρόγραμμα.

```
// pages/api/hello.js
export default function handler(req, res) {
  res.status(200).json({ text: 'Γειά!' });
}
```

Εικόνα 3.34: API endpoint στη Next JS

Αυτό το αρχείο προσθέτει ένα API endpoint στη διαδρομή /api/hello, το οποίο επιστρέφει ένα JSON αντικείμενο.

Συμπέρασμα

Η Next.js είναι ένα ισχυρό εργαλείο για τη δημιουργία δυναμικών web εφαρμογών με React, προσφέροντας προηγμένες λειτουργίες για απόδοση, SEO και ευελιξία στη διαχείριση δεδομένων. Είναι επίσης ευρέως διαδεδομένη ως το πιο γνωστό και χρησιμοποιημένο framework βασισμένο στις τεχνολογίες της React και του Jamstack, απο web developers.

Εικόνα 3.35: Ποσοστό χρήσης Next JS

3.9 Git

Το Git είναι ένα διανεμημένο σύστημα ελέγχου έκδοσης (**distributed version control system - DVCS**) που επιτρέπει στους προγραμματιστές να διαχειρίζονται και να παρακολουθούν τις αλλαγές στον κώδικα τους με αποτελεσματικότητα και ασφάλεια. Δημιουργήθηκε από τον **Linus Torvalds**, τον δημιουργό του Linux, το 2005[29]. Το Git αναπτύχθηκε ως απάντηση στην ανάγκη για ένα ισχυρό, γρήγορο και ευέλικτο εργαλείο ελέγχου έκδοσης που θα υποστήριζε την ανάπτυξη του πυρήνα του Linux. Μερικές από τις βασικές λειτουργίες του Git περιλαμβάνουν:

- **Δημιουργία Αποθετηρίων (Repositories)**: Ένα αποθετήριο είναι το μέρος όπου αποθηκεύεται ολόκληρη η ιστορία του project, συμπεριλαμβανομένων όλων των εκδόσεων και των αλλαγών στον κώδικα.
- **Commit:** Η ενέργεια του commit αποθηκεύει τις αλλαγές σε ένα αποθετήριο. Κάθε commit περιέχει ένα μήνυμα που περιγράφει τις αλλαγές που έγιναν, επιτρέποντας στους προγραμματιστές να παρακολουθούν και να τεκμηριώνουν την εξέλιξη του project.

- **Branching και Merging:** Το branching επιτρέπει στους προγραμματιστές να δημιουργούν ανεξάρτητους κλάδους (branches) για την ανάπτυξη νέων χαρακτηριστικών ή τη διόρθωση σφαλμάτων, χωρίς να επηρεάζουν τον κύριο κώδικα. Όταν οι αλλαγές σε ένα branch είναι έτοιμες, μπορούν να συγχωνευτούν (merge) με τον κύριο κλάδο (main branch).
- **Pull και Push:** Το pull φέρνει τις αλλαγές από ένα απομακρυσμένο αποθετήριο στο τοπικό αποθετήριο του χρήστη, ενώ το push στέλνει τις αλλαγές του χρήστη από το τοπικό στο απομακρυσμένο αποθετήριο.

Το Git προσφέρει πολλά πλεονεκτήματα που συμβάλλουν στη βελτίωση της ανάπτυξης και της παρακολούθησης της εξέλιξης ενός project:

- **Ιστορικό Αλλαγών:** Το Git καταγράφει κάθε αλλαγή που γίνεται στο project, επιτρέποντας στους προγραμματιστές να επιστρέψουν σε προηγούμενες εκδόσεις του κώδικα αν χρειαστεί. Αυτό διευκολύνει την εύρεση και διόρθωση σφαλμάτων, καθώς και την κατανόηση της εξέλιξης του κώδικα.
- **Συνεργασία:** Το Git επιτρέπει σε πολλούς προγραμματιστές να εργάζονται ταυτόχρονα στο ίδιο project χωρίς να συγκρούονται οι αλλαγές τους. Μέσω των branches, κάθε προγραμματιστής μπορεί να δουλεύει ανεξάρτητα και στη συνέχεια να συγχωνεύει τις αλλαγές του με τον κύριο κλάδο, διευκολύνοντας τη συνεργασία σε μεγάλες ομάδες.
- **Διαχείριση Κλάδων:** Με τα branches, οι προγραμματιστές μπορούν να πειραματίζονται με νέες ιδέες ή να αναπτύσσουν νέα χαρακτηριστικά χωρίς να διακινδυνεύουν την σταθερότητα του κύριου κώδικα. Αυτό παρέχει μια ασφαλή μέθοδο ανάπτυξης και δοκιμής.
- **Αναστρέψιμες Αλλαγές:** Αν κάτι πάει στραβά, οι αλλαγές μπορούν εύκολα να αναστραφούν χάρη στην πλήρη ιστορία των commits. Αυτό μειώνει τον κίνδυνο μόνιμης απώλειας δεδομένων και βοηθάει στη γρήγορη αποκατάσταση του κώδικα.
- **Ενσωμάτωση με άλλα Εργαλεία:** Το Git μπορεί να ενσωματωθεί με διάφορα άλλα εργαλεία ανάπτυξης και διαχείρισης έργων, όπως το GitHub, το GitLab και το Bitbucket, προσφέροντας πρόσθετες δυνατότητες για παρακολούθηση προβλημάτων, συνεχή ενσωμάτωση (continuous integration) και ανάπτυξη (deployment).

Η χρήση του Git καθιστά τη διαδικασία ανάπτυξης λογισμικού πιο αποδοτική και ευέλικτη, ενώ παράλληλα παρέχει τα εργαλεία που χρειάζονται οι προγραμματιστές για να παρακολουθούν και να διαχειρίζονται τις αλλαγές στον κώδικα με ασφάλεια.

3.10 Dependencies lifecycle

Στα σύγχρονα JavaScript projects, η χρήση βιβλιοθηκών και frameworks είναι απαραίτητη για την ταχεία ανάπτυξη και την υλοποίηση προηγμένων λειτουργιών. Ωστόσο, αυτή η εξάρτηση από εξωτερικά dependencies φέρνει μαζί της και το ζήτημα του dependency update, δηλαδή την ανάγκη να διατηρούνται οι εξαρτήσεις ενημερωμένες.
Η εργασία με legacy code, δηλαδή κώδικα που γράφτηκε σε παλαιότερες εκδόσεις βιβλιοθηκών ή frameworks, μπορεί να παρουσιάσει διάφορα προβλήματα:

- **Ασυμβατότητες και Bugs:** Καθώς οι βιβλιοθήκες αναβαθμίζονται, μπορεί να εισάγουν αλλαγές που δεν είναι συμβατές με τον υπάρχοντα κώδικα. Αυτό μπορεί να οδηγήσει σε bugs ή ακόμη και σε πλήρη αστοχία λειτουργιών.

- **Ασφάλεια:** Παλαιότερες εκδόσεις βιβλιοθηκών συχνά περιέχουν ευπάθειες ασφαλείας που έχουν διορθωθεί σε νεότερες εκδόσεις. Η χρήση μη ενημερωμένων βιβλιοθηκών μπορεί να αφήσει το project εκτεθειμένο σε επιθέσεις.
- **Υποστήριξη:** Οι δημιουργοί βιβλιοθηκών συχνά σταματούν να υποστηρίζουν παλαιότερες εκδόσεις, παρέχοντας ενημερώσεις και διορθώσεις μόνο για τις νεότερες εκδόσεις. Αυτό σημαίνει ότι ο παλαιότερος κώδικας μπορεί να μην λαμβάνει τις απαραίτητες βελτιώσεις ή διορθώσεις σφαλμάτων.
- **Τεχνικό Χρέος:** Η συνεχής χρήση παλαιότερων εκδόσεων μπορεί να οδηγήσει σε τεχνικό χρέος(**tech debt**), καθιστώντας πιο δύσκολη και χρονοβόρα τη συντήρηση και την ανάπτυξη νέων χαρακτηριστικών στο μέλλον.

Για να αποφευχθούν τα παραπάνω προβλήματα, είναι κρίσιμο να διατηρούμε και να αναβαθμίζουμε συχνά το source code και τις εξαρτήσεις του. Αυτό προσφέρει τα εξής οφέλη:

- Βελτιωμένη Ασφάλεια: Οι συχνές αναβαθμίσεις διασφαλίζουν ότι το project χρησιμοποιεί τις πιο πρόσφατες και ασφαλείς εκδόσεις των βιβλιοθηκών, μειώνοντας τον κίνδυνο ευπαθειών.
- Σταθερότητα και Απόδοση: Νεότερες εκδόσεις συχνά περιλαμβάνουν βελτιώσεις στην απόδοση και τη σταθερότητα, επιτρέποντας στο project να λειτουργεί πιο αποτελεσματικά.
- Νέες Λειτουργίες: Οι αναβαθμίσεις συχνά φέρνουν νέες δυνατότητες και βελτιώσεις, που μπορούν να ενισχύσουν την ανάπτυξη και τη λειτουργικότητα του project.
- Μείωση Τεχνικού Χρέους: Η τακτική συντήρηση και αναβάθμιση του κώδικα μειώνει το τεχνικό χρέος, διευκολύνοντας τη μελλοντική ανάπτυξη και συντήρηση του project.
- Καλύτερη Υποστήριξη: Η χρήση ενημερωμένων βιβλιοθηκών σημαίνει ότι το project μπορεί να εκμεταλλευτεί τη συνεχή υποστήριξη και τις ενημερώσεις που προσφέρουν οι δημιουργοί των βιβλιοθηκών.

Συνολικά, η διαχείριση των dependencies και η τακτική αναβάθμιση του κώδικα είναι ζωτικής σημασίας για τη διατήρηση της ασφάλειας, της σταθερότητας και της ευελιξίας ενός JavaScript project. Η επένδυση χρόνου και πόρων στη συντήρηση και την αναβάθμιση του κώδικα όχι μόνο προστατεύει το project από πιθανούς κινδύνους, αλλά επίσης διασφαλίζει ότι παραμένει ανταγωνιστικό και αποδοτικό μακροπρόθεσμα.

3.11 Επίλογος

Στο τρίτο κεφάλαιο, είχαμε την ευκαιρία να εξερευνήσουμε βαθύτερα μια σειρά από βασικές τεχνολογίες που συνθέτουν τον πυρήνα της εφαρμογής μας. Κάθε τεχνολογία που παρουσιάστηκε συμβάλλει με μοναδικό τρόπο στη λειτουργία και την απόδοση της εφαρμογής, αναδεικνύοντας την πολυπλοκότητα και την δυναμική του σύγχρονου web development.
Από την διαχείριση πακέτων με εργαλεία όπως npm, μέχρι τον προγραμματισμό web scripting για τη δημιουργία δυναμικών web σελίδων, κάθε τεχνολογία έχει τη δική της σημασία. Το JWT (Json Web Tokens) μας επέτρεψε να χειριζόμαστε την ασφάλεια και την ταυτοποίηση των χρηστών, ενώ οι Axios Interceptors βελτίωσαν τον τρόπο διαχείρισης των HTTP αιτημάτων και αποκρίσεων.
Στην καρδιά της διεπαφής χρήστη βρίσκεται το React.js, που μας παρέχει τη δυνατότητα να δημιουργούμε διαδραστικά και αποδοτικά περιβάλλοντα. Επιπλέον, με τη χρήση του Material UI και του Tailwind, καταφέραμε να δώσουμε στυλ και αισθητική στην εφαρμογή μας, ενώ ταυτόχρονα διατηρούμε την καθαρότητα και τη συνοχή του κώδικα.

Τέλος, το Next.js έδωσε μια νέα διάσταση στην ανάπτυξη της εφαρμογής, προσφέροντας ταχύτητα, απόδοση και προσαρμοστικότητα μέσω των λειτουργιών του SSR και SSG, καθώς και των δυνατοτήτων του API routing.
Αφήσαμε εκτός από το κεφάλαιο αυτό τεχνολογίες που αναφέρθηκαν στο κεφάλαιο 2 και τεχνολογίες και τεχνικές που θα αναλύσουμε στα επόμενα 2 κεφάλαια.

Κεφάλαιο 4ο: Περιγραφή εργασίας

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περιγράψουμε σύντομα την εργασία που παραλάβαμε από τον προηγούμενο φοιτητή και τις αλλαγές που κάναμε πάνω στον υπάρχον κώδικα και design. Η λογική της αναβάθμισης ήταν πάντα στα πλαίσια του να μην ξαναγράψουμε μια εφαρμογή από την αρχή αλλά να δουλέψουμε με όσο το δυνατόν περισσότερο ανακυκλώσιμο κώδικα μπορούμε. Η παρουσίαση θα γίνει με τον εξής τρόπο:

- Υπάρχον κομμάτι του project και ιδέα για αναβάθμιση/προσθήκη λειτουργίας. Ανάλυση των λόγων και των θετικών καθώς και των αρνητικών.
- Υλοποίηση της ιδέας. Επικοινωνία ανάμεσα στα μέλη της ομάδας για τον τρόπο υλοποίησης για την ιδέα αυτή.

4.2 Refactoring για χρήση Typescript

Το πρώτο και μεγαλύτερο εγχείρημα που ανέλαβα ήταν η μετατροπή του κώδικα από απλή Javascript σε Typescript. Ο υπάρχον κώδικας βρισκόταν εξ ολοκλήρου σε σκέτη Javascript. Τα αρχεία της React ήταν όλα απλά jsx αρχεία. Η μετατροπή αυτή διήρκησε αρκετό σχεδιασμό και πολλές εβδομάδες δουλειάς.

4.2.1 Λόγοι προσθήκης Typescript

Ένα σύγχρονο project, ειδικά ένα δυναμικό web app, που χρησιμοποιεί ένα από τα πιο σύγχρονα web frameworks, είναι πάντα πολύ πιθανό να χρειαστεί να αλλάξουν ή να δουλέψουν πάνω σε αυτά πολλά διαφορετικά άτομα. Το τελευταίο ισχύει στη δική μας περίπτωση καθώς η εργασία περνάει από φοιτητή και αυτός με τη σειρά του καλείτε να μελετήσει το υπάρχον project και να υλοποιήσει ή να αλλάξει υπηρεσίες και λειτουργίες.
Η Typescript μας δίνει τη δυνατότητα να κρατάμε τον κώδικα μας πολύ πιο οργανωμένο με τη χρήση των **types** και των **interfaces**. Όσο μια εφαρμογή επεκτείνεται, τα entities μιας εφαρμογής γίνονται περισσότερα ή και τα πιθανά πεδία μέσα σε ένα υπάρχον entity πολλαπλασιάζονται. Για έναν προγραμματιστή αυτό σημαίνει πως πρέπει να θυμάται τις αλλαγές που έχουν γίνει και τα πεδία που ήδη υπήρχαν ή άλλαξαν.

```
export const fetchVenueProductionsByVenueId = async (id: number) => {
  try {
    return (await mainFetcher(`Venues/${id}/productions`))
      .results        You, 27 minutes ago • Uncommitted changes
  } catch (error) {
    console.log("Error in fetch venue productions by id", error);
    return null;
  }
};
```

Εικόνα 4.1: API call χωρίς TS

```
// Returns type any.        (alias) fetchVenueProductionsByVenueId(id: number):
//Doens't                   Promise<any>
// type check               import fetchVenueProductionsByVenueId
const productions = await fetchVenueProductionsByVenueId(id);
productions.wrongField;
```

Εικόνα 4.2: Παράδειγμα χωρίς χρήση TS

Στην παραπάνω εικόνα βλέπουμε ότι ένα call στο API μας επιστρέφει type **any**. Αυτό σημαίνει ότι το IDE που χρησιμοποιούμε, δεν θα μας βοηθήσει να δούμε λάθη που κάνουμε την ώρα που γράφουμε τον κώδικά μας. Τέτοια λάθη είναι αρκετά συνηθισμένα. Συντακτικά λάθη όπως για παράδειγμα, **.resuls** αντί για **.results**. Τέτοια λάθη οδηγούν σε Runtime errors, που κάνουν την παραγωγή κώδικα αρκετά πιο αργή και ενοχλητική για έναν developer. Στη δική μας περίπτωση καταφέραμε να διορθώσουμε αρκετά τέτοια λάθη που ήδη υπήρχαν ή που προέκυψαν στην πορεία. Είναι συνηθώς πολύ μικρά σφάλματα στον κώδικα τα οποία λύνονται με 1 ή 2 κινήσεις. Παρόλα αυτά το debugging τους συνήθως είναι ενοχλητικό και παίρνει πολύ ώρα χωρίς τη βοήθεια της Typescript.

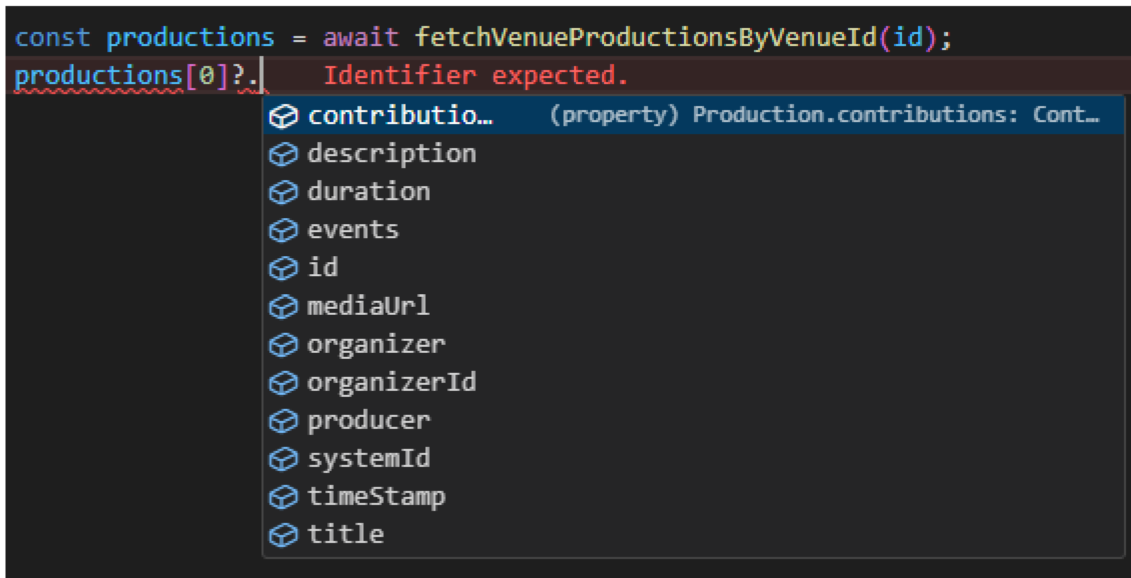
```
export const fetchVenueProductionsByVenueId = async (
  id: number
): Promise<Production[] | null> => {
  try {
    return (await mainFetcher(`Venues/${id}/productions`))
      .results as Production[];
  } catch (error) {
    console.log("Error in fetch venue productions by id", error);
    return null;
  }
};
```

Εικόνα 4.3: API call με TS

```
const productions = await fetchVenueProductionsByVenueId(id);

    Property 'wrongField' does not exist on type
    'Production[]'. ts(2339)
    any
    View Problem (Alt+F8)    No quick fixes available
productions.wrongField;    'productions' is possibly 'null'.
//Type checking. Prevents runtime errors
```

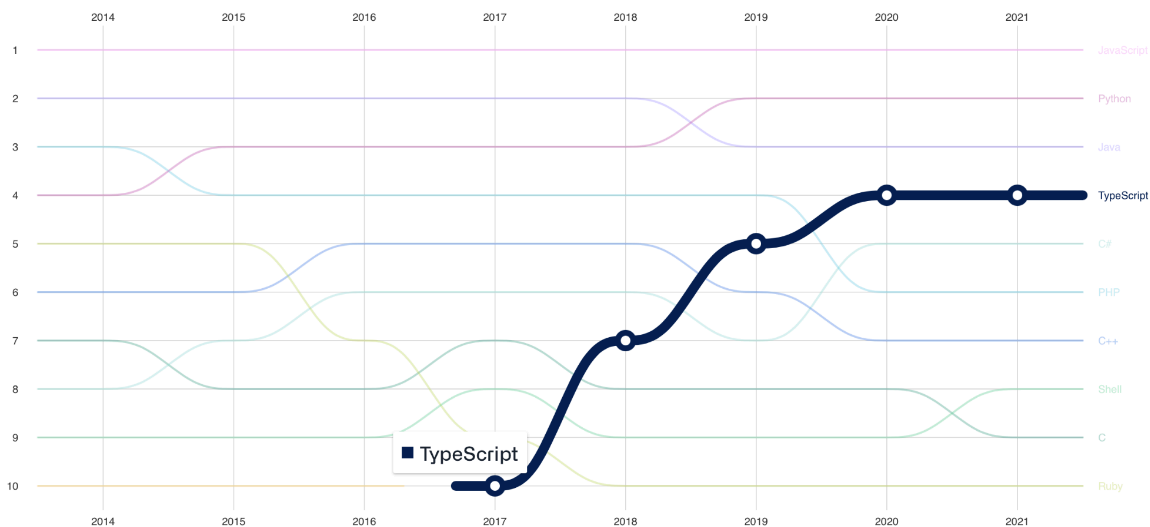
Εικόνα 4.4: Παράδειγμα με χρήση TS

Στην αντίθετη περίπτωση όπως βλέπουμε, η Typescript μας προειδοποιεί ότι, με βάση το interface του **Production** που χρησιμοποιούμε, το **wrongField**, είναι ένα πεδίο που δεν υπάρχει. Επίσης μας δείχνει ότι το απευθείας access των πεδίων του **productions** μπορεί να προκαλέσει προβλήματα καθώς το αποτελέσματα του API call ίσως να είναι **null.** Στην πραγματικότητα, το autocomplete που μας προσφέρει η Typescript, μαζί με το static type checking, είναι εργαλεία τα οποία μας προτρέπουν να μην κάνουμε λάθη που θα καθυστερήσουν το development του project μας. Ένας προγραμματιστής που δεν έχει ξαναδεί αυτό το project, μπορει απευθείας να αναλάβει μικρά task προς υλοποίηση, χωρίς να χρειαστεί να ξέρει από πριν το layout του API μας και τα entities της εφαρμογής μας.



Εικόνα 4.5: Autocomplete με TS

Στην περίπτωση λοιπόν, της typescript, σε οποιοδήποτε από τα δημοφιλές IDE, όταν κάνουμε access μια μεταβλητή με ήδη asserted type, αυτό είναι το autocomplete prompt που βλέπουμε. Οι λόγοι λοιπόν που αναφέραμε είναι αυτό που μας έκανε να πάρουμε την απόφαση να κάνουμε στροφή του project μας προς την Typescript.



Εικόνα 4.6: Χρήση TS τα τελευταία χρόνια

4.2.2 Προσθήκη Typescript

Όπως αναφέραμε και προηγουμένως η προσθήκη της Typescript σε ένα project είναι μια αρκετά απλή διαδικασία.

```
\theatrica_frontend_web> npm install typescript
```

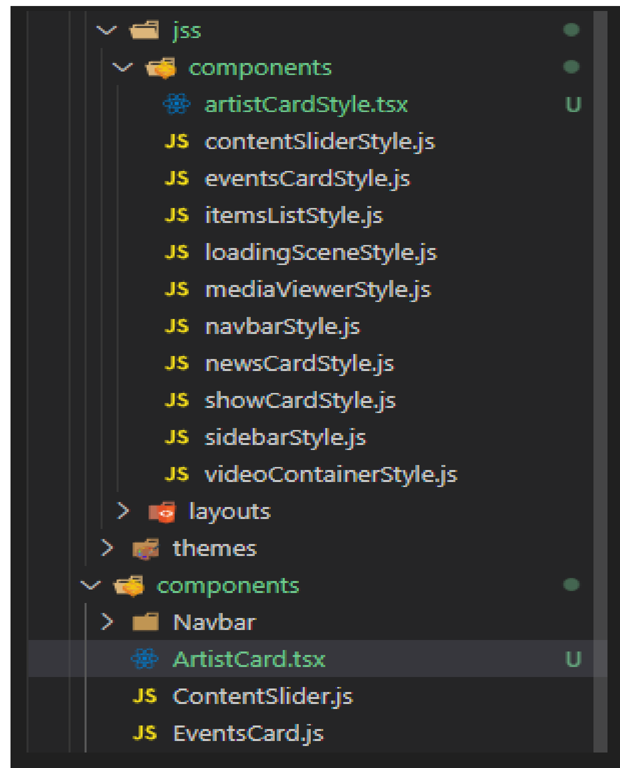
Εικόνα 4.7: Εγκατάσταση Typescript με npm

Αφού εγκαταστήσουμε λοιπόν την TS και κάνουμε τα κατάλληλα βήματα, ρυθμίζοντας μέσω του tsconfig.json αρχείου τον compiler μας ώστε να λειτουργεί σωστά με την εφαρμογή μας, μπορούμε να ξεκινήσουμε να μετατρέπουμε το project μας από απλή JS σε type safe και δοδημένη Typescript. Ξεκινάμε με το να σκεφτόμαστε με τους τύπους που θα χρησιμοποιήσουμε στην εφαρμογή μας. Με πιο απλά λόγια, κάνουμε μια στροφή και αρχίζουμε να σκεφτόμαστε με πιο αντικειμενοστρεφή τρόπο.

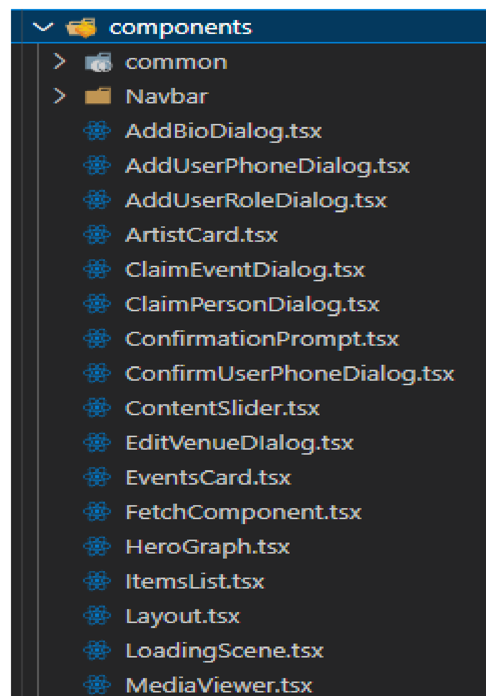
```
12  export interface User {
13    id: number;
14    email: string;
15    emailVerified: boolean;
16    _2FA_enabled: boolean;
17    role: string;
18    performerRoles: string[];
19    transactions?: Transaction[];
20    balance: number;
21    facebook?: string;
22    instagram?: string;
23    youtube?: string;
24    profilePhoto?: UserPhoto;
25    bioPdfLocation?: string;
26    userImages: UserPhoto[];
27    phoneNumber?: string;
28    phoneNumberVerified?: boolean;
29    claimedPerson: Person;
30    claimedVenues: Venue[];
31    claimedEvents: Event[];
32  }
```

Εικόνα 4.8: Παράδειγμα ενός TS interface

Ουσιαστικά το πιο σημαντικό κομμάτι της μετατροπής σε TS είναι η σωστή χαρτογράφηση στο μυαλό μας της εφαρμογής μας και πως θα μοιάζει στην τελική της μορφή. Χρησιμοποιώντας αυτά τα αρχικά interfaces, ξεκινάμε το δεύτερο μέρος που είναι η μετατροπή των αρχείων Javascript σε Typescript και των αντίστοιχων **JSX** σε **TSX**.



Εικόνα 4.9: Αρχικός κώδικας σε JSX.



Εικόνα 4.10: Μετατροπή σε TSX.

Με τη σταδιακή αυτή μετατροπή, ο compiler της Typescript απαιτεί από εμάς να αποδώσουμε ουσιαστικά types σε οτιδήποτε χρησιμοποιούμε. Ταυτόχρονα αν χρησιμοποιούμε κάποιο type σε ένα μέρος που δεν πρέπει, ο compiler μας προειδοποιεί για το λάθος αυτό.

```
function ArtistCard({ id, fullName, image }) {
}
```

Εικόνα 4.11: React component με JSX

```
export interface ArtistCardProps extends Person {
}

const ArtistCard: React.FC<ArtistCardProps> = ({
};
```

Εικόνα 4.12: React component με TSX

Στην εικόνα 4.12 βλέπουμε πως πλέον, αν και με παραπάνω κώδικα, τα components μας, αλλά και οτιδήποτε άλλο χρησιμοποιούμε μέσα στην εφαρμογή, είναι πολύ πιο ανθεκτικά και σε μελλοντικές προσθήκες. Αν μελλοντικά αποφασίσουμε να αλλάξουμε κάποια πράγματα στο interface του **Person,** ο compiler θα μας δείξει που ακριβώς χρησιμοποιούσαμε αυτό το interface πριν τις αλλαγές. Έτσι θα γίνουν οι απαραίτητες διορθώσεις, χωρίς να αφιερώνουμε περιττό χρόνο, κάνοντας debugging σε απρόσμενα runtime errors.

4.3 Προσθήκη Tailwind

Για την ανάπτυξη των καινούργιων components, πήραμε την απόφαση να χρησιμοποιήσουμε tailwind. Ένας απο τους λόγους είναι η ξεπερασμένη πλέον τεχνολογία από μερικές απόψεις, του **createStyles** και **useStyles** του Material UI, όπως αναφέραμε και στο Κεφάλαιο 3. Η tailwind είναι μια βιβλιοθήκη που κάνει την ανάπτυξη ευέλικτων και όμορφων component πολύ πιο γρήγορη. Επίσης η ανάπτυξη με tailwind είναι πολύ πιο εύκολη στην ανάγνωση και κατανόηση του κώδικα, από ένα άτομο που δεν έχει ξαναδεί το συγκεκριμένο κώδικα. Καθώς τα επιμέρους tasks αυτού του project έχουν κατανεμηθεί σε 2 άτομα και καθώς η λογική αυτής της εφαρμογής είναι το να αλλάζει χέρια από φοιτητή σε φοιτητή και να γίνεται όλο και καλύτερη και πιο σύγχρονη, η tailwind σίγουρα θα βοηθήσει στο να κομμάτι αυτό.

4.4 Προσθήκη λογικής JWT

Το δεύτερο μεγαλύτερο task που είχαμε στη εφαρμογή που παραλάβαμε, ήταν να δημιουργήσουμε μια ασφαλή και προσωποποιημένη επικοινωνία ανάμεσα στο app και τη βάση δεδομένων μας μέσω του API. Όπως αναφέραμε και στο Κεφάλαιο 2, η χρήση των **JSON Web Tokens,** είναι από τις πιο διαδεδομένες τεχνικές διασφάλισης επικοινωνίας μέσω του **HTTPS**, συγκεκριμένα για ένα stateless API όπως το δικό μας.

Για τις νέες ανάγκες της εφαρμογής μας, αξιοποιήσαμε την ήδη υπάρχον βιβλιοθήκη **Axios.** Μέσω του Axios μπορούμε να ελέγχουμε τα **requests** καθώς και τα **responses** ολόκληρου του app, με έναν τρόπο πολύ απλό και με μόλις λίγες γραμμές κώδικα. Το Axios προσφέρει by default, για κάθε instance που δημιουργείται, **interceptos,** τα οποία με απλά λόγια κάνουν κάθε request και response εύκολα διαχειρίσιμα με τη χρήση απλής Javascript. Από το αντλήσουμε πληροφορία για τον τύπο ενός request(put, get, post, delete), είτε να ελέγξουμε το **status code** ενός response ώστε να το διαχειριστούμε αναλόγως, οι interceptors μας προσφέρουν όλες αυτές τις λειτουργίες.

4.4.1 Χρήση interceptors

Η λογική των interceptors είναι ακριβώς αυτή που περιμένουμε να είναι με βάση την ονομασία τους. Κάνουν **intercept** ένα response προτού φτάσει στο τελικό του σημείο, δηλαδή στο σημείο που καλέσαμε για παράδειγμα ένα get request στο API μας. Αντίστοιχα κάνουν intercept ένα request και το τροποποιούν αναλόγως, προτού αυτό φύγει από τη μεριά του client και σταλεί στο API.
Στη δική μας περίπτωση χρησιμοποιήσαμε ένα αρχείο με όνομα **AxiosInstances.tsx.** Δημιουργήσαμε το βασικό instance το οποίο και κάναμε **export** για χρήση στο υπόλοιπο app.

```
export const baseURL =
  "https://theatricalapi.jollybay-0ad0b06b.germanywestcentral.azurecontainerapps.io/api";

export const mainAxios = axios.create({
  baseURL,
});
```

Εικόνα 4.13: Δημιουργία mainAxios instance

Στη συνέχεια με έναν συνδυασμό request και response interceptors σε αυτό το intance, κάναμε την εφαρμογή μας να χειρίζεται σωστά τα JW Tokens.

```
mainAxios.interceptors.request.use((config) => {
  if (typeof window !== "undefined") {
    const token = localStorage.getItem("authToken");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
  }
  return config;
});
```

Εικόνα 4.14: Εισαγωγή JWT

Όπως βλέπουμε στο παραπάνω παράδειγμα, η συνάρτηση **use** του **mainAxios.interceptors.request** παίρνει μια παράμετρο **config,** που είναι τύπου **AxiosRequestConfig.** Μέσω αυτής της παραμέτρου έχουμε πρόσβαση σε κάθε **request** που περνάει μέσα από το **mainAxios** instance. Στη συγκεκριμένη περίπτωση προσθέτουμε στο **Authorization** των **headers** το JWT που έχουμε λάβει από το API μας με το **login.** Έχουμε αποθηκεύσει προηγουμένως το **token** στο **localStorage** του browser με **key "authToken"**. Θα δείξουμε παρακάτω πως χειριζόμαστε περιπτώσεις που το JWT δεν υπάρχει ή έχει λήξει.
Ο έλεγχος **if(typeof window !== "undefined")** γίνεται λόγω του Next JS. Ο κώδικάς μας θα ξανατρέξει στη μεριά του server και καθώς ο server δεν έχει πρόσβαση σε κάποιο αντικείμενο **window**(μόνο οι browsers στο client side έχουν πρόσβαση σε αυτό), θα προκύψει runtime error και ο

κώδικας μας δε θα λειτουργήσει. Περιπτώσεις σαν αυτές είναι παραδείγματα των αρνητικών πτυχών του να έχεις ένα hybrid SSR σύστημα. Ο χειρισμός όμως στην περίπτωση αυτή είναι αρκετά εύκολος και απλός.
Αντίστοιχη διαδικασία ακολουθούμε για να εξάγουμε ένα εισερχόμενο JWT, αλλά αυτή τη φορά χρησιμοποιώντας ενα response interceptor.

```
mainAxios.interceptors.response.use(
  (response) => {
    // If a new token is provided in the response, update it in localStorage
    if (typeof window !== "undefined" && response.data.data?.access_token) {
      const token = response.data.data.access_token;
      localStorage.setItem("authToken", token);
    }
    return response;
  },
```

Εικόνα 4.15: Εξαγωγή του JWT

Σε αυτή τη περίπτωση χειριζόμαστε την απάντηση ενός αιτήματος που έγινε από το **mainAxios** instance. Ελέγχουμε για άλλη μια φορά την ύπαρξη του αντικειμένου **window,** και στη συνέχεια την ύπαρξη του **access_token** μέσα στο αντικείμενο **data** του response, καθώς δεν έχουν όλες μας οι απαντήσεις ένα JWT. Συγκεκριμένα για το API μας, αλλά και το πιο σύνηθες, είναι οι απαντήσεις από endpoints όπως το **/login** ή **/refresh-token** να επιστρέφουν κάποιο JWT σχετικό με το Authorization της εφαρμογής.
Αν λοιπόν υπάρχει το **access_token** το αποθηκεύουμε στο **localStorage** με **key "authToken",** το οποίο θα χρησιμοποιούμε σε επόμενα requests όπως είδαμε προηγουμένως.

Η χρήση των δύο παραπάνω interceptors που αναλύσαμε, μέσα σε περίπου 10 γραμμές κώδικα, αναλαμβάνει να χειριστεί την εισαγωγή και εξαγωγή των JWT σε ολόκληρη την εφαρμογή από στα requests και από τα responses αντίστοιχα. Σε περίπτωση που δε χρησιμοποιούσαμε την τεχνική αυτή θα έπρεπε σε κάθε request που στέλναμε να βάζαμε χειροκίνητα ξανά και ξανά το token στο Authorization header. Αντίστοιχα σε κάθε response θα έπρεπε να ελέγξουμε πρώτα την ύπαρξη access_token και να ακολουθήσουμε επαναλαμβανόμενα την διαδικασία αποθήκευσης. Η μέθοδος αυτή μας γλυτώνει πάρα πολύ χρόνο και κάνει τον κώδικα επεκτάσιμο, αν για παράδειγμα στο μέλλον θέλουμε να προσθέσουμε και κάποιο άλλο endpoint που απαιτεί χειρισμό των JWT στο request και το response κομμάτι του.

4.5 React Toast

Κάθε σύγχρονη εφαρμογή, η οποία δίνει τη δυνατότητα στο χρήστη της να αλληλεπιδρά με το Interface και να πραγματοποιεί διάφορες λειτουργίες σε πραγματικό χρόνο, πρέπει να κρατά ενήμερο το χρήστη για το τι συμβαίνει. Αυτό ισχύει είτε όταν ο χρήστης πραγματοποιεί μία ενέργεια και είναι επιτυχής, αλλά και ειδικότερα όταν κάτι πάει λάθος και ουσιαστικά η ενέργειά του δεν κατοχυρώθηκε στη βάση δεδομένων μας.
Σκεφτείτε μια εφαρμογή στην οποία υποβάλετε μια φόρμα, ή κανετε upload μια φωτογραφία ή ένα έγγραφο. Ανεβάζετε τη φωτογραφία αλλά δε βλέπετε κάποια αλλαγή στο interface και αναρωτιέστε αν κάνατε τη διαδικασία σωστά. Στην πραγματικότητα όμως η σύνδεση με τη σελίδα έχει χαθεί λόγω κάποιου τεχνικού προβλήματος την ώρα που κάνατε υποβολή. Η εφαρμογή θα έπρεπε να σας

ενημερώσει πως υπάρχει κάποιο πρόβλημα, παρόλα αυτα δεν το έκανε και ο χρήστης μένει να αναρωτιέται τι ακριβώς έγινε.

4.5.1 Χρήση του Toastify

Αυτοί είναι οι λόγοι που αποφασίσαμε να προσθέσουμε τη βιβλιοθήκη **React Toastify** στην εφαρμογή μας.Το React-Toastify είναι μια ελαφριά και ευέλικτη βιβλιοθήκη στο React, η οποία προσφέρει μια απλή και αποδοτική μέθοδο για την προσθήκη ειδοποιήσεων **(toasts)** στις εφαρμογές. Αυτές οι ειδοποιήσεις είναι ιδανικές για να δείχνουν μηνύματα επιβεβαίωσης, προειδοποιήσεις ή ενημερώσεις στον χρήστη με έναν διακριτικό και φιλικό προς τον χρήστη τρόπο. Η βιβλιοθήκη **React Toastify** παρέχει components και API για την εμφάνιση και διαχείριση των toasts. Τα **toasts** είναι σύντομα μηνύματα που εμφανίζονται στην οθόνη για μια συγκεκριμένη διάρκεια και στη συνέχεια εξαφανίζονται αυτόματα.
Η φιλοσοφία μας στην περίπτωση της εφαρμογής είναι τα global toasts, δηλαδή ασχέτως σε ποια σελίδα βρισκόμαστε, να εμφανίζονται οι απαραίτητες ενημερώσεις. Για να το λόγο αυτό, αφου εγκαταστήσουμε το πακέτο **react-toastify,** ακολουθούμε μια παρόμοια λογική με το React Context. Προσθέτουμε στο **Layout** της εφαρμογής ένα **ToastContainer,** το οποίο κάνει ουσιαστικά τα **toast** μας διαθέσιμα σε οποιαδήποτε σελίδα ανήκει μέσα στο **<main>** tag του **Layout.**

```
import "react-toastify/dist/ReactToastify.css";
import { ToastContainer } from "react-toastify";
const Layout: React.FC<LayoutProps> = ({ children }) => {
  return (
    <>
      <CssBaseline />
      <SWRConfig
        value={{ ···
        }}
      >
        <DrawerContextProvider> ···
        </DrawerContextProvider>
        <div>
          <ToastContainer
            theme="colored"
            // toastStyle={{ backgroundColor: "gray" }}
            position="top-center"
            autoClose={5000} // The toast will close after 5 seconds
            progressStyle={{ backgroundColor: "#30608d" }}
            newestOnTop={false}
            closeOnClick
            rtl={false}
            pauseOnFocusLoss
            draggable
            pauseOnHover
          />
          <main>{children}</main>
        </div>
      </SWRConfig>
    </>
  );       You, 5 months ago • .tsx for ItemList, loadingScene,NewsCar
};
```

Εικόνα 4.16: Χρήση του ToastContainer

Όπως βλέπουμε το **ToastContainer** δέχεται ορισμένα **props** τα οποία μπορούμε να ορίσουμε ως default. Το **autoClose** είναι η διάρκεια που ορίζουμε να παραμένει η ειδοποίηση προτού εξαφανιστεί

(5 δευτερόλεπτα). Με το **progressStyle** θέτουμε το χρώμα της μπάρας προόδου(μεταξύ άλλων) η οποία δείχνει πόσο θα κάνει η ειδοποίηση να εξαφανιστεί. Τα props **draggable, pauseOnHover** και **closeOnlick** έχουν να κάνουν με την αλληλεπίδραση που πιθανόν να έχει ο χρήστης με τις ειδοποιήσεις. Άλλα props όπως το **theme,** μας αφήνουν αργότερα να παραμετροποιούμε το χρώμα της ειδοποίησης και το **position** το που θα εμφανιστεί.
Μια σημαντική λεπτομέρεια είναι πως πρέπει να κάνουμε **import** τα **css styles** που είναι σχετικά με τη βιβλιοθήκη του **React Toastify** όπου θέλουμε να χρησιμοποιήσουμε το **ToastContainer,** κάτι που κάνει την εφαρμογή μας λιγότερο αποδοτική, αλλά μας προσφέρει πάρα πολλές επιλογές για το πως θα εμφανίσουμε και θα διαχειριστούμε τα **toast** μας.

4.5.2 Generic-use Toasts

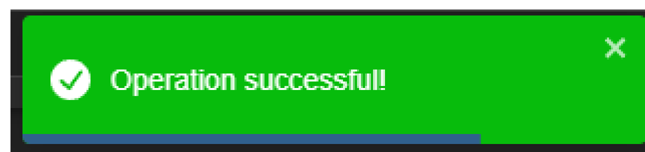
Παρακάτω θα δούμε σύντομα πως εφαρμόσαμε συνδυαστικές τεχνικές για να δημιουργήσουμε γενικά notifications για περιπτώσεις επιτυχίας αλλά και σφαλμάτων.

4.5.2.1 Success Toasts

```
mainAxios.interceptors.response.use(
  (response) => {
    // If the response comes from a POST or PUT request
    if (
      response.config.method === "post" ||
      response.config.method === "put" ||
      response.config.method === "delete"
    ) {
      // Display success toast notification
      toast.success("Operation successful!", { theme: "colored" });
    }
```

Εικόνα 4.17: Χρήση του toast.success σε interceptor

Στο παραπάνω παράδειγμα χρησιμοποιούμε τη δημιουργία ενός **toast.success** μέσα σε ένα **response interceptor** που αναφέραμε και προηγουμένως. Σε αυτό το block κώδικα υπάρχουν μόνο responses των οποίων τα requests ήταν επιτυχημένα. Οπότε, ελέγχοντας την μέθοδο που χρησιμοποιήσαμε μέσω του **config,** αν αυτή είναι **post, put** ή **delete,** τότε πρόκειται για ένα mutation στη βάση. Δηλαδή ο χρήστης έκανε μια ενέργεια (υποβολή φόρμας, συναλλαγή κλπ) η οποία καταγράφηκε με επιτυχία στη βάση δεδομένων μας. Σε αυτή λοιπόν την περίπτωση δημιουργούμε ένα **toast** το οποίο ενημερώνει τον χρήστη πως η ενέργεια που εκτέλεσε ήταν επιτυχής. Το **.success** χρησιμοποιεί κάποια **default styling props** που είναι συνηθισμένα σε ειδοποιήσεις τέτοιου τύπου.



Εικόνα 4.18: Παράδειγμα toast.success

Η παραπάνω ειδοποίηση εμφανίζεται σε οποιοδήποτε response έρχεται που ουσιαστικά προκύπτει από κάποιο interaction του χρήστη με την εφαρμογή μας. Αποφεύγουμε έτσι το να χειριζόμαστε τις ειδοποιήσεις για τέτοια responses ξεχωριστά (αν και έχουμε την επιλογή για εξαιρέσεις).

4.5.2.2 Error Toasts

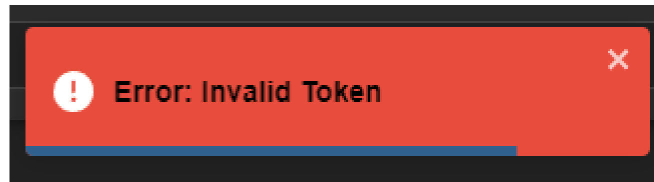
Αντίστοιχη λογική χρησιμοποιούμε και πάλι με το συνδυασμό των interceptors, για τον χειρισμό σφαλμάτων.

```
mainAxios.interceptors.response.use(
  (response) => {…
  },
  (error) => {
    if (
      error.response.status === 409 &&
      error.response.data.errorCode === "_2FaEnabled"
    ) {
      // Handle the 409 error for two-factor authentication enabled
      const customError = new Error("Two-factor authentication is enabled.");
      customError.message = "twoFactorEnabled";
      return Promise.reject(customError);
    }
    // Handle errors globally and show a notification
    const message =
      error.response && error.response.data && error.response.data.message
        ? error.response.data.message
        : "An unexpected error occurred";
    toast.error(
      <div style={{ color: "black" }}>
        <div
          style={{
            fontWeight: "bold",
            display: "flex",
            alignItems: "center",
          }}
        >
          <div>Error: {message}</div>
        </div>
      </div>,
      {
        theme: "colored",
      }
    );
    return Promise.reject(error);        You, 2 months ago • Add automated inte
  }
);
```

Εικόνα 4.19: Χρήση του toast.error σε interceptor

Όπως βλέπουμε στην εικόνα χρησιμοποιούμε το **toast.error** και συνδυασμό jsx για να προσαρμόσουμε την ειδοποίηση που θα προκύψει με αντίστοιχο τρόπο όπως κάναμε με το toast.success. Στην περίπτωση που θέλουμε να μην εφαρμοστεί αυτό το generic toast, όπως βλέπουμε για παράδειγμα σε συγκεκριμένο error που επιστρέφει αναφορικά με το **two-factor authentication,**

χειριζόμαστε νωρίτερα το **Promise** και χρησιμοποιούμε ένα **custom error** το οποίο στέλνουμε με το **Promise.reject** πίσω στο σημείο του request που προκάλεσε το συγκεκριμένο error.



Εικόνα 4.20: Παράδειγμα toast.error

Παρακάτω θα δείξουμε ανά περίπτωση πως χειριζόμαστε άλλες διαφορετικές περιπτώσεις, ώστε να μην εμφανίζεται το generic αυτο toast, αλλά κάτι πιο συγκεκριμένο.

4.6 User Context

Όπως αναφέραμε ήδη, ένα από τα μεγαλύτερα task της εργασίας ήταν να μετατρέψουμε την σελίδα μας σε διαδραστική, **user-related**, web εφαρμογή. Η μετατροπή μιας απλής στατικής web εφαρμογής σε μια διαδραστική και δυναμική πλατφόρμα αποτελεί ένα σημαντικό βήμα στην ανάπτυξη σύγχρονων web εφαρμογών. Σε αυτό το κεφάλαιο, θα εξετάσουμε πώς η δημιουργία ενός **User Context**, σε συνδυασμό με **Custom Hooks** και την χρήση δύο **hook factories** - τα **useUserQueries** και **useUserMutations** - ενίσχυσε τη διαδραστικότητα της εφαρμογής μας.
Σε αυτό το υποκεφάλαιο θα ασχοληθούμε στο πως δημιουργήσαμε τα παραπάνω. Θα αναλύσουμε και θα εξηγήσουμε ποια ήταν η λογική πίσω από τις τεχνικές που χρησιμοποιήσαμε, έχοντας πάντα κατά νου το να καταλήξουμε με μια εφαρμογή που θα είναι, πέρα από δυναμική, εύκολα επεκτάσιμη για μελλοντικά νέα features.

4.6.1 Δημιουργία UserContext και UserContextProvider

Στο κεφάλαιο 3 δείξαμε τη βασική λειτουργία του **React Context.** Χρησιμοποιώντας αυτές τις τεχνικές δημιουργούμε λοιπόν ένα **userContext**, μαζί με ένα component **UserContextProvider**, το οποίο θα κάνει wrap όλη την εφαρμογή μας.

```
interface UserContextData {
  isLoggedIn: boolean;
  setIsLoggedIn: (loggedIn: boolean) => void;
  user: User | null;
  setUser: (user: User | null) => void;
  handleLogout: () => void;
}

You, 4 months ago | 1 author (You)
interface UserProviderProps {
  children: ReactNode;
}

const UserContext = createContext<UserContextData | undefined>(undefined);

export const UserContextProvider: React.FC<UserProviderProps> = ({
  children,
}) => {
```

Εικόνα 4.21: Δημιουργία UserContext και UserContextProvider

Στην παραπάνω εικόνα μπορούμε να δούμε τα πεδία που θα χρησιμοποιήσουμε μέσα στο component του **UserContextProvider.**

- **isLoggedIn:** boolean πεδίο για να ελέγχουμε αν ένας χρήστης έχει έγκυρο logged in session.
- **setIsLoggedIn:** setter function για το **state** του **isLoggedIn**
- **user:** Το αντικείμενο του current user. Περιέχει όλα τα δεδομένα που θα λάβουμε ή έχουμε λάβει από το backend.
- **setUser:** setter function για το **state** του **user.**
- **handleLogout:** void function με την οποία χειριζόμαστε το τι θα γίνει όταν ο χρήστης κάνει logout.

Αρχικά, μέσα στο **UserContextProvider,** αρχικοποιούμε τα states των πεδίων που αναφέραμε, όπου αυτό είναι απαραίτητο.

```
const [isLoggedIn, setIsLoggedIn] = useState(
  !!localStorage.getItem("authToken")
);
const userItem = localStorage.getItem("user");
const [user, setUser] = useState<User | null>(
  userItem ? JSON.parse(userItem) : null
);
useEffect(() => {
  if (isLoggedIn) {
    localStorage.setItem("user", JSON.stringify(user));
  } else {
    localStorage.removeItem("authToken");
    localStorage.removeItem("user");
  }
}, [isLoggedIn, user]);
```

Εικόνα 4.22: Διαχείριση πεδίων του UserContextProvider

Επίσης, χρησιμοποιώντας το **useEffect** hook, τοποθετούμε ή αφαιρούμε αντικείμενα στο **localStorage,** με γνώμονα πάντα το **isLoggedIn** που λειτουργεί στην περίπτωση μας ως flag για το state του χρήστη.

```
const handleLogout = () => {
  // Clear the authToken and reset the user state
  localStorage.removeItem("authToken");
  localStorage.removeItem("user");
  sessionStorage.removeItem("emailVerificationToastShown");
  setIsLoggedIn(false);
  setUser(null);
};      You, 4 months ago • Add route filtering on auth sta
```

Εικόνα 4.23: Συνάρτηση handleLogout

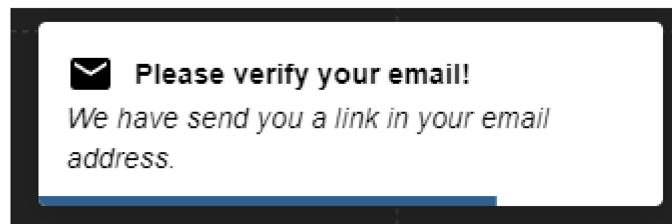
Κάτι παρόμοιο κάνουμε και στη void συνάρτηση **handleLogout,** όπου ουσιαστικά επαναφέρουμε το state της εφαρμογής στην αρχική του κατάσταση**.**

Μία ακόμη λειτουργία που χειριζόμαστε μέσα σε αυτό το **context,** είναι η ειδοποίηση προς το χρήστη, για να επιβεβαιώσει το email του, σε περίπτωση που δεν το έχει κάνει.

```
useEffect(() => {
  // If user is not verified and we haven't shown the toast in this session
  if (
    user &&
    !user.emailVerified &&
    !sessionStorage.getItem("emailVerificationToastShown")
  ) {
    toast(
      <div style={{ color: "black" }}>
        <div
          style={{
            fontWeight: "bold",
            display: "flex",
            alignItems: "center",
          }}
        >
          <EmailIcon style={{ marginRight: "10px" }} />
          <div>Please verify your email!</div>
        </div>
        <div style={{ fontStyle: "italic" }}>
          We have send you a link in your email address.
        </div>
      </div>
    );
    // Mark toast as shown for this session
    sessionStorage.setItem("emailVerificationToastShown", "true");
  }
}, [user]);
```

Εικόνα 4.24: Κώδικας για email verification toast

Όπως βλέπουμε στον κώδικα, ελέγχουμε αν το πεδίο **user.emailVerified** δεν είναι **true** και κάνουμε επίσης χρήση του **sessionStorage** του browser για να δούμε αν δεν υπάρχει κάποιο αντικείμενο με key **"emailVerificationToastShown".** Αν οι τρεις αυτοί έλεγχοι περάσουν, τότε δείχνουμε το παρακάτω **toast**.



Εικόνα 4.25: Verify email toast

Ο λόγος για την προσθήκη του ελέγχου για το κλειδί **"emailVerificationToastShown"** στο **sessionStorage,** είναι το ότι σε κάθε load μιας σελίδας, ή αλλαγή του global state, ο κώδικας μέσα στο **UserContextProvider,** θα ξανατρέξει, οπότε ο χρήστης θα βλέπει το notification ξανά και ξανά, κάτι που ήταν ενοχλητικό.
Εν τέλει επιστρέφουμε τα δεδομένα αυτά, μετά από όλους αυτούς τους ελέγχους και τις μετατροπές που κάναμε στο state τους, με τον εξής τρόπο:

```
const contextValue: UserContextData = {
  isLoggedIn,
  setIsLoggedIn,
  user,
  setUser,
  handleLogout,
};      You, 4 months ago • Add UserContextProvider for global user state.

return (
  <UserContext.Provider value={contextValue}>{children}</UserContext.Provider>
);
};
```

Εικόνα 4.26: Επιστροφή των values του Provider

Το **exported** function component **UserContextProvider** είναι έτοιμο να χρησιμοποιηθεί. Πήγαινουμε λοιπόν στο main αρχείο της εφαρμογής μας, στην περίπτωσή μας το **_app.tsx,** και κάνουμε wrap όλο το **Layout,** ώστε το **context** να ανταποκρίνεται σε οποιαδήποτε αλλαγή συμβαίνει στο state των παιδιών του **Layout.**

```
function MyApp({ Component, pageProps }: AppProps) {
  return (
    <ThemeContextProvider>
      <UserContextProvider>
        <Layout>        You, 4 months ago • Add UserCo
          <Component {...pageProps} />
        </Layout>
      </UserContextProvider>
    </ThemeContextProvider>
  );
}
```

Εικόνα 4.27: Χρήση UserContextProvider στο _app.tsx

Στο αρχείο που κάνουμε export το **UserContextProvider,** κάνουμε επίσης export και το παρακάτω hook:

```
export const useUserContext = () => {
  const context = useContext(UserContext);
  if (!context) {
    throw new Error("useUserContext must be used within a UserProvider");
  }
  return context;
};
```

Εικόνα 4.28: Export του useUserContex hook

Το hook αυτό θα μπορούμε να το χρησιμοποιούμε στα υπόλοιπα react components μας εφόσον, αυτά ανήκουν μέσα στο στο **UserContextProvider,** δηλαδή μέσα στο Layout.

4.6.2 Δημιουργία user custom hooks

Στο δεύτερο κομμάτι του integration του **user context** μέσα στην εφαρμογή μας, αποφάσισα να δημιουργήσω αρχεία τα οποία περιέχουν από ένα JS factory το καθένα. Η λογική που ακολούθησα είναι πως θα ήταν αρκετά πιο εύκολο και βολικό, όλα τα user related **mutations/actions** και αντίστοιχα όλα τα **queries**, να είναι μαζεμένα σε ένα κεντρικό σημείο στον κώδικα μας, όσο το δυνατόν περισσότερο.[29]

4.6.2.1 useUserQueries

Το πρώτο αρχείο που δημιούργησα ήταν το **useUserQueries.ts**. Το αρχείο αυτό έχει ουσιαστικά ως σκοπό να μαζέψει όλα τα queries που πρόκειται να γίνουν στον πίνακα **User** της βάσης μας. Και εδώ όπως και στο επόμενο hook που θα δούμε, χρησιμοποιώ ένα JS factory. Ουσιαστικά δημιουργώ ένα arrow function, το οποίο κάνει return όλα τα υπόλοιπα functions που κάνουν calls στο API μας. Με τον τρόπο αυτό έχω ένα scope και ένα κεντρικό σημείο χειρισμού του state και του context της εφαρμογής μου.

```
export const useUserQueries = () => {      You, 2 mont
  const [loading, setLoading] = useState(false);
  const { setUser, setIsLoggedIn } = useUserContext();

  const fetchUserInfo = async () => {
    setLoading(true);
    try {
      const res = await mainAxios.get("User/info");
      const user = res.data.data as User;
      setIsLoggedIn(true);
      setUser(user);
      console.log("user", user);
      return user;
    } catch (error) {
      console.log(error);
      return undefined;
    } finally {
      setLoading(false);
    }
  };

  return { fetchUserInfo, loading };
};
```

Εικόνα 4.29: Το useUserQueries hook

Στην προκειμένη περίπτωση, σε αυτό το factory, χρειαζόμαστε μόνο ένα returned hook. Η λογική αυτή όμως είναι απόλυτα επεκτάσιμη, καθώς αν μελλοντικά χρειαστεί να προσθέσουμε και άλλα **get** endpoints στον πίνακα **User,** μπορούμε απλά να προσθέσουμε ένα custom hook function με τον ίδιο τρόπο και να το χρησιμοποιήσουμε οπουδήποτε χρειάζεται.

Το **fetchUserInfo** είναι το πιο βασικό user related call στο API μας. Επιστρέφει όλα τα στοιχεία του **User,** με βάση το **access_token,** που θα στείλουμε. Μέσα σε αυτό το hook χρησιμοποιούμε το **loading** και το **setLoading.** Με αυτόν τον τρόπο, κάνοντας return το **loading,** μπορούμε αργότερα να

το χρησιμοποιήσουμε για να ξέρουμε σε οποιοδήποτε σημείο της εφαρμογής ότι περιμένουμε απάντηση από το backend. Για παράδειγμα όσο το **loading** είναι **true,** μπορούμε να έχουμε **disabled** ένα κουμπί ή μια φόρμα, δείχνοντας ταυτόχρονα στο χρήστη το τι συμβαίνει. Το **fetchUserInfo** επιστρέφει το αντικείμενο του χρήστη ως **User** και ταυτόχρονα καλεί το **setUser** του **useUserContext,** ώστε να ενημερώσει το context όλης της εφαρμογής.

4.6.2.2 useUserMutations

Το δεύτερο αρχείο ακολουθεί και αυτό την ίδια λογική. Ονομάστηκε **useUserMutations.ts,** και κάνει export ένα αντίστοιχο factory. Σε αυτό το factory έχουμε μαζέψει όλα τα **actions** που κάνει ο χρήστης με την εφαρμογή και προκαλούν κάποιο **mutation** στη βάση δεδομένων μας.

```
export const useUserMutations = () => {
  const { setIsLoggedIn, user } = useUserContext();
  const { fetchUserInfo } = useUserQueries();
  const [loading, setLoading] = useState(false);

  const toggle2FA = async (enable: boolean) => {…
  };
  const updateSocial = async (…
  };
  /**…
  const loginUser = async (email: string, password: string) => {…
  };
  const claimAccount = async (…
  };
  const claimProduction = async (…
  };
  const uploadUserPhoto = async (…
  };
  const deleteUserPhoto = async (imageId: number) => {…
  };
  const addRole = async (role: string) => {…
  };
  const removeRole = async (role: string) => {      You, 4 weeks ago
  };
  const requestPhoneVerification = async (phoneNumber: string) => {…
  };
  const confirmPhoneVerification = async (…
  };
  const addBio = async (userBioPdf: string) => {…
  };
  const claimVenue = async (id: number) => {…
  };
  return {…
  };
};
```

Εικόνα 4.30: Το useUserMutations hook

Όπως βλέπουμε τα hooks που έχουν δημιουργηθεί είναι πολλά. Χωρίς την ύπαρξη του factory αυτού είναι πολύ εύκολο να κάνουμε κάποια λανθασμένη αλλαγή στο state της εφαρμογής. Σε περίπτωση που όντως κάνουμε λάθος, θα πρέπει να εντωπίσουμε από που προήλθε. Στη συγκεκριμένη περίπτωση οποιοδήποτε mutation-related error που θα προκύψει, προκύπτει από τον κώδικα που βρίσκεται σε αυτό το block κώδικα τις περισσότερες φορές. Αυτό κάνει το debugging απίστευτα πιο απλό και γρήγορο. Η βασική αρχιτεκτονική αυτών των hooks ακολουθούν αυτές τις αρχές:

- **async/await:** Όλα τα hooks μας που ουσιαστικά είναι arrow functions, έχουν την ένδειξη **async.** Εφόσον ξέρουμε ότι θα δημιουργηθούν **Promises** μέσα σε αυτά τα block, τα οποία θα χρειαστεί να κάνουμε **await** για να λάβουμε απάντηση από το API.
- **setLoading(true):** Στην αρχή κάθε hook, αλλάζουμε το state του **loading** σε **true.** Ο κώδικας που θα ακολουθήσει θα καλέσει το API μας και έπειτα θα περιμένει μια απάντηση. Έτσι είμαστε σίγουροι ότι όπου χρησιμοποιηθεί αυτό το hook θα έχουμε ένα σωστό χρονολογικά state.
- **try/catch block:** Το κομμάτι του κώδικα που κάνει call στη βάση μας βρίσκεται πάντα μέσα σε ένα try/catch block. Ο λόγος είναι για να χειριστούμε τυχόν errors που μπορεί να προκύψουν είτε εξαιτίας χαμένης σύνδεσης με το API μας, είτε για οποιονδήποτε άλλο λόγο. Στις περιπτώσεις που λοιπόν που προκύψει κάποιο σφάλμα κατά τη διάρκεια ενός call, η εκτέλεση του κώδικα μας συνεχίζει απευθείας στο **catch block.** Εκεί χειριζόμαστε τα σφάλματα επιστρέφοντας **null** ή **undefined** αναλόγως την περίπτωση.
- **return:** Θεωρητικά τα mutations δεν είναι αναγκαίο να επιστρέφουν κάτι, δηλαδή θα μπορούσαμε να χρησιμοποιήσουμε async void συναρτήσεις. Παρόλα αυτά προτιμούμε να επιστρέφουμε κάτι πάντα, είτε προέκυψε σφάλμα είτε όχι. Σε όλες τις περιπτώσεις λοιπόν επιστρέφουμε ένα **boolean flag** ή κάποιου είδους **CustomAxiosResponse** αναλόγως την περίπτωση.
- **finally:** Μετά το try/catch block, χρησιμοποιούμε το key word **finally.** Ο κώδικας που βρίσκεται μέσα στο finally εκτελείται τελευταίος, ασχέτως αν προέκυψαν σφάλματα ή όχι κατά τη διάρκεια του call.

Παρακάτω θα αναλύσουμε σύντομα πως λειτουργεί κάθε hook, το οποίο δημιουργήθηκε με βάση τα functional requirements της εργασίας που μας ανατέθηκαν:

- **registerUser:**

```
const registerUser = async (email: string, password: string) => {
  setLoading(true);
  try {
    return await mainAxios.post("User/register", {
      email,
      password,
      role: 0,
    });
  } catch (error) {
    console.log("Error in register:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.31: registerUser hook

Το hook αυτό πολύ απλά δέχεται δύο **string** values, για **email** και **password** αντίστοιχα και καλεί το endpoint **"User/register"** και επιστρέφει είτε το αποτέλεσμα είτε **false** σε περίπτωση σφάλματος. Το **role: 0**, δείχνει πως θέλουμε να δημιουργήσουμε account ενός απλού user(όχι admin ή manager).

- **loginUser**

```
const loginUser = async (email: string, password: string) => {
  setLoading(true);
  try {
    const response = await mainAxios.post("User/login", {
      email,
      password,
    });
    //this will not reach if 2fa enabled
    setIsLoggedIn(true);
    await fetchUserInfo();
  } catch (error) {
    // Handle other errors
    if ((error as Error).message === "twoFactorEnabled") {
      console.log("Error in login:", error);
      return -1;
    }
    return false
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.32: loginUser hook

Το **loginUser** έχει αντίστοιχη λογική με το **registerUser.** Αυτό που αλλάζει είναι μια ιδιαιτερότητα στο API μας, στην περίπτωση που είναι ενεργοποιημένο το **two factor authentication,** οπού επιστρέφεται error αντί για επιτυχημένο call στο endpoint**.** Στην περίπτωση αυτή ελέγχουμε αν το error περιλαμβάνει το συγκεκριμένο message. Σε αυτήν την περίπτωση **-1** αντί για **false,** ώστε να χειριστούμε αναλόγως την περίπτωση.

- **addRole**

```
const addRole = async (role: string) => {
  setLoading(true);        You, 4 weeks ago • Add AddUserRoleDialog for
  try {
    return await mainAxios.post(`User/Add-Artist-Role/?role=${role}`);
  } catch (error) {
    console.log("Error in add role:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.33: addRole hook

Το παραπάνω hook προσθέτει ένα ρόλο τη φορά, ως **query parameter** και καλείται για να ενημερώσει το προφίλ του χρήστη.

- **removeRole**

Η λογική εδώ είναι ακριβώς ίδια με το **addRole.** Το μόνο που αλλάζει είναι το endpoint σε **“Remove”** από **“Add”.**

- **toggle2FA**

```
const toggle2FA = async () => {
  setLoading(true);
  try {
    let result = false;
    result = user?._2FA_enabled
      ? await mainAxios.post("User/enable2fa")
      : await mainAxios.post("User/disable2fa");
    return result;
  } catch (error) {
    console.log("Error toggling 2fa", error);
    return false;
  } finally {
    setLoading(false);
  }        You, 21 minutes ago • Uncommitted change
};
```

Εικόνα 4.34: toggle2FA hook

Εδώ μπορούμε να καλέσουμε δύο endpoint, αναλόγως τη τιμή του **user._2FA_enabled**. Το hook αυτό λοιπόν ενεργοποιεί ή απενεργοποιεί το **two factor authentication.**

- **addBio**

```
const addBio = async (userBioPdf: string) => {
  setLoading(true);
  try {
    await mainAxios.post("User/Upload/Bio", userBioPdf);
    return true;
  } catch (error) {
    console.log("Error in add user bio", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.35: addBio hook

Χρησιμοποιούμε αυτό το hook για να κάνουμε upload το βιογραφικό ενός User. Το **userBioPdf** είναι ουσιαστικά ένα αρχείο σε **base64 format.**

- **removeBio**

Ίδια λογική για να αφαιρέσουμε το bio. Δεν δέχεται κάποια παράμετρο, είναι ένα απλό **post** request.

- **claimAccount**

```
const claimAccount = async (
  personId: number,
  identificationDocument: string
) => {
  setLoading(true);
  try {
    return (await mainAxios.post("AccountRequests/RequestAccount", {
      personId,
      identificationDocument,
    })) as AxiosResponse<{
      success: boolean;
      message: string;
      errorCode: string;
    }>;
  } catch (error) {
    console.log("Error in claiming account:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.36: claimAccount hook

Δημιουργήσαμε αυτό το hook για να μπορεί ένας χρήστης να κάνει claim ένα προφίλ καλλιτέχνη. Ουσιαστικά το hook αυτό δημιουργεί ένα **Request**, το οποίο θεωρητικά χειρίζεται από κάποιον admin ή claim Manager. Όταν το Request γίνει δεκτό δημιουργείται σύνδεση ανάμεσα στον **User** και στο προφίλ του **Person. PersonId** το id του καλλιτέχνη και **identificationDocument** αρχείο σε **base64.**

- **claimVenue**

```
const claimVenue = async (id: number) => {
  setLoading(true);
  try {
    await mainAxios.post(`Venues/claim-venue/${id}`);
    return true;
  } catch (error) {
    console.log("Error in claim venue", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.37: claimVenue hook

Αντίστοιχο hook με το προηγούμενο. Στην περίπτωση αυτή το API μας δεν δημιουργεί **Request** και δέχεται το claim απευθείας.

- **claimProduction**

```
const claimProduction = async (
  productionId: number,
  identificationDocument: string
) => {
  setLoading(true);
  try {
    return (await mainAxios.post("ProductionRequests/RequestProduction", {
      personId: productionId,
      identificationDocument,
    })) as AxiosResponse<{
      success: boolean;
      message: string;
      errorCode: string;
    }>;
  } catch (error) {
    console.log("Error in claiming production:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.38: claimProduction hook

Ίδια λογική με το **claimAccount** ακολουθώντας τον πίνακα των functional requirements. Δεν υπάρχει διαθέσιμο αντίστοιχο endpoint στη βάση.

- **updateSocial**

```
const updateSocial = async (
  link: string,
  type: "facebook" | "instagram" | "youtube"
) => {
  setLoading(true);
  try {
    let result = false;
    result = await mainAxios.put(
      `User/@/${type}?link=${encodeURIComponent(link)}`
    );

    return result;
  } catch (error) {
    console.log("Error updating social", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.39: updateSocial hook

Ένα hook που μπορεί να καλέσει 3 διαθέσιμα endpoints, αναλόγως το **type** που θα δώσουμε.

- **uploadUserPhoto**

```
const uploadUserPhoto = async (
  photo: string,
  label: string,
  isProfile: boolean
) => {
  setLoading(true);
  try {
    return await mainAxios.post("User/UploadPhoto", {
      photo,
      label,
      isProfile,
    });
  } catch (error) {
    console.log("Error in uploading user photo:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.40: uploadUserPhoto hook

Το συγκεκριμένο endpoint δέχεται links από ήδη διαθέσιμες φωτογραφίες σε κάποιο δημόσιο domain. Δεν αποθηκεύει κάπου αρχεία φωτογραφιών. Δέχεται επίσης ένα **label** για κάθε φωτογραφία και ένα **isProfile** flag που θέτει τη φωτογραφία ως φωτογραφία προφίλ του χρήστη.

- **deleteUserPhoto**

```
const deleteUserPhoto = async (imageId: number) => {
  setLoading(true);
  try {
    return await mainAxios.delete(`User/Remove/Image/${imageId}`);
  } catch (error) {
    console.log("Error in delete user photo:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.41: deleteUserPhoto hook

Διαγράφει το αντικείμενο **UserPhoto** από τον πίνακα **User** με βάση το **imageId** που θα δώσουμε ως παράμετρο.

- **requestPhoneVerification**

```
const requestPhoneVerification = async (phoneNumber: string) => {
  setLoading(true);
  try {
    await mainAxios.post(
      `User/request-verification-phone-number/?phoneNumber=${encodeURIComponent(
        phoneNumber
      )}`
    );
    return true;
  } catch (error) {
    console.log("Error in add request verification phone number:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.42: requestPhoneVerification hook

Ανοίγει μια σύνδεση με τον αριθμό **phoneNumber** που θα δώσουμε. Η σύνδεση αυτή στέλνει κωδικό OTP στον αριθμό αυτό με SMS.

- **confirmPhoneVerification**

```
const confirmPhoneVerification = async (
  phoneNumber: string,
  verificationCode: string
) => {
  setLoading(true);
  try {
    await mainAxios.post(
      `User/confirm-verification-phone-number/?phoneNumber=${encodeURIComponent(
        phoneNumber
      )}&verificationCode=${verificationCode}`
    );
    return true;
  } catch (error) {
    console.log("Error in add request verification phone number:", error);
    return false;
  } finally {
    setLoading(false);
  }
};
```

Εικόνα 4.43: confirmPhoneVerification hook

Σε αυτό το hook δίνουμε τον κωδικό OTP που στέλνεται μέσω του προηγουμένου, μέσω του **verificationCode**. Πρέπει επίσης να δώσουμε και τον αριθμο **phoneNumber** που δώσαμε πριν.

Χρήση αυτών των hook θα δούμε στα επόμενα μέρη του κεφαλαίου.

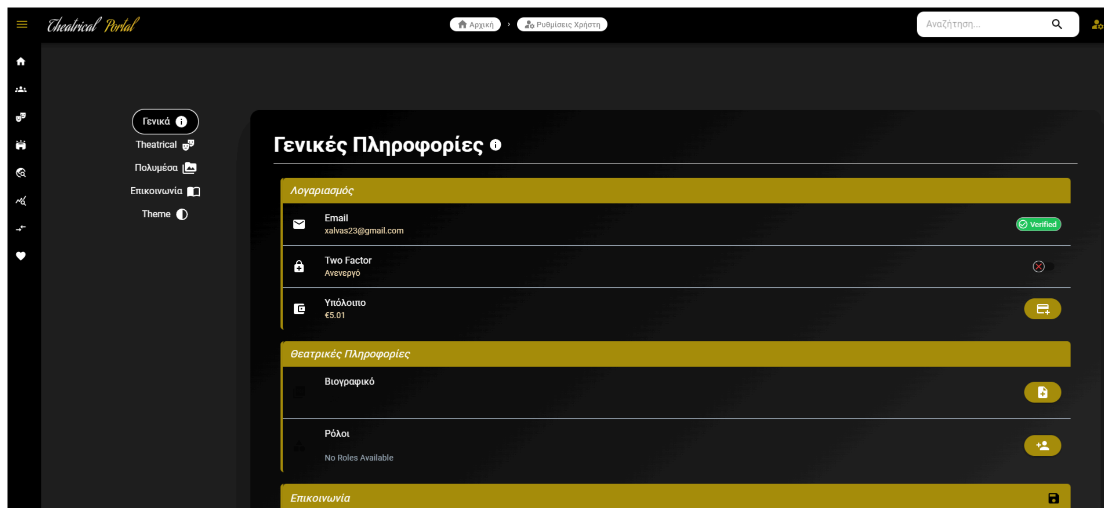
4.7 User page και user actions

Εδώ θα αναφέρουμε τις αλλαγές που κάναμε πάνω στο UI για να μπορούμε να χρησιμοποιήσουμε τα calls που περιγράψαμε προηγουμένως. Αυτό περιλαμβάνει κυρίως την προσθήκη ενός νέου path για την εφαρμογή μας. Το path αυτό είναι το **/user.** Μας κατευθύνει στο **User page,** όπου υπάρχουν λεπτομέρειες για το προφίλ του τρέχον χρήστη και ενέργειες που μπορεί να κάνει πάνω στο προφίλ με βάση τα διαθέσιμα endpoints του API μας.

4.7.1 User page

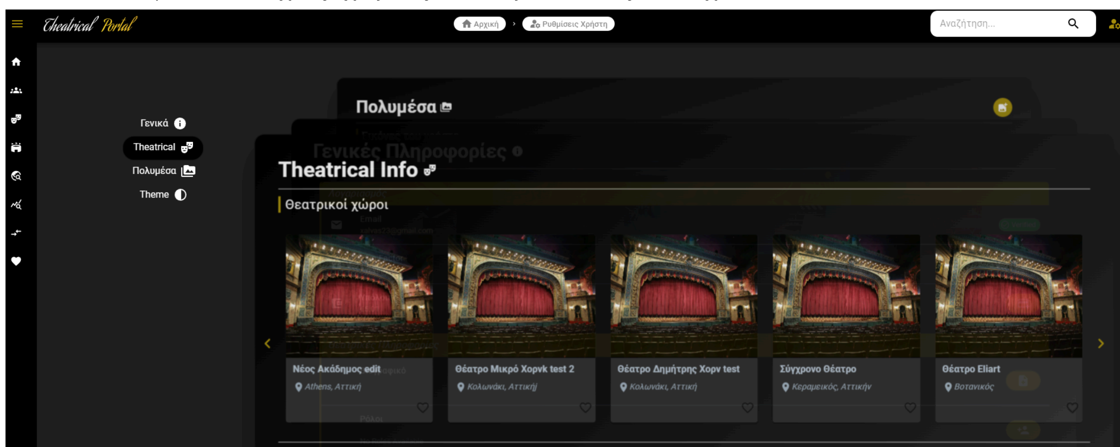
Χωρίσαμε τη σελίδα σε **tabs** τα οποία εναλλάσσονται μεταξύ τους μέσα ενός animation που δημιουργήσαμε με **tailwind** και **framer-motion**. Παρακάτω θα εξηγήσουμε το κάθε tab ξεχωριστά.

- **Γενικά:** Πληροφορίες για τον χρήστη. Μπορεί να ελέγχει βασικές πληροφορίες όπως **πληροφορίες σύνδεσης, υπόλοιπο και συναλλαγές,** καθώς και **πληροφορίες επικοινωνίας.** Επίσης μπορεί να ανεβάσει ένα αρχείο Pdf για το **βιογραφικό** του.



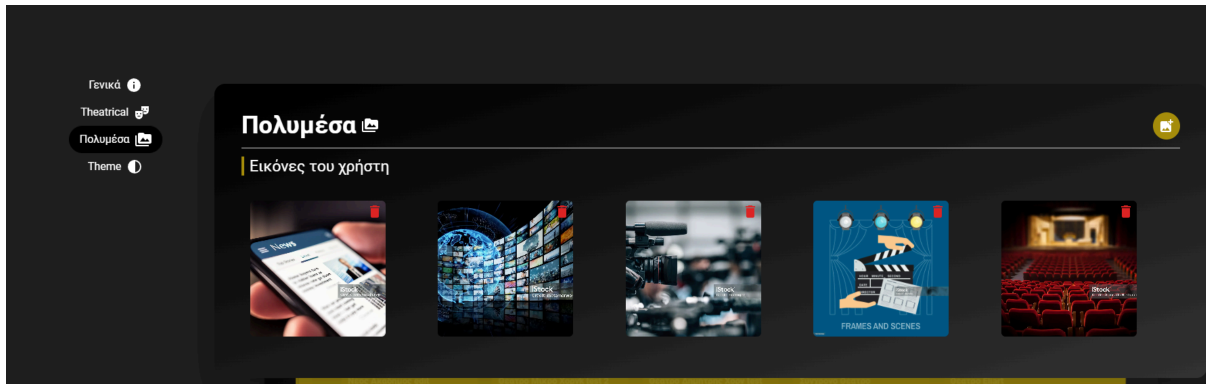
Εικόνα 4.44: General Tab

- **Theatrical:** Σε αυτο το tab ο χρήστης μπορεί να δει αναλυτικά πληροφορίες που έχουν να κάνουν με καλλιτέχνες, χώρους και παραστάσεις που έχει κάνει **claim.**



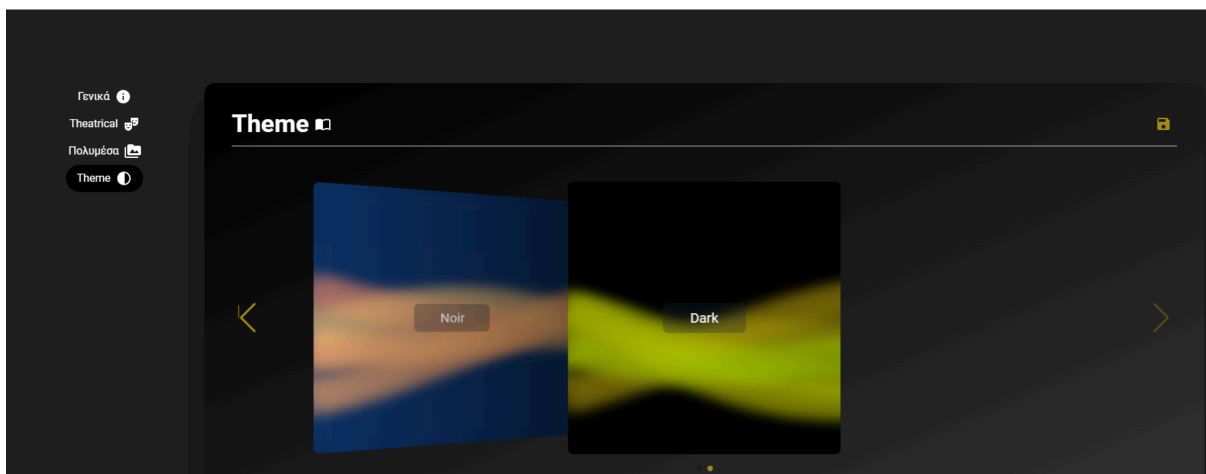
Εικόνα 4.45: Theatrical Tab

- **Πολυμέσα:** Στο tab αυτό ο χρήστης μπορεί να δει τις φωτογραφίες του και να αλλάξει φωτογραφία προφίλ.



Εικόνα 4.46: Πολυμέσα Tab

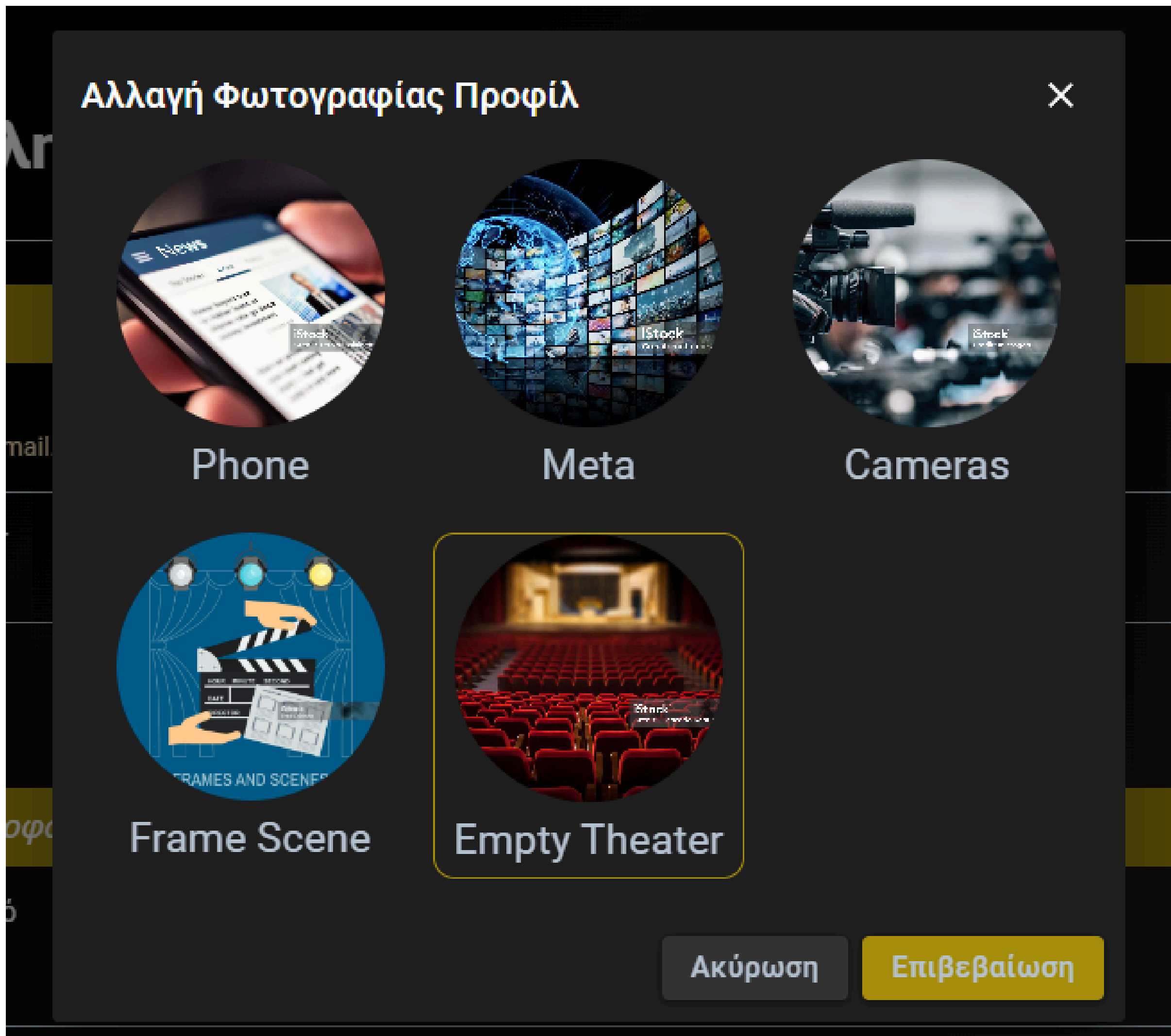
- **Theme:** Εδώ υπάρχει η δυνατότητα να αλλάξει το theme όλης της εφαρμογής. Με έναν συνδυασμό από **global css variables**, **tailwind theme extension** και μικρή παραμετροποίηση του **material ui theme**.



Εικόνα 4.47: Theme Tab

Αναλυτικά όλα τα actions με τη σειρά που αναφέρθηκαν:

- **Save Changes:** Το κουμπί αυτό είναι διαθέσιμο κάθε φορά το state του αντικειμένου του user είναι διαφορετική από το **initial state.** Πιο απλά, όταν κάνουμε κάποια αλλαγή στα inputs των social media. Πατώντας το κουμπί, αποστέλλονται τα απαραίτητα calls στο API μας. Το κουμπί αυτό μπήκε με σκοπό να είναι επεκτάσιμα τα inputs που θα υπάρχουν σε αυτό section της σελίδας.
- **Change Profile Photo:** Ανοίγει το component **SelectProfilePhotoDialog.** Σε αυτό το component μπορούμε να διαλέξουμε μια φωτογραφία που ήδη υπάρχει ή να δώσουμε ένα link για μια νέα φωτογραφία και θα τεθεί αυτόματα ως φωτογραφία προφίλ.

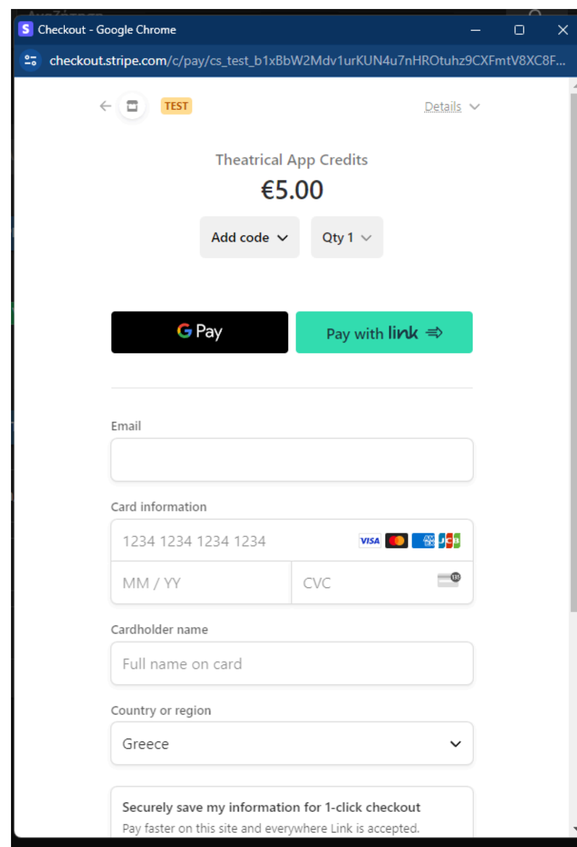


Εικόνα 4.48: Το SelectProfilePhotoDialog

- **Two Factor:** Πρόκειται για ένα **Switch** του Material UI. Το state του καθορίζεται από το **user._2FA_enabled.** Κάνοντας κλικ καλούμε το **toggle2FA** hook.
- **Balance:** Εδώ βλέπουμε τα διαθέσιμα που έχει ένας χρήστης στο προφίλ του. Με το κουμπί **Add Credits** ανοίγουμε ένα **checkout session** σε ένα popup παράθυρο, χρησιμοποιώντας το προκαθορισμένο endpoint.

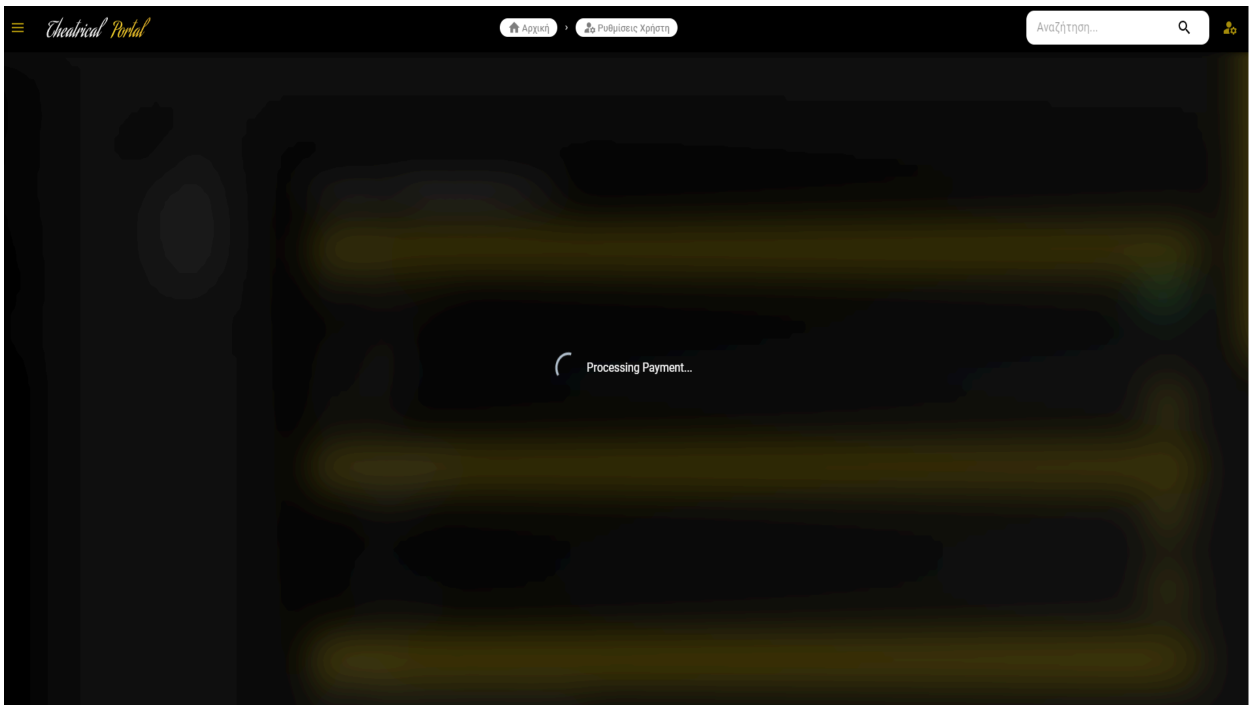
```
const handleOpenPaymentDialog = () => {
  const url =
    "https://theatricalapi.jollybay-0ad0b06b.germanywestcentral.azurecontainerapps.io/api/stripe/create-checkout-session";
  const popup = window.open(
    url,
    "PaymentWindow",
    `width=600,height=800,
    top=${window.screenTop + (window.outerHeight - 800) / 2},
    left=${window.screenLeft + (window.outerWidth - 600) / 2}`
  );

  if (popup) {
    setIsPaymentDialogOpen(true); // Show overlay
    const checkPopupClosed = setInterval(async () => {
      if (!popup || popup.closed) {
        clearInterval(checkPopupClosed);
        setIsPaymentDialogOpen(false); // Hide overlay when popup is closed
        await fetchUserInfo();
      }
    }, 500);
  }
};
```

Εικόνα 4.49: Κώδικας για δημιουργία checkout session



Εικόνα 4.50: Checkout session popup window

Όσο το παράθυρο αυτό είναι ανοιχτό, έχουμε δημιουργήσει ένα overlay, το οποίο ουσιαστικά δεν αφήνει το χρήστη να κάνει κάποιο άλλο action προτού το παράθυρο του checkout έχει κλείσει.



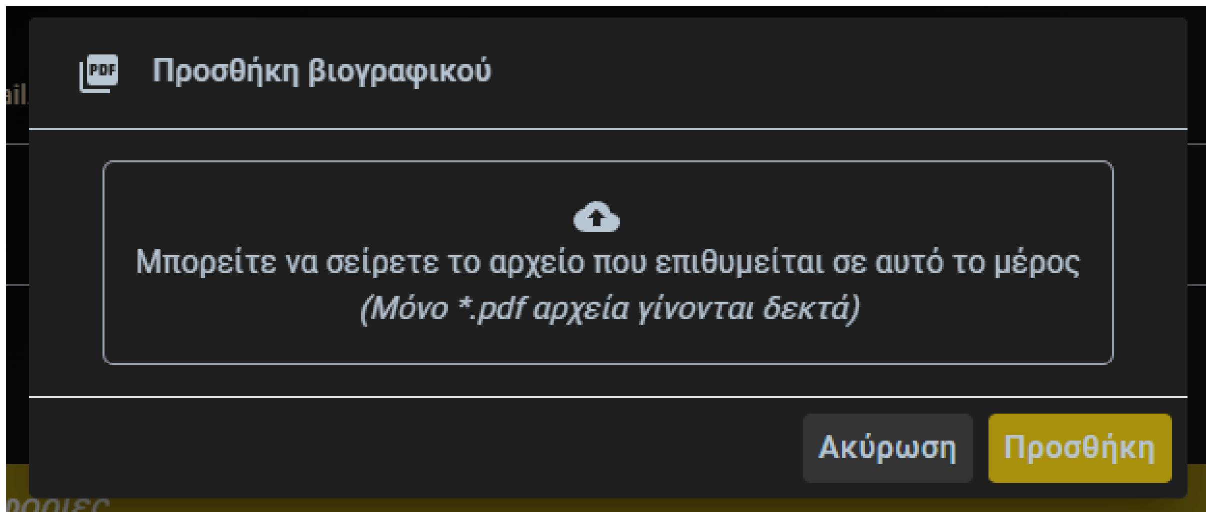
Εικόνα 4.51: Χρηση payment overlay

```
{/* Backdrop for when payment dialog is open */}
<Backdrop style={{ zIndex: 1201 }} open={isPaymentDialogOpen}>
  <div className="flex gap-2 items-center">
    <CircularProgress color="inherit" />
    <div style={{ color: "white" }}>Processing Payment...</div>
  </div>
</Backdrop>
```

Εικόνα 4.52: Κώδικας για Backdrop

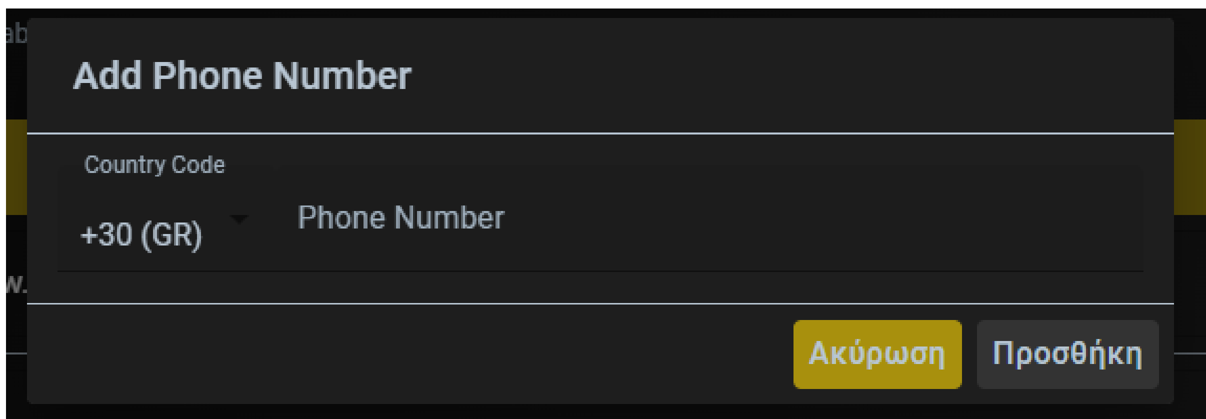
Για το overlay αυτό χρησιμοποιούμε το **Backdrop** του Material UI.

- **Social Media:** Τα 3 inputs που βλέπουμε έχουν ένα απλό formatting check, έτσι ώστε να είμαστε σίγουροι ότι ο χρήστης στέλνει έγκυρα links στο backend.
- **User Bio:** Εδώ βλέπουμε το βιογραφικό του χρήστη, αν είναι διαθέσιμο. Με το κουμπί **Add bio,** ανοίγει το component **AddBioDialog.**



Εικόνα 4.53: Το AddBioDialog component

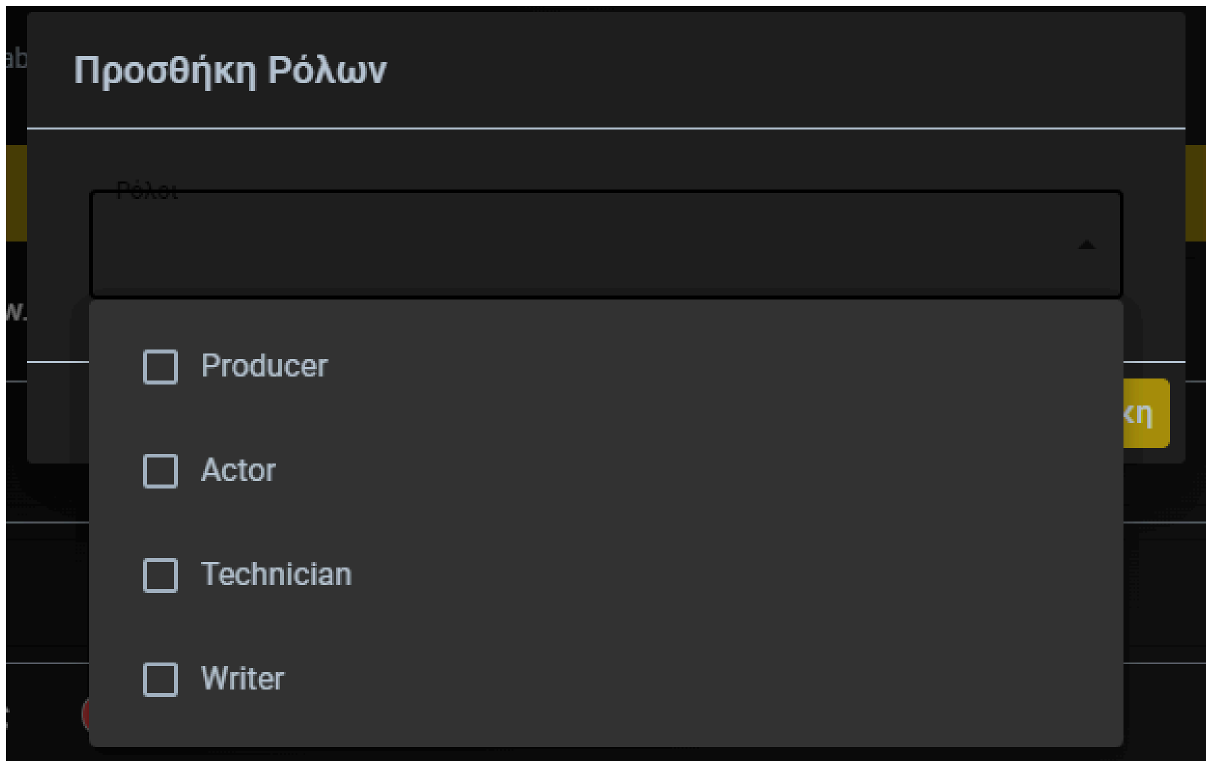
- **Phone number:** Εδώ φαίνεται ο αριθμός τηλεφώνου εφόσον υπάρχει. Το **add phone** ανοίγει το **AddPhoneDialog** component.



Εικόνα 4.54: Το AddPhoneDIalog component

Διαλέγουμε πρώτα ένα συγκεκριμένο country code, γράφουμε τον αριθμό στο input που έχει βασικό formatting check και η ολοκλήρωση καλεί το **requestPhoneVerification** hook.

- **Roles:** Οι υπάρχον ρόλοι που έχει ένας χρήστης. Φαίνονται σε μια λίστα από **Chips,** αν υπάρχουν. Πατόντας το **x** σε ένα chip καλούμε το **removeRole** hook. Το **add role** κουμπί ανοίγει το **AddRolesDialog** component.



Εικόνα 4.53: Το AddRolesDialog component

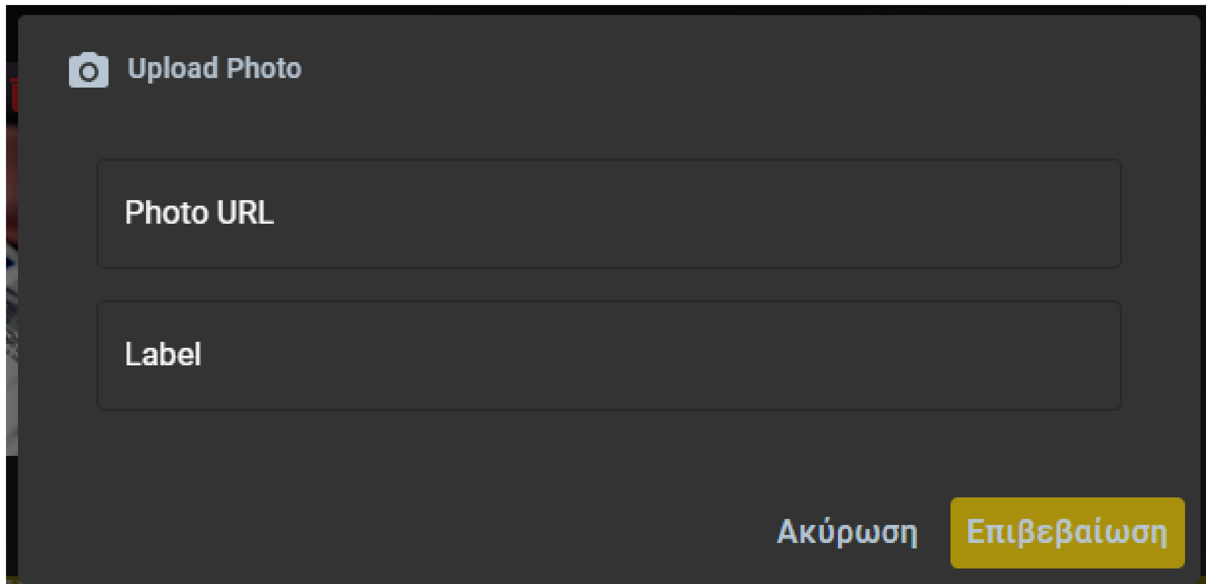
Το component αυτό ανοίγει ένα **Select** του Material UI. Οι ρόλοι που έχει ο χρήστης είναι ήδη **selected.** Στη λίστα που ανοίγει εμφανίζονται τα roles που μας επιστρέφει το **fetchAvailableRoles,** ένα αντίστοιχο Hook που δημιουργήσαμε για να μας φέρνει τους διαθέσιμους ρόλους.

```
<Dialog fullWidth maxWidth="sm" open={isOpen} onClose={onClose}>
  <DialogTitle>Add Roles</DialogTitle>
  <DialogContent>
    <FormControl fullWidth>
      <InputLabel>Roles</InputLabel>
      <Select
        multiple
        value={selectedRoles}
        onChange={handleChange}
        renderValue={(selected) => (selected as string[]).join(", ")}
      >
        {availableRoles.map((role) => (
          <MenuItem key={role} value={role}>
            <Checkbox checked={selectedRoles.includes(role)} />
            <ListItemText primary={role} />
          </MenuItem>
        ))}
      </Select>
    </FormControl>
  </DialogContent>
```

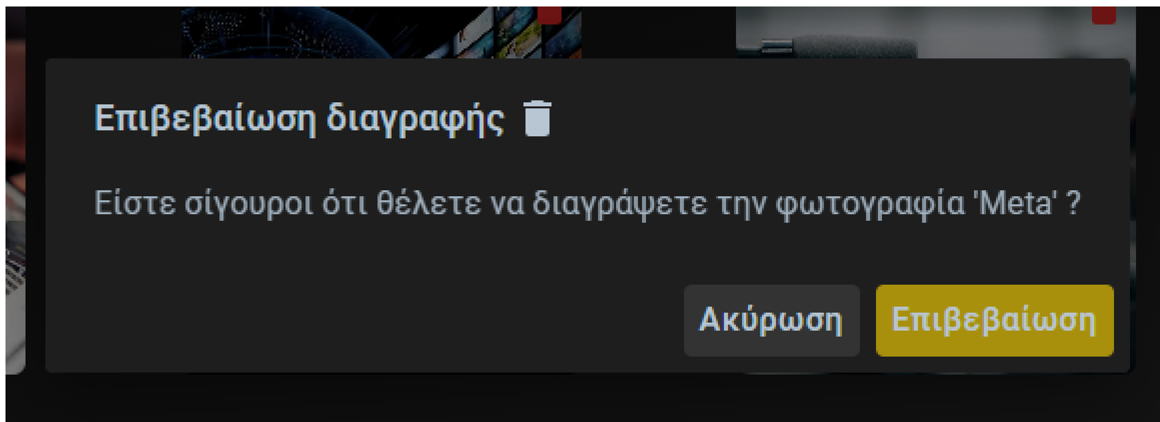
Εικόνα 4.54: Κώδικας του AddRolesDialog

- **Photo Gallery:** Για τις φωτογραφίες του χρήστη δημιουργήσαμε το **UserPhotoCarousel** component. Το component αυτό είναι υπεύθυνο για να δείχνει τις υπάρχουσες φωτογραφίες,

για το action του να ανεβάζεις μια καινούργια φωτογραφία και του να διαγράφεις μία υπάρχουσα.



Εικόνα 4.55: Upload Photo prompt



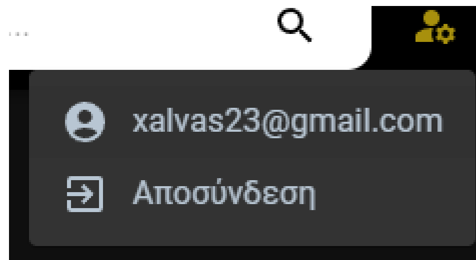
Εικόνα 4.56: Delete Photo prompt

4.7.2 Άλλα User actions

Σε αυτό το κομμάτι θα περιγράψουμε τα υπόλοιπα features που προσθέσαμε σχετικά με το user interaction.

- **Two factor authentication:** Αν ο χρήστης που κάνει **login** έχει ενεργοποιημένο το **two factor authentication** για το λογαριασμό του, πρέπει να συμπληρώσει τον κωδικό που θα του αποσταλεί μέσω email για να ολοκληρώσει τη διαδικασία του login.

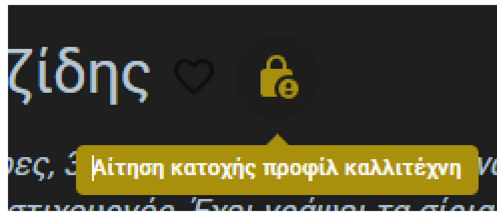
- **User modal:** Όταν υπάρχει συνδεδεμένος χρήστης, το κουμπί του **login,** στα δεξιά του navbar γίνεται ένα κουμπί, το οποίο όταν ανοίγει, ο χρήστης, έχει τις επιλογές να μπει στο προφίλ του ή να κάνει **logout.**



Εικόνα 4.57: User modal

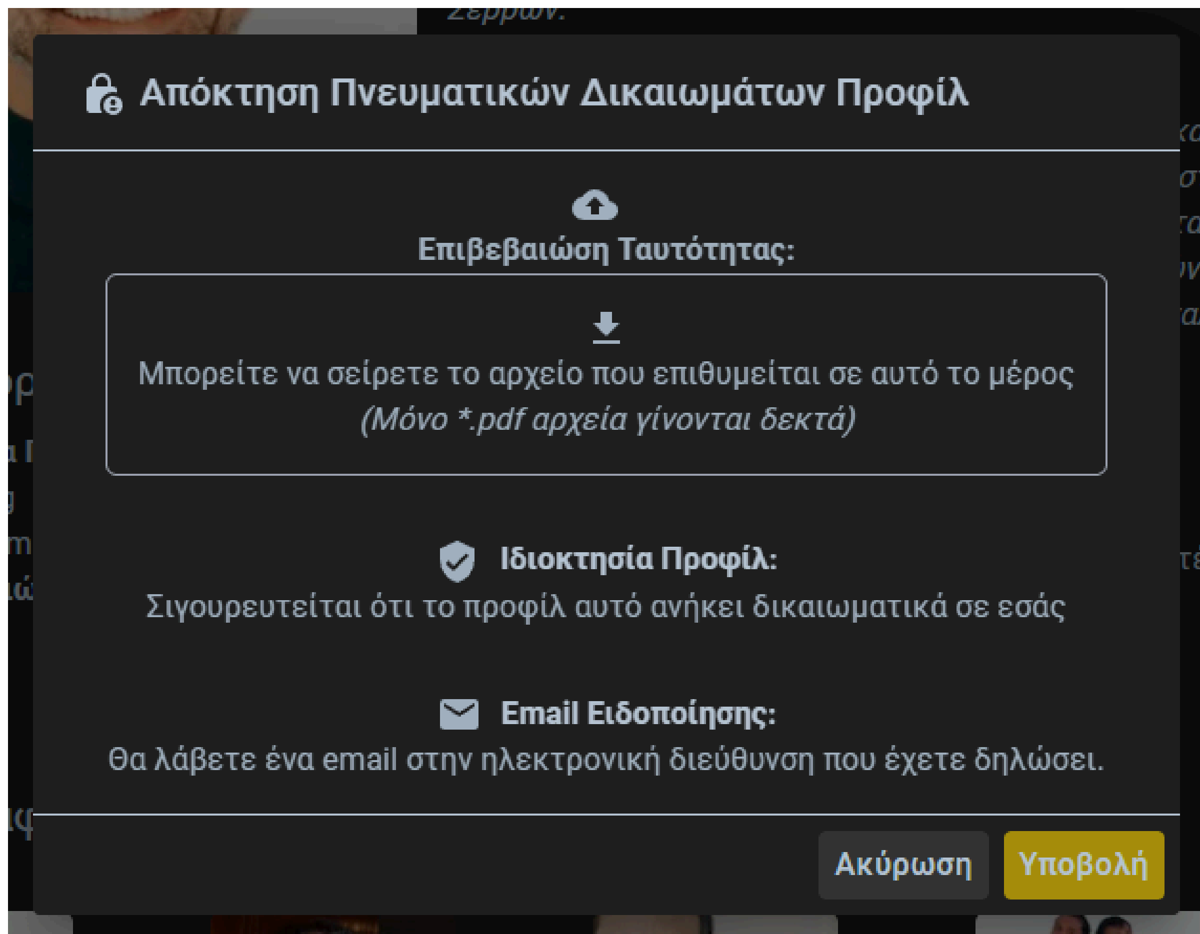
Για τη δημιουργία του χρησιμοποιούμε το **Popover** του Material UI.

- **Claim Person profile:** Όταν ο χρήστης βρίσκεται σε μια σελίδα ενός συγκεκριμένου **artist,** έχει την επιλογή, εφόσον το προφίλ αυτό δεν είναι ήδη claimed, να κάνει claim κάνοντας την απαραίτητη διαδικασία.



Εικόνα 4.58: Κουμπί claim για artists

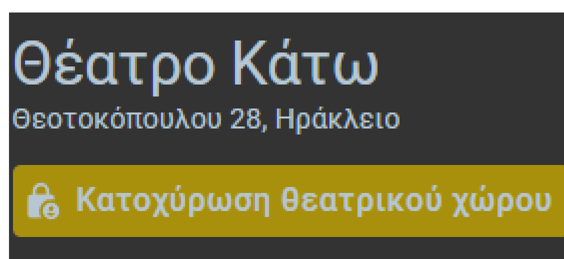
Το κουμπί αυτό ανοίγει το **ClaimPersonDialog** component.



Εικόνα 4.59: Το component ClaimPersonDialog

Μέσα σε αυτό το component μπορούμε να κάνουμε drag and drop ή να διαλέξουμε μέσω ενός file picker, ένα **αρχείο pdf.** Ενημερώνουμε το χρήστη για τη διαδικασία που πρέπει να ακολουθήσει. Το component αυτό καλεί το **claimAccount** hook.

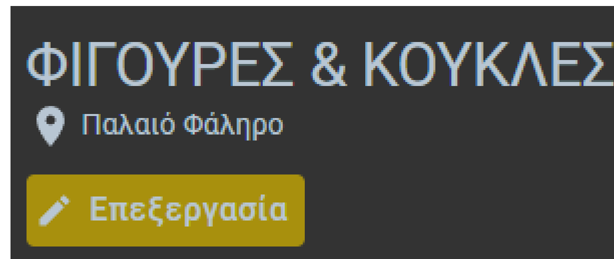
- **Claim Venue:** Όταν ο χρήστης βρίσκεται στη σελίδα ενός θεατρικού χώρου, μπορεί να κάνει claim τον χώρο αυτό, εφόσον δεν έχει γίνει ήδη.



Εικόνα 4.60: Claim Venue

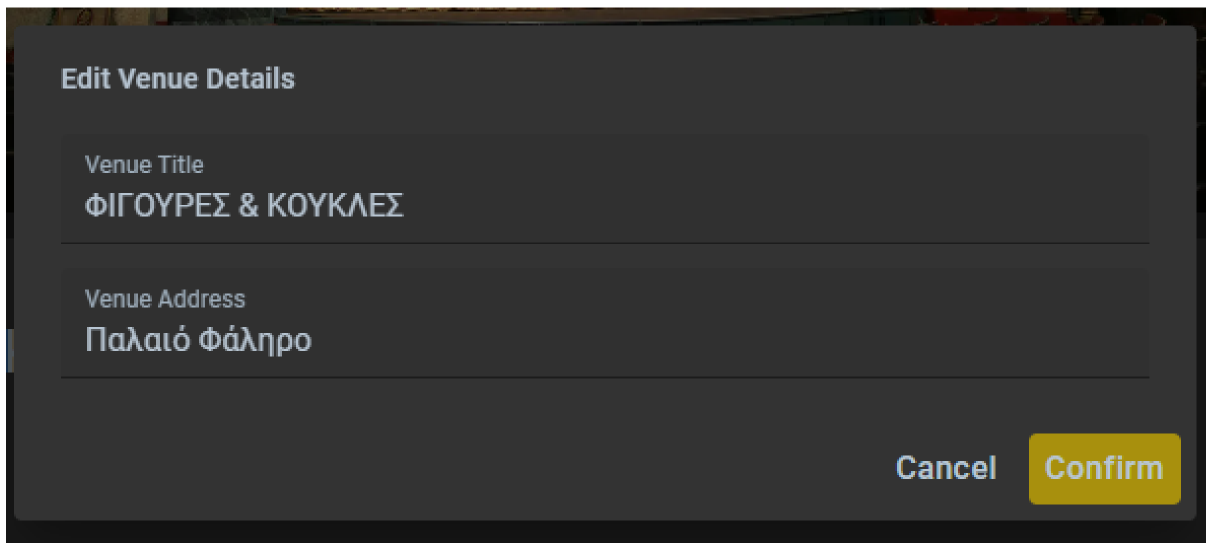
Στην παραπάνω εικόνα εμφανίζεται μόνο ένα από τα δύο κουμπιά ή κανένα σε περίπτωση που ο χώρος είναι ήδη claimed από κάποιον άλλον χρήστη. Το κουμπί **Claim this Venue** καλεί απλά το **claimVenue** hook.

- **Edit Venue:** Αν ο χώρος είναι κατοχυρωμένος ο χρήστης μπορεί να αλλάξει βασικές πληροφορίες.



Εικόνα 4.61: Edit Venue

Το κουμπί **Edit this Venue** ανοίγει το **EditVenueDialog** component.



Εικόνα 4.62: Το EditVenueDialog component

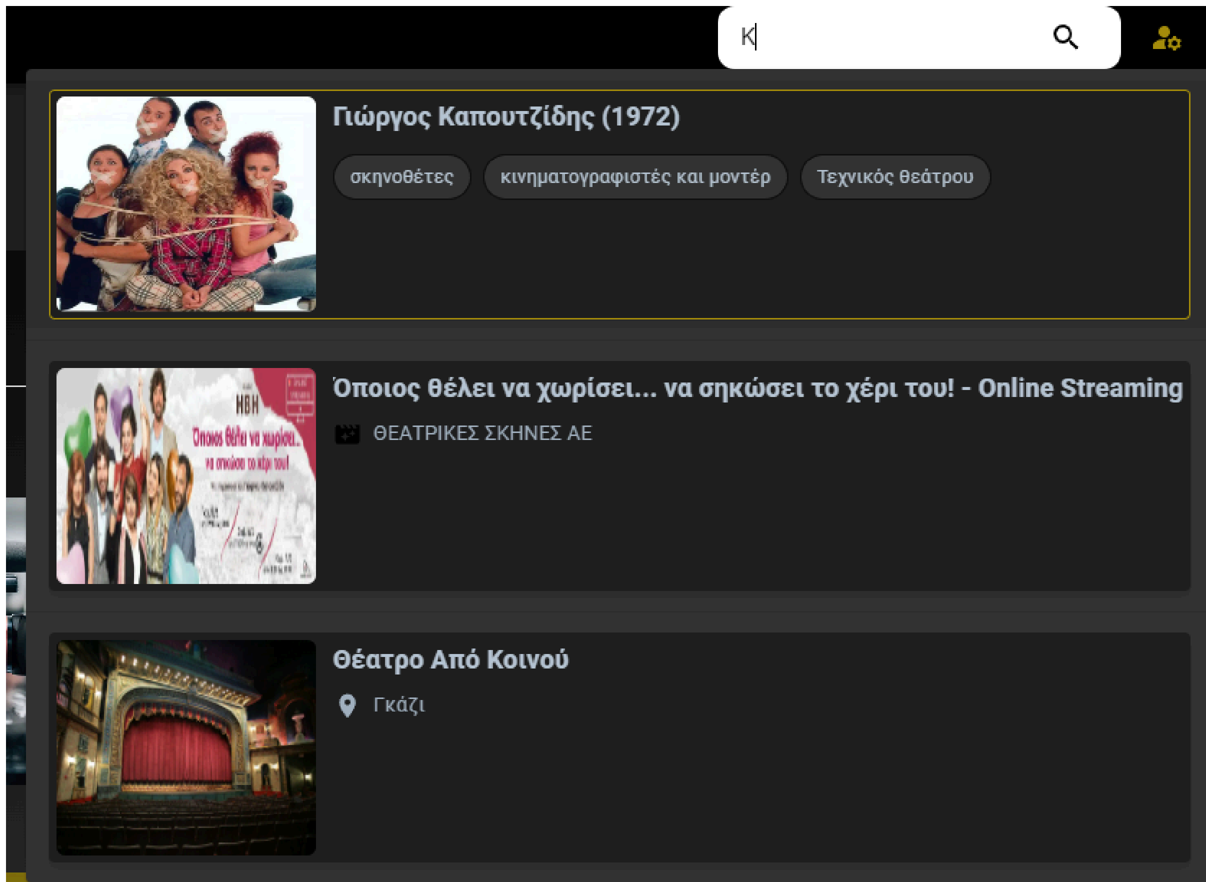
- **Claim Production:** Ίδια λογική με το **claim venue.** Χρησιμοποιούμε το **ClaimProductionDialog** component που είναι ίδιου τύπου με το **ClaimProfileDialog.** Δεν υπάρχει διαθέσιμο endpoint για το claim ενός Production.

4.8 Επιπλέον αλλαγές

Σε αυτό το υποκεφάλαιο θα παρουσιάσουμε συνοπτικά επιπλέον αλλαγές που έγιναν στην εφαρμογή, είτε λειτουργικές είτε στο UI.

4.8.1 Προσθήκη αναζήτησης

Η εφαρμογή μας έχει πλέον ένα κεντρικό **Search bar** στο οποίο μπορούμε να αναζητήσουμε ηθοποιούς, χώρους και παραστάσεις. Η αναζήτηση φιλτράρει απευθείας το input του χρήστη και επιστρέφει τα πιο σχετικά αποτελέσματα. Τα αποτελέσματα μετά παρουσιάζονται σε ένα dropdown χωρισμένο σε sections.



Εικόνα 4.63: Δυναμική αναζήτηση

4.8.2 Breadcrumbs

Στο **NavBar** υπάρχουν πλέον **breadcrumbs** τα οποία δίνουν περισσότερες και άμεσες πληροφορίες στον χρήστη για το που βρίσκεται και κάνουν την περιήγηση πιο απλή και εύκολη.

Εικόνα 4.64: Breadcrumbs

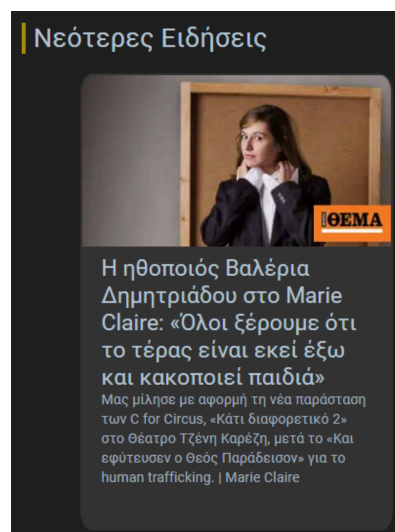
4.8.3 MediaViewer Component

Η προβολή φωτογραφιών στην εφαρμογή μας είναι ένα μεγάλο κομμάτι που το συναντάμε αρκετά συχνά. Με τη δημιουργία του **MediaViewer,** ενός component που προβάλει τις φωτογραφίες μας σε fullscreen με έναν όμορφο animation(framer motion), ο κώδικάς μας γίνεται επαναχρησιμοποιήσιμος

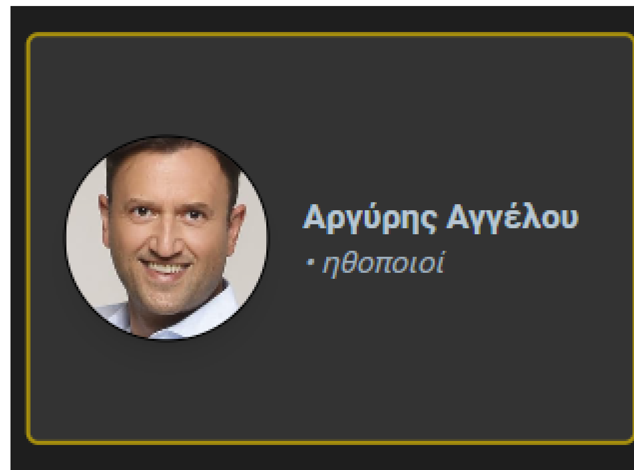
Εικόνα 4.65: MediaViewer Component

4.8.4 Reusable Card Components

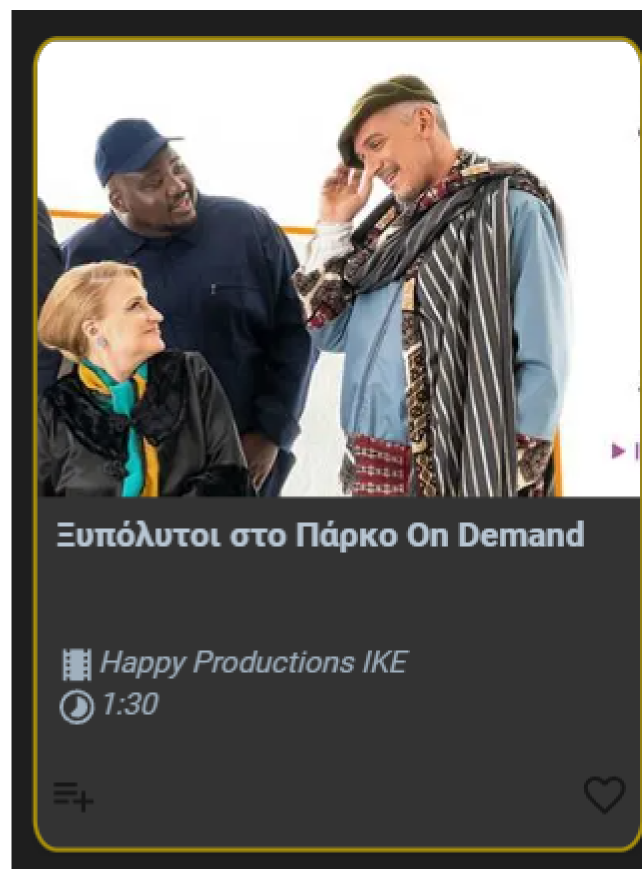
Με την ίδια φιλοσοφία για reusable components, δημιουργήσαμε κάρτες προβολής πληροφοριών. Ένα παράδειγμα είδαμε στο dropdown της αναζήτησης. Παρακάτω θα δείξουμε επιπλέον reusable κάρτες:



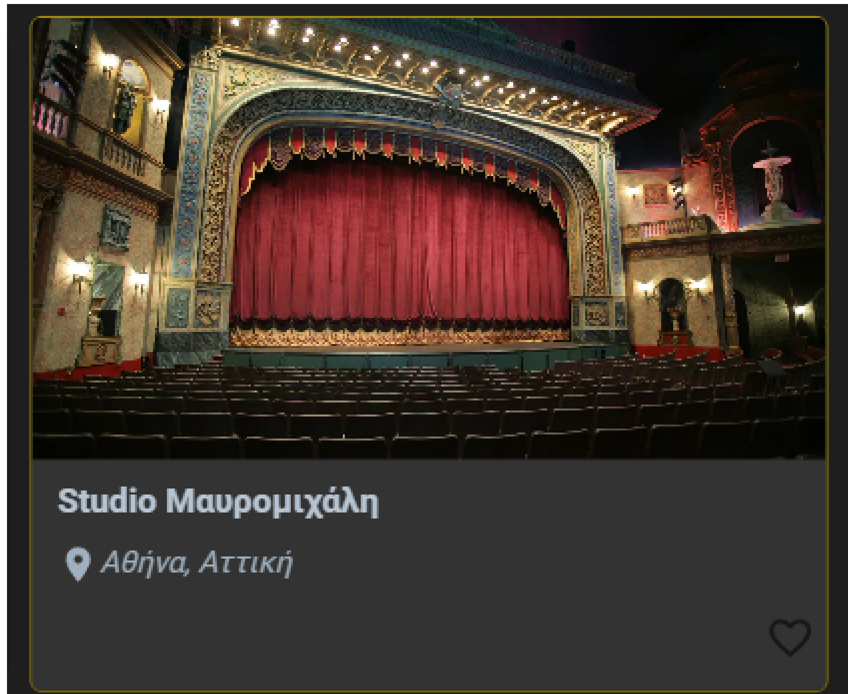
Εικόνα 4.66: News Card



Εικόνα 4.67: Artist Card



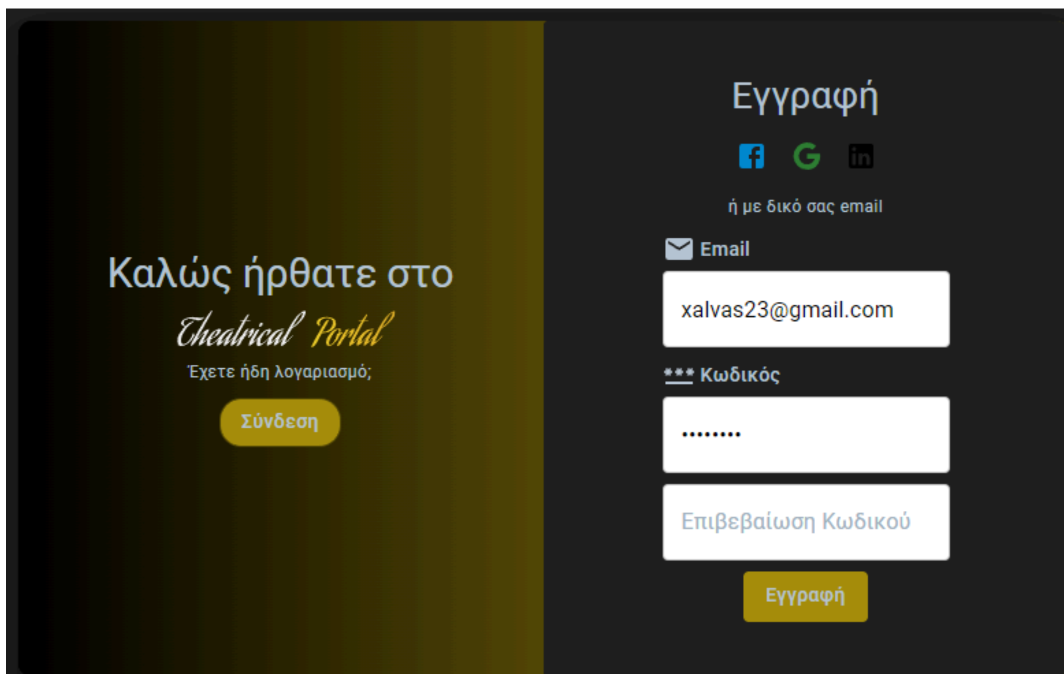
Εικόνα 4.68: Show Card



Εικόνα 4.69: Venue Card

4.8.5 Login page

Δημιουργήσαμε ένα όμορφο animated login page. Ο χρήστης μπορεί να κάνει βασικές λειτουργίες όπως login, login με 2FA και εγγραφή.



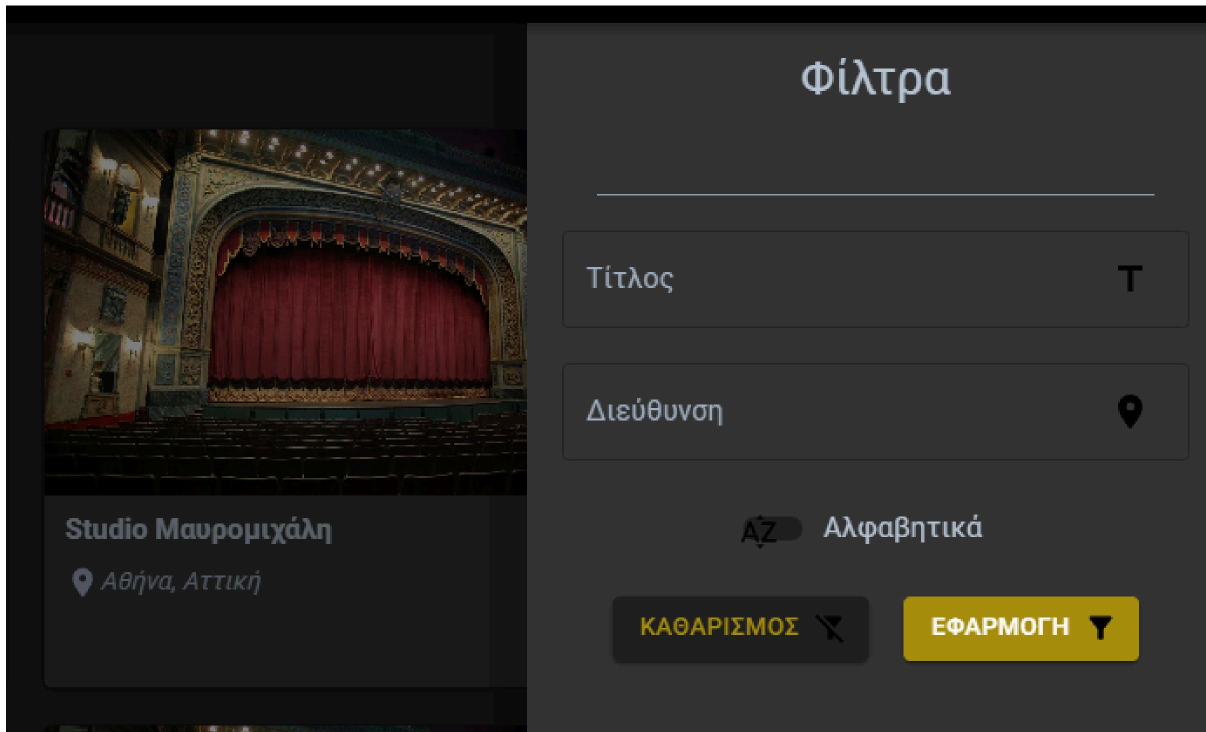
Εικόνα 4.70: Login page

4.8.6 Φίλτρα αναζήτησης

Στις σελίδες των καλλιτεχνών, των παραστάσεων και των χώρων, υπάρχουν χιλιάδες αποτελέσματα. Η εφαρμογή μας δεν θα ήταν λειτουργική εάν δεν προσέφερε τη δυνατότητα αναζήτησης μέσω φίλτρων. Φίλτρα που υπάρχουν:

- **Καλλιτέχνες:** Όνομα, επάγγελμα/ρόλοι και αλφαβητική ταξινόμηση
- **Παραστάσεις:** Τίτλος και Παραγωγός
- **Χώροι:** Τίτλος, διεύθυνση και αλφαβητική ταξινόμηση

Ενδεικτικά δείχνουμε το UI των φίλτρων για του χώρους, με τη χρήση του **Drawer.**



Εικόνα 4.71: Φίλτρα αναζήτησης

4.9 Επίλογος

Σε αυτό το κεφάλαιο δείξαμε κατά κύριο λόγο τις λειτουργίες που προσθέσαμε με βάση των πίνακα των functional requirements που μας δόθηκε στην παραλαβή της εργασίας. Αναλύσαμε κυρίως τις τεχνικές που μας βοήθησαν να μετατρέψουμε την εφαρμογή μας από στατική σε δυναμική και interactive. Δείξαμε πολλές εικόνες, τόσο από σημεία κλειδιά του κώδικα, αλλά και του τελικού αποτελέσματος στην εμφάνιση της σελίδας μας.

Κεφάλαιο 5ο: Προτάσεις για βελτίωση

Στο 5ο και τελευταίο κεφάλαιο θα δώσουμε μερικές προτάσεις προς βελτίωση στα άτομα που θα αναλάβουν την εργασία σε επόμενο εξάμηνο. Θα αναφέρουμε τεχνολογίες που θα ήθελα και εγώ ο ίδιος να χρησιμοποιήσω και που πιστεύω πως θα έκαναν την ανάπτυξη ολόκληρου του project, όχι μόνο στο frontend κομμάτι, πολύ πιο ευχάριστη και παραγωγική. Η μορφή αυτού του κεφαλαίου θα είναι ως εξής:

- Μικρή ανάλυση προβλήματος που υπήρξε κατα τη διάρκεια της εκπόνησης
- Πως το χειριστήκαμε
- Πως θα μπορούσε να γίνει καλύτερα

5.1 Outdated dependencies

Στον κόσμο του προγραμματισμού και πόσο μάλλον σε αυτόν του web development, οι ταχύτητες με τις οποίες τα πράγματα αλλάζουν και προσαρμόζονται είναι τεράστιες. Το project αυτό, που ξεκίνησε από προηγούμενο φοιτητή το 2019, αν και μόλις ούτε 5 ετών, χρησιμοποιεί κάποιες τεχνολογίες οι οποίες είναι ελάχιστα ξεπερασμένες. Η προσθήκη της Typescript ήταν ένα καλό πρώτο βήμα στο να συνεχιστεί με μεγαλύτερη ευκολία η αναβάθμιση της εφαρμογής. Παρακάτω θα αναφέρουμε μερικά κομμάτια του app και ορισμένα libraries που χρησιμοποιεί τα οποία θα ήταν καλό να αναβαθμιστούν στα επόμενα βήματα. Με την προσθήκη της TS, ακολούθησε η αναβάθμιση των:

- **React:** 16 → 18
- **NextJS:** 11 → 14
- **Material UI:** 4 → 5
- **Axios:** 17 → 21
- **Swiper:** 9 → 11

Αυτά ήταν τα major dependencies με την αναβάθμιση και άλλων μικρότερων που δεν αναφέρουμε εδώ αναλυτικά. Το κλειδί για να αποφύγουμε στο μέλλον τόσο μεγάλα migrations είναι οι συνεχής αναβάθμιση και διατήρηση του κώδικα στα σύγχρονα standards.

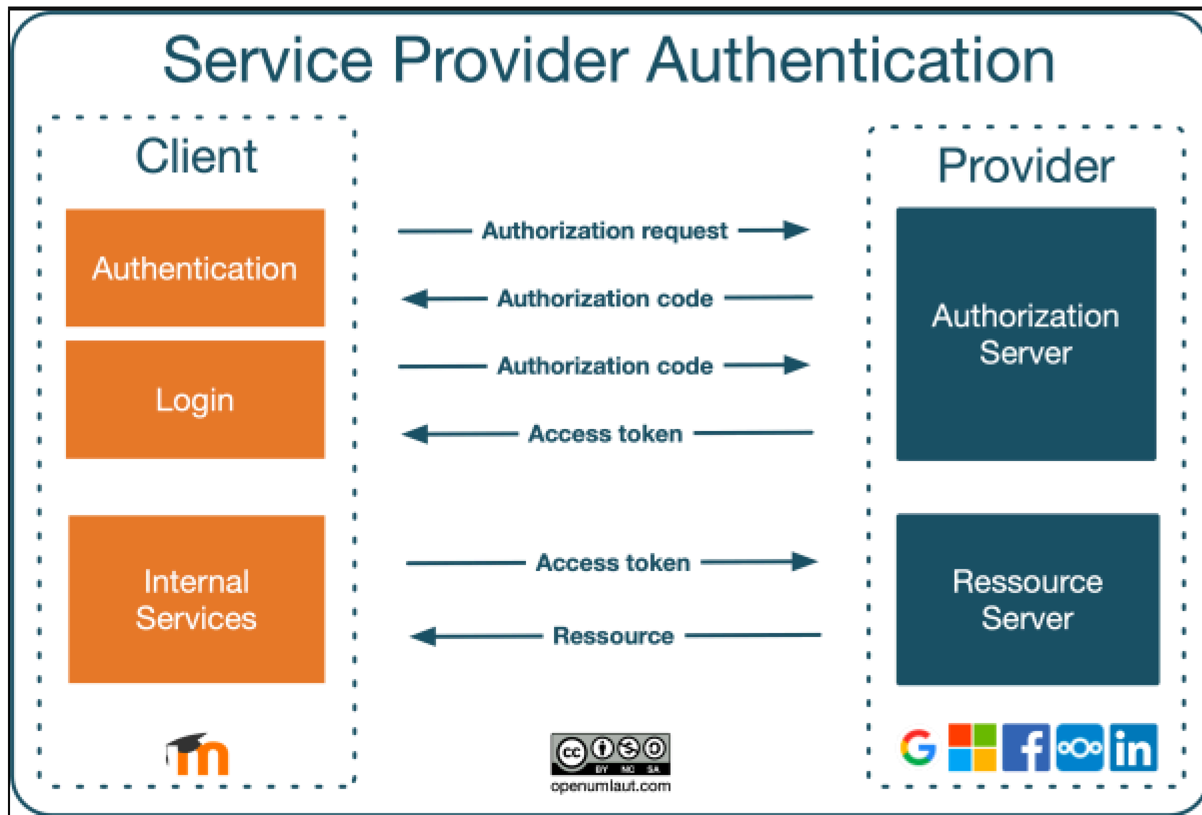
5.2 Documentation

Ένα από τα πρώτα προβλήματα που αντιμετώπισα, ειδικά στις αρχές της ανάπτυξης της εφαρμογής, είναι η έλλειψη documentation. Αν και δεν είναι απόλυτα σύνηθες να υπάρχει documentation για τα components μιας frontend εφαρμογής, εφόσον η φύση της εργασίας είναι να αλλάζει χέρια τακτικά, θα βοηθούσε πολύ να γίνει μια απλή και γρήγορη χαρτογράφηση του πως λειτουργεί η εφαρμογή σε κάθε της κομμάτι.

5.3 OAUTH2

Ένα από τα κύρια μειονεκτήματα της εφαρμογής μας αυτή τη στιγμή είναι η ασφάλεια. Όπως δείξαμε και στα προηγούμενα κεφάλαια, χρησιμοποιούμε **JWT,** για τη μεταφορά, τροποποίηση και αποθήκευση του state της εφαρμογής μας. Αυτό δεν ήταν πρόβλημα όσο η εφαρμογή μας ακολουθούσε τα πρότυπα ενός blog, χωρίς να υπάρχει κάποια ευαίσθητη πληροφορία την οποία παροουσιάζουμε στο χρήστη ή με την οποία ο χρήστης αλληλεπιδρά. Πλέον όμως ζητάμε από το χρήστη να κάνει **login,** να δώσει **αρχεία ταυτοπροσωπίας,** για να ολοκληρώσει συγκεκριμένες

ενέργειες και έχουμε την επιλογή ακόμη και για **συναλλαγές.** Το JWT δεν είναι πλέον αποδεκτός τρόπος για να διασφαλίσουμε την επικοινωνία με το API μας.
Το **OAUTH2** είναι ένα **standard** επικοινωνίας μεταξύ client και ενός host, το οποίο ξεκίνησε να εφαρμόζεται το 2012, αντικαθιστώντας τον προκάτοχό του, **OAUTH1.**[30]

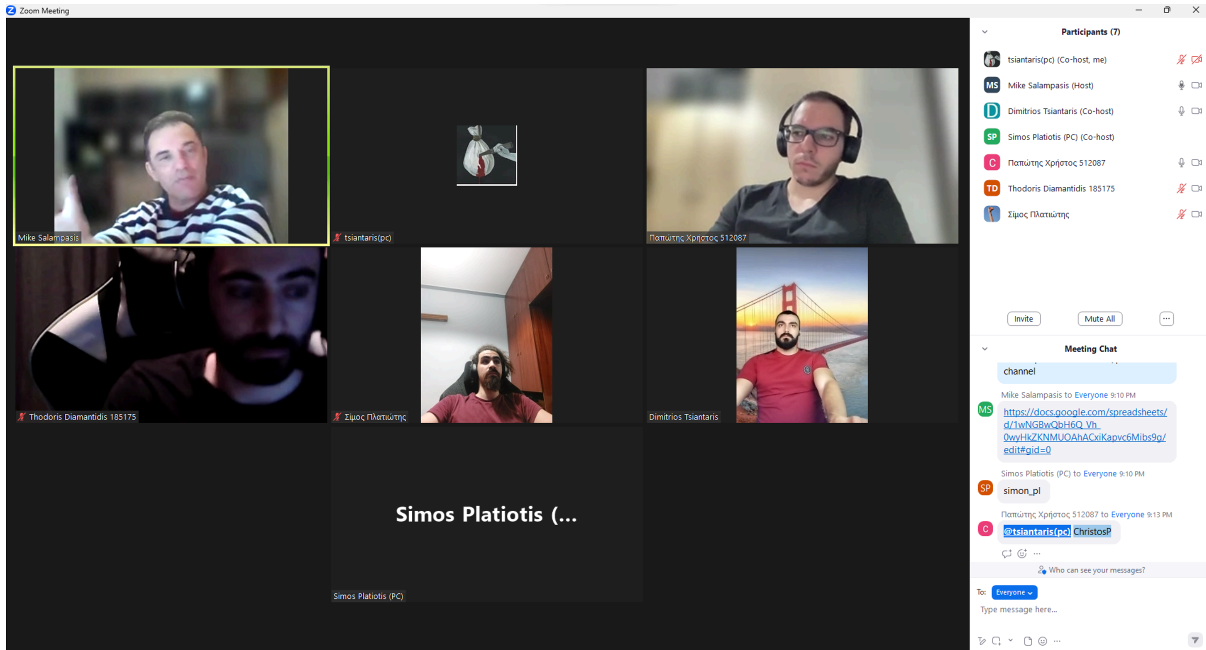


Εικόνα 5.1: Λειτουργία του OAUTH2

Η διαδικασία επικοινωνίας προϋποθέτει περισσότερα βήματα σε σχέση με το JWT. Βασικό και πρώτο βήμα είναι το API μας να ακολουθήσει αυτό το standard και να δέχεται αιτήματα μόνο από εφαρμογές που ακολουθούν και αυτές το ίδιο. Για την περίπτωση μας, συστήνω το **oidc-client-ts,** που είναι μια βιβλιοθήκη γραμμένη σε Typescript και αναλαμβάνει, μετά από κάποια βασική παραμετροποίηση, να ακολουθεί τα standard του OAUTH2 μέσα από τα functions που μας προσφέρει για **login**, **redirect, refreshToken** κλπ.

5.4 Επικοινωνιακό

Σε μία εργασία όπως η δικιά μας, που αποτελείται από πολλά κινούμενα κομμάτια τα οποία διαχειρίζονται διαφορετικά άτομα, η επικοινωνία είναι ένα από τα πιο σημαντικά θέματα. Για να ικανοποιήσουμε τις ανάγκες επικοινωνίας ανάμεσα στα μέλη όλης της ομάδας χρησιμοποιήσαμε κυρίως meetings προκαθορισμένα από τον επιβλέποντα καθηγητή.



Εικόνα 5.2: Διαδικτυακό meeting 31-10-2023

Ο επιβλέπων καθηγητής φρόντιζε σε αυτές τις συναντήσεις να καθιστά όσο το δυνατόν πιο σαφές τα καθήκοντα του κάθε φοιτητή.

5.4.1 Jira Issues

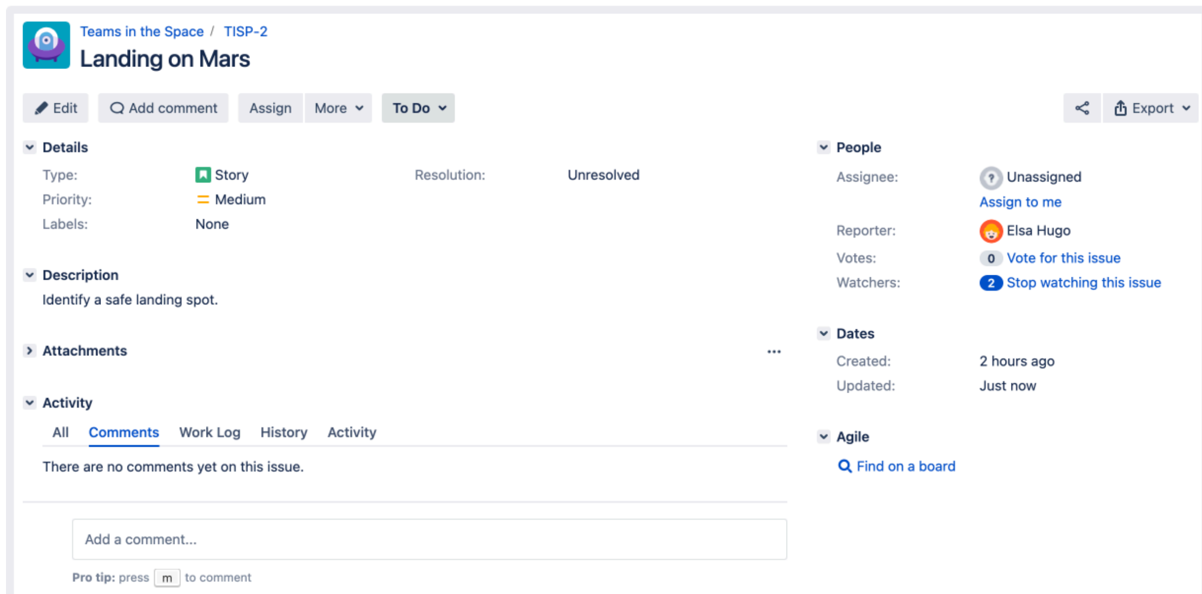
Αν και οι συναντήσεις που αναφέραμε είναι πολύ σημαντικές και εποικοδομητικές, υπάρχουν δυσκολίες όσον αφορά την συχνότητά τους, λόγω της φύσεώς τους. Είναι αρκετά δύσκολο να είναι συχνές, καθώς δεν είναι εφικτός ο συγχρονισμός της καθημερινότητας όλων των φοιτητών και του καθηγητή.

Το **Jira**[31] είναι ένα δημοφιλές λογισμικό διαχείρισης έργων και ζητημάτων, αρχικά σχεδιασμένο για να βοηθά τις ομάδες λογισμικού στη διαχείριση των διαδικασιών **ανάπτυξης** και **εντοπισμού σφαλμάτων**, το **Jira** έχει εξελιχθεί σε μια ολοκληρωμένη λύση που υποστηρίζει διάφορες μεθοδολογίες διαχείρισης έργων, όπως **Agile** και **Scrum**. Το Jira δημιουργήθηκε το 2002 από την Atlassian, μια αυστραλιανή εταιρεία λογισμικού. Ο στόχος του ήταν να παρέχει μια ευέλικτη και προσαρμόσιμη πλατφόρμα για τη διαχείριση εργασιών και ζητημάτων που προκύπτουν κατά την ανάπτυξη λογισμικού.

Κύρια χαρακτηριστικά του Jira:

- **Διαχείριση Ζητημάτων(issue tracking):** Το Jira επιτρέπει την καταγραφή, ταξινόμηση και παρακολούθηση διαφόρων ζητημάτων ή αιτημάτων αλλαγής, διευκολύνοντας την οργάνωση και την προτεραιοποίηση.
- **Υποστήριξη Agile Μεθοδολογιών:** Με ενσωματωμένες δυνατότητες για Agile πλαίσια όπως Scrum και Kanban, το Jira είναι ιδανικό για ομάδες που υιοθετούν ευέλικτες μεθοδολογίες ανάπτυξης.
- **Προσαρμόσιμες Ροές Εργασίας(workflows):** Χρήστες μπορούν να δημιουργήσουν και να προσαρμόσουν τις ροές εργασίας για να αντικατοπτρίζουν τις συγκεκριμένες ανάγκες των έργων τους.
- **Συνεργασία Ομάδας:** Παρέχει εργαλεία για αποτελεσματική συνεργασία ομάδας, όπως την κοινοποίηση εργασιών, σχόλια, και ενσωματωμένες ειδοποιήσεις.
- **Ενσωμάτωση με Άλλα Εργαλεία:** Το Jira μπορεί να ενσωματωθεί με πληθώρα άλλων εργαλείων και υπηρεσιών, προσφέροντας ευρεία συνδεσιμότητα και διασυνδεσιμότητα.

- **Αναφορές και Μετρήσεις:** Παρέχει πλούσια σετ δεδομένων και αναφορών για την ανάλυση της προόδου των έργων και της απόδοσης της ομάδας.



Εικόνα 5.3: Παράδειγμα Jira issue

Με βοήθεια εργαλείων όπως το Jira, η επικοινωνία μια ομάδας μπορεί να γίνει αμέσως πολύ καλύτερη. Αν για παράδειγμα παρατηρήσω ένα **bug** στη λειτουργία ενός endpoint στο API μας, μπορώ να δημιουργήσω ένα **issue,** θέτοντας εμένα ως **reporter.** Μπορώ να κάνω **assign** αυτό το issue στο άτομο που είναι υπεύθυνο να το διορθώσει και να δώσω περιγραφή screenshots ή και να δημιουργήσω ένα **relation** ανάμεσα σε αυτό το issue και κάποιο άλλο. Τα υπόλοιπα άτομα της ομάδας μπορούν να παρακολουθούν την πρόοδο του issue και ο assignee μπορεί να αναφέρει το **issueID** στα **commits** που κάνει στο repository που χρησιμοποιεί η ομάδα.
Αντίστοιχα οι **sys admins** και ο καθηγητής μπορούν να ελέγχουν την πρόοδο τον project και να παίρνουν καθοριστικές αποφάσεις δίνοντας περισσότερες λεπτομέρειες και πιο σαφής οδηγίες, με πιο άμεσο τρόπο, χωρίς να χρειάζεται η απευθείας επικοινωνία μέσω κάποιου meeting.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] World Wide Web [Online]: https://en.wikipedia.org/wiki/World_Wide_Web
[2] Comparison of JavaScript-based web frameworks [Online]:
https://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_web_frameworks
[3] The Science of Semantic HTML [Online]:
https://medium.com/geekculture/the-scienceof-semantic-html-c66fda24f105
[4] JavaScript [Online]: https://en.wikipedia.org/wiki/JavaScript
[5] What is an API? [Online]: https://aws.amazon.com/what-is/api/
[6] API [Online]: https://en.wikipedia.org/wiki/API
[7] Client-side web APIs [Online]:
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs
[8] HTTP request methods [Online]: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
[9] Statelessness in REST API [Online]: https://restfulapi.net/statelessness/
[10] Promise - Javascript [Online]:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
[11] What is axios [Online]: https://axios-http.com/docs/intro
[12] Introducing JSON [Online]: https://www.json.org/
[13] What is npm [Online]: https://www.w3schools.com/whatis/whatis_npm.asp
[14] Web Scripting and its Types [Online]:
https://www.geeksforgeeks.org/web-scripting-and-its-types/
[15] What is Typescript [Online]:
https://www.typescripttutorial.net/typescript-tutorial/what-is-typescript/
[16] JSON Web Token Introduction [Online]: https://jwt.io/introduction
[17] Interceptors | Axios Docs [Online]: https://axios-http.com/docs/interceptors
[18] React - A Javascript library for building user interfaces [Online]: https://legacy.reactjs.org/
[19] JSX [Online]: https://facebook.github.io/jsx/
[20] Components [Online]: https://react.dev/reference/react/Component
[21] Managing State [Online]: https://react.dev/learn/managing-state
[22] Built-in React Hooks [Online]: https://react.dev/reference/react/hooks
[23] React Lifecycle [Online]: https://www.w3schools.com/react/react_lifecycle.asp
[24] Context - React [Online]: https://legacy.reactjs.org/docs/context.html
[25] MUI: The React component library you always wanted [Online]: https://mui.com/
[26] Material UI Theming [Online]: https://mui.com/material-ui/customization/theming/
[27] Introduction to Tailwind CSS [Online]:
https://www.geeksforgeeks.org/introduction-to-tailwind-css/
[28] How Next.js Works [Online]:
https://nextjs.org/learn/foundations/how-nextjsworks/cdns-and-edge
[29] 10 Years of Git: An Interview with Git Creator Linus Torvalds [Online]:
https://www.linuxfoundation.org/blog/blog/10-years-of-git-an-interview-with-git-creator-linus-torvald
s
[30] API consumption best practices in ReactJS [Online]:
https://medium.com/@bhairabpatra.iitd/api-consumption-best-practices-in-reactjs-70873851cd96
[31] What is OAuth 2.0 and what does it do for you? [Online]:
https://auth0.com/intro-to-iam/what-is-oauth-2

[32] Jira Software [Online]: https://www.atlassian.com/software/jira