ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Word Wizardry

"Δημιουργία Παιχνιδιού παρόμοιου με το Λεξομαγεία"

Του φοιτητή

Κούλα Σταύρου

Αρ. Μητρώου: 154476

Επιβλέπων

Κοκκώνης Γεώργιος,
Επίκουρος Καθηγητής

Θεσσαλονίκη 2024

Τίτλος Π.Ε: "Δημιουργία Παιχνιδιού παρόμοιου με το ΛεξοΜαγεία"

Κωδικός Π.Ε: 23139

Ονοματεπώνυμο φοιτητή/τών: Κούλας Σταύρος

Ονοματεπώνυμο εισηγητή: Κούλας Σταύρος

Ημερομηνία ανάληψης Π.Ε: 15-03-2023

Ημερομηνία περάτωσης Π.Ε. …

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 πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η παρούσα πτυχιακή εργασία συντάχθηκε από τον φοιτητή Κούλα Σταύρο υπό την εποπτεία του κ. Κοκκώνη Γεώργιου. Η πτυχιακή εργασία πρόκειται για ένα παιχνίδι σταυρολέξου, το οποίο μοιάζει στο γνωστό παιχνίδι 'Λεξομαγεία'. Σ' αυτή την εργασία θα δούμε τη λογική ανάπτυξης και τον προγραμματιστικό τρόπο υλοποίησης των διαφορετικών ενοτήτων του παιχνιδιού, όπως την εγγραφή-σύνδεση χρήστη, το Main Menu, τη δημιουργία συνδιασμών γραμμάτων από το χρήστη, το χτίσιμο του σταυρολέξου και την υλοποίηση αυτού.

Θέμα Πτυχιακής

Η εφαρμογή θα στηριχτεί στο Γραφικό Περιβάλλον της εφαρμογής ΛεξοΜαγεία και στον τρόπο ανταμοιβής του λύτη των σταυρολέξων, με τις παρακάτω διαφοροποιήσεις : - Το λεξικό θα αποτελείται από όλες τις ελληνικές λέξεις με όλα τα μέρη του λόγου, όπως Ουσιαστικό, Επίθετο, Αντωνυμία, Ρήμα, Μετοχή, (μόνο σε πρώτο πρόσωπο, ενικό ή πληθυντικό αριθμό), Πρόθεση, Επίρρημα, Σύνδεσμος, Επιφώνημα, όλες οι λέξεις με πάνω από 2 γράμματα. - Το ίδιο ισχύει για ξένες λέξεις που έχουν μπει στο ελληνικό λεξιλόγιο (τρυκ, στοκ). - Το ίδιο ισχύει για λέξεις της αρχαίας ελληνικής. - Δεν θα συμπεριλαμβάνονται Ονόματα ανθρώπων, πόλεων, ποταμών κ.λ.π.. - Να επιτρέπεται ο χρήστης να επιλέξει αν τα σταυρόλεξα θα έχουν μοναδική ή πολλαπλές λύσεις. Π.χ. σε κάποιο σταυρόλεξο με κοινό γράμμα το Α και διαθέσιμα γράμματα τα Α,Α,Ε,Κ,Ρ,Τ να μπορεί ο χρήστης να συμπληρώσει μια από τις παρακάτω λέξεις, ΡΑΚΕΤΑ, ΚΑΡΕΤΑ, ΑΡΚΕΤΑ, ΚΑΡΑΤΕ ή ΜΟΝΟ μια από τις παραπάνω λύσεις. - Στις ΕΠΙΠΛΕΟΝ ΛΕΞΕΙΣ να υπάρχουν διαθέσιμες όλες οι λέξεις που μπορούν να σχηματιστούν με τα διαθέσιμα γράμματα. - Τα επίπεδα θα στηρίζονται στον αριθμό των γραμμάτων και στην ευκολία ή δυσκολία των αντίστοιχων σταυρολέξων και θα υπάρχει μια κλιμακωτή δυσκολία. - Ο χρήστης θα μπορεί να αγνοήσει κάποιο επίπεδο, αν το θεωρεί πολύ εύκολο. Η εφαρμογή θα πρέπει να μπορεί να εγκατασταθεί και να τρέχει και σε περιβάλλον Windows και σε Android. Θα πρέπει να χρησιμοποιηθούν διαφορετικά εργαλεία από αυτά στην Διπλωματική 23140, π.χ. Flutter, .NET, vanilla JavaScript jQuery, Bootstrap, React, Vue, Angular, Kotlin.

"Creation of a game similar to LexoMageia"

<<Stauros Koulas>>

Abstract

The application will be based on the Graphical Interface of the LexoMageia application and the reward system for the crossword solver, with the following modifications: - The dictionary will consist of all Greek words across all parts of speech, such as Noun, Adjective, Pronoun, Verb, Participle (only in the first person, singular or plural), Preposition, Adverb, Conjunction, Interjection, and all words with more than two letters.- The same applies to foreign words that have been incorporated into the Greek vocabulary (e.g., "τρυκ", "στοκ"). - The same applies to words from ancient Greek. - Names of people, cities, rivers, etc., will not be included. - The user will be able to choose whether the crosswords will have a single or multiple solutions. For example, in a crossword with a common letter "A" and the available letters being A, A, E, K, R, T, the user will be able to fill in one of the following words: "PAKETA" (RAKETA), "KAPETA" (KARETA), "APKETA" (ARKETA), "KAPATE" (KARATE), or only one of these solutions. - In the ADDITIONAL WORDS, all possible words that can be formed with the available letters should be shown. - The levels will be based on the number of letters and the ease or difficulty of the respective crosswords, with a gradual increase in difficulty. - The user will be able to skip a level if they consider it too easy. - The application should be able to be installed and run in both Windows and Android environments. Different tools should be used from those in Thesis 23140, such as Flutter, .NET, vanilla JavaScript, jQuery, Bootstrap, React, Vue, Angular, Kotlin.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Γεώργιο Κοκκώνη, Επίκουρο καθηγητή του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνές Πανεπιστημίου της Ελλάδος, ο οποίος είναι ο επιβλέπων καθηγητής της συγκεκριμένης πτυχιακής εργασίας, καθώς και τον κ. Κωνσταντίνο Γουλιάνα, ο οποίος μου ανέθεσε τη συγκεκριμένη πτυχιακή εργασία και συνέβαλε σημαντικά στην ανάπτυξη κατά τα αρχικά στάδια .

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Σταυρόλεξα

Στο σύγχρονο κόσμο υπάρχουν αμέτρητες επιλογές σε παιχνίδια σταυρολεξου. Είναι χρήσιμο, σαν πρώτο βήμα στην υλοποίηση ενός παιχνιδιού σε στυλ σταυρολέξου, να εξετάσουμε προσεκτικά τις ήδη διαθέσιμες επιλογές, με σκοπό την έμπνευση και υιοθέτηση λογικών και μεθόδων υλοποίησής του, αλλά και εύρεση στοιχείων και features που λείπουν από αυτές και μπορούν να προστεθούν στη δική μας.

1.2 Παρόμοιες εφαρμογές και διαφορές πτυχιακής από αυτές

Κάποιες από τις πιο γνωστές εφαρμογές για σταυρόλεξα είναι:

- Λεξομαγεία(Words of Wonders – Wordscapes) : ‘Ισως το πιο γνωστό crossword application και η ιδέα πάνω στην οποία βασίστηκε η πτυχιακή. Ο τρόπος λειτουργίας είναι ο εξής: μπαίνοντας σε ένα επίπεδο,εμφανίζεται στο χρήστη ένα κενό σταυρόλεξο,όπως και τα controls,εν προκειμένω ένα πλαίσιο σφαιρικού σχήματος το οποίο περιέχει κάποια γράμματα. ‘Ολες οι λέξεις που υπάρχουν στο σταυρόλεξο σχηματίζονται με συνδιασμό αυτών αποκλειστικά των γραμμάτων. Για να σχηματίσει ο χρήστης ένα συνδιασμό αρκεί να σύρει το δάχτυλό του από το ένα γράμμα της επιλογής του στο άλλο. Αν με το συνδιασμό που επέλεξε υπάρχει μια λέξη στο σταυρόλεξο,αυτή εμφανίζεται. Επίσης,με κάθε επίπεδο που ολοκληρώνει ένας χρήστης αποκτάει κάποιους πόντους,τους οποίους μπορεί να χρησιμοποιήσει ώστε να του εμφανιστεί μία λέξη σε ένα σταυρόλεξο που τον δυσκολεύει. Η εφαρμογή ξεχωρίζει τόσο για τον πρωτοποριακό τρόπο σχηματισμού λέξεων, για το ευχάριστο user interface, καθώς και για την υποστήριξη διαφορετικών γλωσσών,επιτρέποντας έτσι σε χρήστες διαφορετικών υπηκοοτήτων να απολαύσουν το παιχνίδι στη μητρική τους γλώσσα.

- CrossWords: Μια ακόμη εφαρμογή με επίπεδα δημιουργημένα στατικά από τους developers. Εν αντιθέσει με τη Λεξομαγεία,όπου ο χρήστης χρησιμοποιεί ένα πλέγμα στο οποίο σέρνει το δάχτυλο πάνω από γράμματα για να σχηματυίσει λέξεις,το CrossWords παρέχει ένα δικό του πληκτρολόγιο,στο οποίο ο χρήστης πληκτρολογεί την κάθε λέξη. Επίσης, η σειρά με την οποία θα πρέπει να βρει τις λέξεις καθορίζεται από την ίδια την εφαρμογή, παρέχοντάς του το περιεχόμενο της λέξης που πρέπει να βρει. Διαθέτει μια μεγάλη ποικιλία με διαφορετικούς τύπους σταυρολέξων στους οποίους μπορεί να επιλέξει ο χρήστης αφού πρώτα συλλέξει τους απαραίτητους πόντους για το ξεκλείδωμα του κάθε σταυρολέξου,μέσω της ολοκλήρωσης σταυρολέξων και της συλλογής των απαραίτητων rewards.

- CodyCross: Μία εφαρμογή παρόμοια με το CrossWords,πάνω στο ότι χρησιμοποιεί και αυτή πληκτρολόγιο και όχι πλέγμα,καθώς και ότι παρέχει και αυτή μια περιγραφή των λέξεων που πρέπει να βρει ο χρήστης. Παρόλ’αυτα, το CodyCross προσφέρει ένα πολύ προσεγμένο και καινοτόμο design,animated βοηθούς (στα πρότυπα του office assistant) που ανιχνεύουν προβλέματα ή “κολλ’ηματα” του χρήστη, καθώς και ένα δικό του storyline με αυτούς τους βοηθούς, κάνοντας την εμπειρία πιο διασκεδαστική και διαδραστική στο χρήστη.

- WordSurf: Πρόκειται για ένα τελείως διαφορετικό τύπο σταυρολέξου και αρκετά δημοφιλή μάλιστα,όπως δείχνουν τα 10 εκατομμύρια downloads.Στο συγκεκριμένο σσταυρόλεξο, ο ρόλος του χρήστη δεν είναι να συμπληρώσει τις λέξεις που λείπουν. Αντ'αυτού, παρέχεται στο χρήστη ένα πλέγμα με γράμματα,στο οποίο ο χρήστης θα πρέπει να εντοπίσει τις λέξεις που "κρύβονται" μέσα αυτό,σέρνοντας το δάχτυλό του πάνω από τα γράμματα ώστε να σχηματίσει τη λέξη.

Οι διαφορές της εφαρμογής που αναπτύχθηκε στα πρότυπα της πτυχιακής με τις προαναφερθείσες εφαρμογές είναι κυρίως δύο:

1. Η πτυχιακή εργασία δεν παρέχει στατικά σταυρόλεξα στο χρήστη, αλλά δημιουργεί δυναμικά επίπεδα με τη μέθοδο της αναδρομής,από το πρώτο επίπεδο ως το τελευταίο. Αυτό σημαίνει ότι τα σταυρόλεξα που θα αντιμετωπίσει ο χρήστης και οι λέξεις που θα του διατεθούν δημιουργούνται κάθε φορά μέσω του αλγορίθμου που αναπτύχθηκε. Πρακτικά, κανένας χρήστης δεν θα έρθει σε επαφή με σταυρόλεξο που έχει αντιμετωπίσει άλλος χρήστης · το κάθε σταυρόλεξο δημιουργείται τη στιγμή που θα επιλέξει ένα επίπεδο,οπότε η εμπειρία καθίσταται πιο συναρπαστική αλλά και με λιγότερο περιθώριο αλλοτρίωσης. Οι παραπάνω εφαρμογές χρησιμοποιούν είτε μερικώς δυναμικά επίπεδα (συνδιασμό στατικών και δυναμικών επιπέδων), είτε πλήρως στατικά επίπεδα.

2. Στις παραπάνω εφαρμογές,ο χρήστης δεν μπορεί να αποθηκεύσει το επίπεδο στο οποίο βρίσκεται,το σταυρόλεξο που έχει να επιλύσει ή τους πόντους που έχει κατακτήσει μέσα από την ίδια την εφαρμογή. Αυτό σημαίνει ότι αν αλλάξει συσκευή ή αν διαγράψει την εφαρμογή,η πορεία του χάνεται. Η εφαρμογή που αναπτύχθηκε στο πλαίσιο της πτυχιακής διαθέτει δικό της login/sign up system που αποθηκεύει αυτόματα την πορεία του χρήστηστη βάση, επιτρέποντάς του να συνεχίσει την εμπειρία από εκεί που έφτασε,ακόμα και αν μεταβεί σε άλλη συσκευή android ή ακόμα και σε έναν υπολογιστή με λειτουργικό σύστημα Windows. Με αυτό το τρόπο, εξασφαλίζεται ότι δεν πρόκειται να χαθεί η εμπειρία του χρήστη,προλαμβάνοντας τις ανωτέρω καταστάσεις,όπως και το δυσάρεστο σενάριο να ξαναρχίσει ο χρήστης την εφαρμογή από την αρχή,κάτι το οποίο μπορεί να επιφέρει την παραγκώνιση της από το χρήστη.

1.3 Κίνητρα επιλογής και Συνεισφορά της εργασίας

Στην επιλογή του συγκεκριμένου θέματος πτυχιακής οδήγησαν τα κίνητρα που συνοδεύονται με την ολοκλήρωση ενός τέτοιου παιχνιδιού. Κατ' αρχήν, η υλοποίηση ενός τέτοιου παιχνιδιού μπορεί να βοηθήσει έναν front-end developer να αναδείξει αλλά και να εμπλουτίσει τις γνώσεις του, αφού θα χρησιμοποιήσει τεχνολογίες γνώριμες προς αυτόν σε ένα απαιτητικό πλαίσιο. Επίσης, θα βοηθήσει στην ανάπτυξη μίας ευρύτερης κατανόησης πάνω στη λειτουργικότητα και το τρόπο δόμησης του προγραμματισμού, καθώς εκτός από το front-end κομμάτι, θα αναλάβει τη διεκπαιρέωση περεταίρω προγραμματιστικών κομματιών, με την υλοποίηση controllers και βάσης. Ακόμα, η απόδειξη ικανότητας δημιουργίας μίας τέτοιας εφαρμογής μπορεί να αποβεί πολύ σημαντική στην εύρυθμη πορεία ενός προγραμματιστή στην επαγγελματική του ζωή. Τέλος, η γνώση που αποκτήθηκε από το χτίσιμο της εφαρμογής μπορεί να μεταλαμπαδευτεί: να βοηθήσει κάποιον άλλο φοιτητή να υλοποιήσει μία παρόμοια πτυχιακή, κάποιον προγραμματιστή σε μία εφαρμογή που μπορεί να χρησιμοποιεί κομμάτια παρόμοια με αυτά που χρησιμοποιήθηκαν στο Word Wizardry, έναν άνθρωπο που έχει την απορία ως προς το πως υλοποιείται μία εφαρμογή σταυρολέξου.

1.4 Οργάνωση της εργασίας

Στο κεφάλαιο που ακολουθεί, θα αναπύξουμε τις βασικές τεχνολογίες ιστού που χρησιμοποιήθηκαν στη συγγραφή της εργασίας, το Mern Stack, το οποίο είναι αρμόδιο για την ανάπτυξη του κώδικα και τη διασύνδεση με τη βάση, όπως και τις περαιτέρω βιβλιοθήκες που χρησιμοποιήθηκαν.

Στο τρίτο κεφάλαιο, θα δείξουμε το τρόπο ανάκτησης λέξεων που θα χρησιμοποιηθούν στην εφαρμογή. Επίσης, παρουσιάζεται ο τρόπος στελέχωσης της βάσης με αυτές.

Στο τέταρτο κεφάλαιο θα παρουσιάσουμε το Main Menu της εφαρμογής. Αυτό το κεφάλαιο δείχνει το τρόπο της ανάπτυξης του γραφικού περιβάλλοντος της εφαρμογής (UI), περέχει τις ειδικές λειτουργίες που αναπτύχθηκαν για την ένταξη στο σταυρόλεξο ενός μη εγγεγραμένου user, αλλά και την εγγραφή (Sign Up) και την επιτυχή σύνδεση ενός χρήστη (Login) σε αυτό, τόσο σε πλαίσιο front-end, όσο και σε back-end, αναπτύσσοντας τη λογική και τη μέθοδο επικοινωνίας αυτών των πλαισίων .

Στο πέμπτο κεφάλαιο δείχνουμε τη δόμηση του component Staurolexo, το οποίο και θα συντονίζει τα components για το σχηματισμό λέξεων μέσω του πλαισίου και του πίνακα των λέξεων.

Στο έκτο κεφάλαιο παρουσιάζεται η ανάπτυξη και η λειτουργικότητα του πλαισίου Word Swipe, που είναι το πλαίσιο που παρέχει στο χρήστη τη δυνατότητα επιλογής γραμμάτων για το σχηματισμό λέξεων, με στόχο την επίλυση του σταυρολέξου.

Στο έβδομο κεφάλαιο αναπτύσσεται η λογική και ο κώδικας δημιουργίας του ίδιου του σταυρολέξου, μέσω του αρχείου CrosswordArray. Δείχνουμε τους κανόνες που υπήρχαν για τη δημιουργία αυτού, την εναπόθεση λέξεων σε αυτό,τις μεθόδους επίλυσής και την αποθήκευση της πορείας του χρήστη στη βάση. Ακόμα, επιδεικνύεται η λειτουργία του All Words, του πλαισίου που δείχνει στο χρήστη όλες τις λέξεις που θα μπορούσαν να τοποθετηθούν στο σταυρόλεξο που επέλεξε.

Τέλος, στο έβδομο κεφάλαιο, ορίζουμε τη λογική ολοκλήρωσης ενός σταυρολέξου από το χρήστη, το πλαίσιο AllWords που θα παρουσιάζει όλες τις λέξεις που μπορούσαν να χρησιμοποιηθούν στο σταυρόλεξο, όπως και τη τεχνική προσέγγιση της λειτουργίας του 'Continue', που επιτρέπει σε ένα χρήστη να ανακτήσει το στάδιο στο οποίο είχε αφήσει την εφαρμογή.

Κεφάλαιο 2ο: Τεχνολογίες

2.1 Βασικές Τεχνολογίες Ιστού

Ως βασικές τεχνολογίες του ιστού ορίζουμε την HTML, την CSS και την JavaScript. Αυτές οι τρεις τεχνολογίες συνεργάζονται και αλληλεπιδρούν για τη δημιουργία και την εμφάνιση ιστοσελίδων στο διαδίκτυο. Οι βασικές τεχνολογίες ιστού αποτελούν τον πυρήνα της ανάπτυξης ιστοσελίδων και εφαρμογών, επιτρέποντας τη δημιουργία οργανωμένου περιεχομένου, τον σχεδιασμό της εμφάνισης και της αισθητικής μιας σελίδας, καθώς και την ενσωμάτωση διαδραστικών και δυναμικών στοιχείων.

2.1.1 HTML

Η HTML (HyperText Markup Language) είναι το πιο βασικό δομικό στοιχείο του Διαδικτύου. Ορίζει τη σημασία και τη δομή του περιεχομένου του ιστού. Η HTML χρησιμοποιεί "σημειώσεις" για να προσθέσει σχόλια σε κείμενα, εικόνες και άλλα περιεχόμενα για εμφάνιση σε έναν περιηγητή ιστού. Οι σημειώσεις της HTML περιλαμβάνουν ειδικά "στοιχεία" όπως <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <search>, <output>, <progress>, <video>, <ul>, <ol>, <li> και πολλά άλλα.Ένα στοιχείο HTML διαχωρίζεται από το υπόλοιπο κείμενο σε ένα έγγραφο με ετικέτες (tags), οι οποίες αποτελούνται από το όνομα του στοιχείου που περιβάλλεται από "<" και ">". [1]

Με την HTML μπορούμε να δημιουργήσουμε το βασικό κώδικα μίας ιστοσελίδας και να τον εμπλουτίσουμε χρησιμοποιώντας CSS για στυλιστικούς σκοπούς και Javascript, για να ορίσουμε δυναμικές κινήσεις και αλληλεπιδράσεις των αντικειμένων δημιουργημένων από την HTML.

2.1.2 CSS

Ενώ η HTML αχολείται με την περιγραφή και το structure των δεδομένων σε μία ιστοσελίδα, η Cascading Style Sheets (CSS σε συντομογραφία) απασχολείται με το πως θα παρουσιαστούν αυτά τα δεδομένα στην ιστοσελίδα και την μορφή που θα έχουν. Με τη CSS μπορούμε να προσθέσουμε χρώματα στα HTML elements, να διαχειριστούμε τις αποστάσεις μεταξύ τους, τη διάταξή τους στην ιστοσελίδα κ.α. [2]

Η CSS διαχειρίζεται τα στοιχεία HTML επιτρέποντας στους προγραμματιστές να εφαρμόζουν στυλ συγκεκριμένα σε κλάσεις και ID, που αντιστοιχούν σε ομάδες ή μεμονωμένα στοιχεία. Οι κλάσεις (classes) χρησιμοποιούνται για να ομαδοποιήσουν πολλά στοιχεία με παρόμοια στυλ, εφαρμόζοντας τους ένα σύνολο κανόνων με βάση ένα κοινό όνομα κλάσης. Αντίθετα, το ID είναι μοναδικό για κάθε στοιχείο και χρησιμοποιείται για να εφαρμόσει στυλ σε ένα συγκεκριμένο, μοναδικό στοιχείο της σελίδας. Η χρήση κλάσεων και ID στην CSS επιτρέπει την ευέλικτη και στοχευμένη διαχείριση της εμφάνισης των στοιχείων HTML, προσφέροντας ταυτόχρονα δυνατότητες για επαναχρησιμοποίηση στυλ και ακριβή έλεγχο της διάταξης και του σχεδιασμού. [3]

2.1.3 Javascript

Η JavaScript είναι η γλώσσα προγραμματισμού του διαδικτύου, καθώς προδιαγραφή της συμπεριφοράς των ιστοσελίδων. Η συντριπτική πλειοψηφία των σύγχρονων ιστοσελίδων χρησιμοποιεί JavaScript και όλοι οι σύγχρονοι περιηγητές – σε επιτραπέζιους υπολογιστές, κονσόλες παιχνιδιών, ταμπλέτες και smartphones – περιλαμβάνουν διερμηνείς JavaScript, καθιστώντας την την πιο πανταχού παρούσα γλώσσα. [4]

Η JavaScript είναι μια δυναμική, ερμηνευόμενη και μη,γλώσσα υψηλού επιπέδου, κατάλληλη για αντικειμενοστραφή και λειτουργικό στυλ προγραμματισμού. [5] Η γλώσσα Javascript χρησιμοποιήθηκε σε μεγάλο βαθμό στην ανάπτυξη της εφαρμογής. Χρησιμοποιήθηκε για τη δήλωση και παραμετροποίηση μεταβλητών, τον εμπλουτισμό μεγάλων όγκων δεδομένων π.χ. πινάκων, αλλά και τον ορισμό όλων των λειτουργιών της εφαρμογής, μέσω των μεθόδων και των functions που χρησιμοποιεί.

2.2 VS Code

Οι προαναφερθείσες γλώσσες προγραμματισμού απαιτούν έναν επεξεργαστή κώδικα για να αλληλεπιδρούν μεταξύ τους, να παράγουν αποτελέσματα και να παρουσιάζουν αυτά στο χρήστη.Το Visual Studio Code είναι ένας ελαφρύς αλλά ισχυρός επεξεργαστής πηγαίου κώδικα (code editor) που είναι διαθέσιμος για τα Windows, macOS και Linux[6]. Διαθέτει ενσωματωμένη υποστήριξη για JavaScript, TypeScript και Node.js και έχει ένα πλούσιο οικοσύστημα επεκτάσεων για άλλες γλώσσες και χρονικά (όπως C++, C#, Java, Python, PHP, Go, .NET). Το Visual Studio Code επιτρέπει τη γραφή,την επεξεργασία και τον εντοπισμό σφαλμάτων στον κώδικα μέσα από την ίδια την εφαρμογή [7].Διατίθεται δωρεάν από τη Microsoft και είναι open-source. Τα δύο αυτά στοιχεία δικαιολογούν και την μεγάλη ποικιλία διαθέσιμων επεκτάσεων που παρέχει, τα οποία μπορούν να βελτιστοποιήσουν τον κώδικα.

2.3 MERN Stack

Το MERN Stack [8] είναι ένα σύνολο τεχνολογιών που χρησιμοποιούνται για την ανάπτυξη full-stack εφαρμογών. Το MERN αντιπροσωπεύει τα πρώτα γράμματα των βασικών τεχνολογιών MongoDB, ExpressJS, React και NodeJs. Ο λόγος που επιλέχθηκε για την συγκεκριμένη εργασία είναι οι εξής λόγοι. Προσφέρει ολοκληρωμένο περιβάλλον ανάπτυξης που καλύπτει την πλευρά του πελάτη (React), τον εξυπηρετητή (NodeJs και ExpressJs) και τη βάση δεδομένων (MongoDB). Χρησιμοποιεί την JavaScript για όλη την εφαρμογή και την MongoDB ως μια NoSQL βάση όπου την καθιστά πιο εύκολη ως προς την αποθήκευση και διαχείριση των δεδομένων.

2.3.1 React

Η React δημιουργήθηκε αρχικά από μηχανικούς του Facebook για να λύσει τις προκλήσεις που προκύπτουν κατά την ανάπτυξη σύνθετων διεπαφών χρήστη με σύνολα δεδομένων που αλλάζουν με την πάροδο του χρόνου[9]. Δημιούργησαν ένα framework το οποίο,βασισμένο στο MVC (Model-View-Controller),επιτρέπει την ανάπτυξη large-scale εφαρμογών μίας σελίδας (large scale ingle-Page Apllications). Η React χρησιμοποιεί μια σύνταξη που ονομάζεται jsx,η οποία είναι ένας συνδιασμός της λειτουργικότητας της javascript ,χρησιμοποιώντας σε πολύ μεγάλο ποσοστό τον τρόπο γραφής της HTML,συνδιάζοντας έτσι δύο τεχνολογίες αρκετά γνώριμες στους developers.'Ετσι,ο κώδικας της React καθίσταται πιο κατανοητός και ευανάγνωστος.
Με το παρακάτω παράδειγμα βλέπουμε τόσο τη χρήση της javascript,όσο και την ομοιότητα στη γραφή της jsx με τη γραφή της HTML,με ορισμένες μικρές διαφορές,όπως το ότι η jsx γράφεται μέσα στο return μιας function, όσο και το ότι δεν χρησιμοποιεί το 'class' όπως html, αλλά το 'className', για το δέσιμο ενός element με μία CSS class.

```
export default function MainMenu() {
    return (
  <>
      <h1 className='Header'>Main Menu</h1>
  <p>Welcome to Main Menu </p>
  </>
    );
  }
```

Σχήμα 2.3.1: Τρόπος γραφής jsx

‘Ενα μεγάλο πλεονέκτημα της React είναι ότι μπορεί να ενσωματώνει διαφορετικά κομμάτια κώδικα μέσα σε ένα έναν πηγαίο κώδικα, καθώς και να ενημερώνει και να αλληλεπιδρά με αυτά μέσα από την ίδια τη σελίδα του πηγαίου κώδικα. Αυτό το επιτυγχάνει με τη δημιουργία components και props. Τα components είναι αυτοσυντηρούμενα κομμάτια κώδικα μίας σελίδας, τα οποία μπορούν να λειτουργούν σαν ξεχωριστά UI elements αυτής, διαθέτοντας,σε ένα μέρος, τη σήμανσή τους, τη λειτουργικότητά τους και την εμφάνισή τους[10]. Τα props αφορούν τα δεδομένα που περνάμε από το ένα component στο άλλο.

Ένα άλλο πλεονέκτημα της React είναι η εικονική DOM (Virtual DOM) που διαθέτει. Η διαχείριση του εικονικού DOM είναι εξαιρετικά γρήγορη και η React φροντίζει να ενημερώνει τον πραγματικό DOM όταν έρθει η κατάλληλη στιγμή. Το κάνει συγκρίνοντας τις αλλαγές μεταξύ του εικονικού DOM και του πραγματικού DOM, προσδιορίζοντας ποιές αλλαγές είναι πραγματικά σημαντικές και πραγματοποιώντας τον ελάχιστο δυνατό αριθμό αλλαγών στον DOM που απαιτούνται για να παραμείνουν όλα ενημερωμένα, σε μια διαδικασία που ονομάζεται **reconciliation** (συμφιλίωση).[11]

Για να επιτελέσει τις λειτουργίες της, η React επιστρατεύει ειδικές συναρτήσεις που ονομάζονται hooks. Το κάθε hook επιτελεί μία ξεχωριστή λειτουργία. Τα σημαντικότερα React hooks είναι:

- **UseState:** Τα components συχνά χρειάζεται να αλλάζουν το περιεχόμενο της οθόνης ως αποτέλεσμα μιας αλληλεπίδρασης. Τα components χρειάζεται να "θυμούνται" πράγματα. Στη React, αυτό το είδος μνήμης που είναι ειδικό για κάθε component ονομάζεται **state** (κατάσταση) [12] . Το useState hook χρησιμοποιείται για ενημέρωση state. Με το useState μπορούμε να επιτύχουμε δυναμικές ενημερώσεις διεπαφής χρήστη **,**διαχείριση εισόδου χρήστη,επικοινωνία μεταξύ των components, ασύγχρονες λειτουργίες και για conditional rendering [13]. Για να ορίσουμε μία μεταβλητή με το useState, αρκεί να χρησιμοποιήσουμε τη γραφή const [<όνομα μεταβλητής>, set<’Ονομα μεταβλητής>] = useState(); . Για να ενημερώσουμε το state, καλούμε το set μέρος της και του περνάμε σαν παράμετρο τη τιμή που θέλουμε να πάρει.

- **UseEffect:** Το useEffect χρησιμοποιείται για διαχείριση side effects (παράπλευρων ενεργειών). Η χρήση των side effects απευθείας μέσα στις συναρτήσεις των React components πρέπει να αποφεύγεται, καθώς αυτό μπορεί να επηρεάσει το rendering του component. Επιπλέον, αν το side effect ενημερώνει μια κατάσταση (state) στη συνάρτηση του component, αυτό θα προκαλέσει εκ νέου rendering του component, με αποτέλεσμα την επανεκτέλεση του side effect. Αυτή η αλληλουχία ενεργειών μπορεί να προκαλέσει έναν ατελείωτο κύκλο rendering. Τα side effects πρέπει να διαχωρίζονται από τη λογική rendering του React και να εκτελούνται σταθερά μόλις το component κάνει render, κάτι που μας προσφέρει το Hook useEffect. Το useEffect είναι μια συνάρτηση που παίρνει δύο ορίσματα. Το πρώτο όρισμα είναι μια συνάρτηση που εκτελεί τη λογική του side effect. Το δεύτερο όρισμα είναι μια προαιρετική λίστα εξαρτήσεων (dependencies) από τις οποίες εξαρτάται η συνάρτηση του side effect. Αυτή η λίστα μπορεί να είναι κενή αν επιλέξουμε να μην αναφέρουμε εξαρτήσεις. Όταν η λίστα είναι κενή, η συνάρτηση useEffect εκτελείται σε κάθε render του component.

- **useCallback:** Δημιουργία callbacks με βελτιστοποίηση απομνημόνευσης (για επαναυπολογισμό των functions όταν αλλάζουν τα dependencies τους)
- **useMemo:** Δημιουργία τιμών με βελτιστοποίηση απομνημόνευσης (για επαναυπολογισμό τιμών σε functions μόνο όταν αλλάζουν τα functions)
- **useRef** :Αποθήκευση μεταβλητών τιμών (για επαναυπολογισμό τιμών που βρίσκονται εκτός κάποιας function) [17]

2.3.2 MongoDB

Η MongoDB είναι ένα σύστημα διαχείρισης βάσεων δεδομένων σχεδιασμένο για την ταχεία ανάπτυξη διαδικτυακών εφαρμογών και υποδομών Διαδικτύου. Το μοντέλο δεδομένων και οι στρατηγικές διατήρησης είναι σχεδιασμένα για υψηλό ρυθμό ανάγνωσης και εγγραφής και για την εύκολη κλιμάκωση με αυτόματη εφεδρεία. Είτε μια εφαρμογή απαιτεί μόνο μία βάση δεδομένων είτε πολλές, η MongoDB μπορεί να προσφέρει εξαιρετικά καλή απόδοση. [18]

2.3.3 NodeJS

Το Node.js είναι γρήγορο και αξιόπιστο για εφαρμογές με βαριά αρχεία και μεγάλο φόρτο δικτύου, λόγω της event-driven, non-blocking και ασύγχρονης προσέγγισής του. Οι προγραμματιστές μπορούν επίσης να διαχειρίζονται ολόκληρα έργα σε μία μόνο σελίδα (SPA) και να το χρησιμοποιούν για το IoT.[19]

2.3.4 ExpressJs

Η πλευρά του διακομιστή, εκτός από το Node js, χρησιμοποιούμε το Express. Το Express ενορχηστρώνει την επεξεργασία δεδομένων, τον έλεγχο ταυτότητας και τη δρομολόγηση, εξασφαλίζοντας γρήγορες και αποδοτικές λειτουργίες[20]. Το Express χρησιμοποιείται για τη διαχείριση των συνεδριών χρηστών, με προαιρετική υποστήριξη από το MongoDB, δημιουργώντας ένα διακομιστή ιστού που ακούει για εισερχόμενα αιτήματα και επιστρέφει σχετικές απαντήσεις, ενώ καθορίζει και μια δομή καταλόγου.[21]

Για να χρησιμοποιήσουμε το Express,αρκεί να δηλώσουμε το express στο server μας (const express = require('express');) , να δημκουργήσουμε μία νέα εφαρμογή express (const app = express();) και να της ορίσουμε το τι θα πρέπει να κάνει ,μέσα στον κώδικα του server.

2.4 Βιβλιοθήκες (Libraries)

Μαζί με τις τεχνολογίες που συνθέτουν το Mern stack, χρησιμοποιήθηκαν βιβλιοθήκες για την ανάπτυξη του Word Wizardry. Οι βιβλιοθήκες είναι έτοιμα κομμάτια κώδικα τα οποία είναι υπεύθυνα για τη διεκπαιρέωση πληθώρα λειτουργιών, όπως το UI της σελίδας, την πλοήγηση από component σε component και την αποστολή και επιστροφή δεδομένων μεταξύ εφαρμογής και βάσης.

Για την εγκατάσταση των βιβλιοθηκών αρκεί να τρέξουμε στην κονσόλα του Visual Studio την εντολή npm install <όνομα βιβλιοθήκης>. Παρακάτω, θα δούμε τις βιβλιοθήκες που χρησιμοποιήθηκαν στη δημιουργία του Word Wizardry.

2.4.1 React Router

Το React Router είναι η καθιερωμένη βιβλιοθήκη για routing στη React .Το routing είναι ο τρόπος για να εισαχθούν λειτουργίες που επιτρέπουν την πλοήγηση στην εφαρμογή μέσω συνδέσμων. Αυτές οι λειτουργίες είναι:

- Ο περιηγητής θα πρέπει να αλλάζει το URL κατά την πλοήγηση σε διαφορετική οθόνη.
- Το deep linking θα πρέπει να λειτουργεί: αν δείξουμε ένα συγκεκριμένο URL στον περιηγητή, η εφαρμογή θα πρέπει να ανασυνθέτει την ίδια όψη που είχε όταν δημιουργήθηκε το URL.
- Τα κουμπιά back και forward του περιηγητή θα πρέπει να λειτουργούν όπως αναμένεται.

Το React Router [22]αναλαμβάνει το ρόλο της διασύνδεσης της πλοήγησης της εφαρμογής με τις λειτουργίες πλοήγησης που προσφέρει ο περιηγητής: τη γραμμή διευθύνσεων και τα κουμπιά πλοήγησης . Τέλος,προσφέρει έναν τρόπο γραφής του κώδικα έτσι ώστε να εμφανίζει συγκεκριμένα components της εφαρμογής μόνο αν η διαδρομή ταιριάζει με αυτό που έχει ορίσειο προγραμματιστής.

Στο Word Wizardry, η βιβλιοθήκη React Router χρησιμοποιήθηκε για περιήγηση από το ένα component στο άλλο. Για τη διασύνδεση αυτών των δύο κομματιών, ορίζονται ειδικά Routes (διαδρομές) τα οποία υποδεικνύουν στο browser πιο component θα χρησιμοποιηθεί και με ποιό url. Ορίζεται το path, το οποίο και αφορά το url που θα χρησιμοποιηθεί από τον browser, αλλά και το element που θα χρησιμοποιηθεί σε αυτό το path, δηλαδή το component που θα τρέξει με την είσοδο σε αυτό το url. Στο παράδειγμα που ακολουθεί αφορά τη μεταφορά από το Main Menu στο component Staurolexo, το οποίο περιέχει το παιχνίδι, ορίζοντας το MainMenu σαν index element της ιστοσελίδας.

```
<BrowserRouter>
  <Routes>
  <Route index element={<MainMenu/>} />
  <Route path="/Staurolexo" element={<Staurolexo />} />
```

Σχήμα 2.4.1: Ενσωμάτωση BrowserRouter και διαδρομών

2.4.2 Jotai

Στον κώδικα πολλές φορές χρειάζεται να χρησιμοποιήσουμε μια μεταβλητή σε διαφορετικά αρχεία,όπου το κάθε αρχείο θα υιοθετεί την τιμή της μεταβλητής απ΄λο ένα άλλο αρχείο και θα την παραλλάσει. Αυτή η διαδικασία μπορεί να γίνει αρκετά επίπονη,καθώς θα πρέπει να λαμβάνουμε συνεχώς υπόψην το data flow (το parent component μοιράζει data στο child component). Το Jotai [23] είναι μια βιβλιοθήκη κατάστασης για το React που βασίζεται στην ιδέα των ατομικών καταστάσεων. Το Jotai επιτρέπει στα στοιχεία να διαχειρίζονται τη δική τους κατάσταση χωρίς την ανάγκη για έναν κεντρικό διαχειριστή.

Το jotai επιτρέπει τη δημιουργία κατάστασης (state) συνδυάζοντας ατομικά στοιχεία και οι αναπαραστάσεις βελτιστοποιούνται αυτόματα με βάση την εξάρτηση από τα ατομικά στοιχεία. Αυτό λύνει το πρόβλημα των επιπλέον επαναπαραστάσεων στο React context, εξαλείφει την ανάγκη για απομνημόνευση, διατηρώντας ένα δηλωτικό μοντέλο προγραμματισμού.Αυτό γίνεται με τη χρήση του hoom ‘useAtom()’[24], το οποίο διαχειρίζεται το state καθολικά στην εφαρμογή, διατηρώντας το τρόπο γραφής του ‘useState’, που είναι ήδη γνώριμος στον προγραμματιστή.

Στην εφαρμογή, το Jotai χρησιμοποιήθηκε για μεταφορά των δεδομένων του χρήστη από τη σελίδα της εγγραφής (Login) στο MainMenu και στο ίδιο το παιχνίδι (Staurolexo).

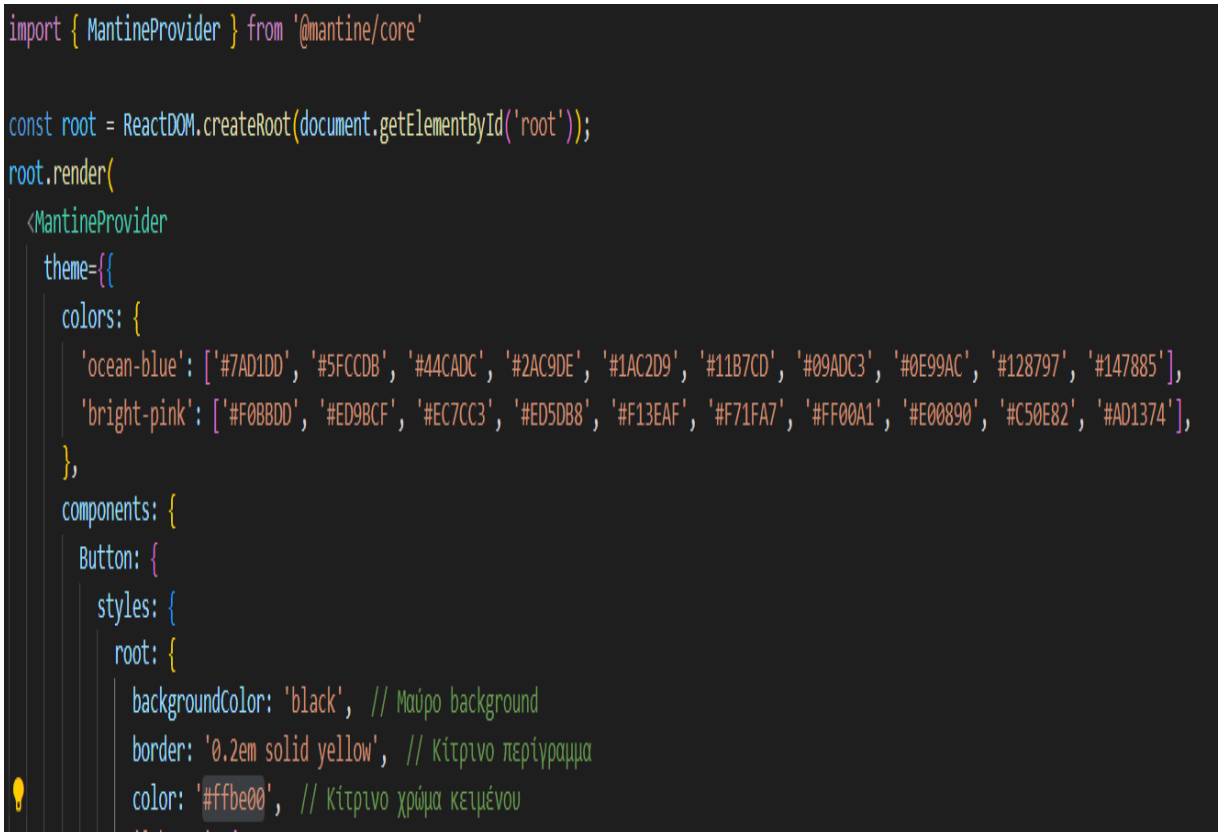
2.4.3 Mantine

Η Mantine είναι μια ελαφριά και εύχρηστη βιβλιοθήκη που παρέχει μια ευρεία γκάμα components και hooks για τη δημιουργία εφαρμογών ιστού υψηλών επιδόσεων. Είναι χτισμένη πάνω από το React και το TypeScript, κάνοντάς την εξαιρετική επιλογή για την ανάπτυξη σύγχρονων εφαρμογών ιστού [25] .

Φυσικά,δεν είναι απαραίτητο με την επιλογή του Mantine να χρησιμοποιηθεί αποκλειστικά αυτό για την ανάπτυξη του Ui της σελίδας, καθώς μπορεί να λειτουργεί δίπλα τόσο στα tags της Html,αλλά και στα Ui components που δημιούργησε ο προγραμματιστής. Επιπροσθέτως, το Mantine προσφέρει server side rendering στα components του,δηλαδή την αποστολή δεδομένων προς τη βάση αλλά και τη λήψη δεδομένων από αυτή [26].

Η βιβλιοθήκη αυτή παρέχει μία ευρεία γκάμα ui components: από απλά buttons και textareas, σε rich editors (inputs για γραφή κειμένου, ενσωμάτωση εικόνων, πινάκων κτλ) και ημερολογίων, παρέχοντας έτοιμα styles, αλλά και τη δυνατότητα στο χρήστη να εφαρμόσει τις δικές του στυλιστικές επιλογές στα components του, χρησιμοποιώντας CSS εντός των Mantine components [27].

Το Mantine λειτουργεί σαν κέλυφος στην εφαρμογή (<App>). Το μόνο που χρειάζεται για τη χρήση του είναι η εισαγωγή του MantineProvider στο αρχείο index.js της εφαρμογής. ‘Επειτα, για να χρησιμοποιήσει ο programmer ένα Ui component της αρεσκείας του σε ένα αρχείο, το μόνο που χρειάζεται να κάνει είναι import το Ui component του Mantine στο αρχείο.



Σχήμα 2.4.3: Εισαγωγή Mantine στο index και ορισμός θεμάτων

2.4.4 Nodemon

Στη διαδικασία προγραμματισμού στο Node.js, χρειάζεται να επανεκκινήσουμε τη διαδικασία που τρέχει, για να εφαρμοστούν οι αλλαγές. Αυτό προσθέτει ένα επιπλέον βήμα στη ροή εργασίας. Για την εξάλειψη αυτού του επιπλέον βήματος, χρησιμοποιήσαμε το nodemon, μια library ειδική στην αυτόματη επανεκκίνηση της διαδικασίας, όταν παρατηρούνται αλλαγές στον κώδικα[28].

Το nodemon αρχικά γράφτηκε για να επανεκκινεί διεργασίες που έχουν "κολλήσει", όπως διακομιστές ιστού, αλλά τώρα υποστηρίζει και εφαρμογές που τερματίζουν ομαλά. Αν το σενάριο τερματίζει ομαλά, το nodemon συνεχίζει να παρακολουθεί το φάκελο (ή τους φακέλους) και θα επανεκκίσει το σενάριο εάν υπάρχουν αλλαγές [29].

Για τη χρήση του nodemon, αρκεί να πληκτρολογήσουμε στην κονσόλα του Visual Studio την εντολή npx nodemon <όνομα αρχείου> [30].

Το nodemon βοήθησε στην ανάπτυξη του app, μιάς και χρησιμοποιήθηκε στην αποθήκευση των λέξεων στη βάση, επανεκκινώνυτας τη διαδικασία αποθήκευσης όταν αυτή είχε κολλήσει.

2.4.5 Axios

Η Axios.js ή απλώς Axios είναι μια πολύ γνωστή βιβλιοθήκη JavaScript που μπορεί να χρησιμοποιηθεί για την εκτέλεση HTTP αιτήσεων, λειτουργώντας τόσο σε περιβάλλον Node.js όσο και σε φυλλομετρητές (browsers)[31].

Η Axios απλοποιεί τη διαδικασία αποστολής HTTP αιτήσεων προσφέροντας μια καθαρή και διαισθητική API. Είναι βασισμένη σε υποσχέσεις (promises), επιτρέποντας στους προγραμματιστές να δημιουργούν ασύγχρονο κώδικα χρησιμοποιώντας τη σύνταξη async/await ή τη σύνταξη .then() [32]

Με τη βιβλιοθήκη Axios μπορούμε να διασυνδέσουμε την εφαρμογή Word Wizardry με τη βάση MongoDb και να αποστέλουμε και να λαμβάνουμε δεδομένα τόσο για το χρήστη,όσο και για τις ενέργειές του.

2.4.6 Capacitor

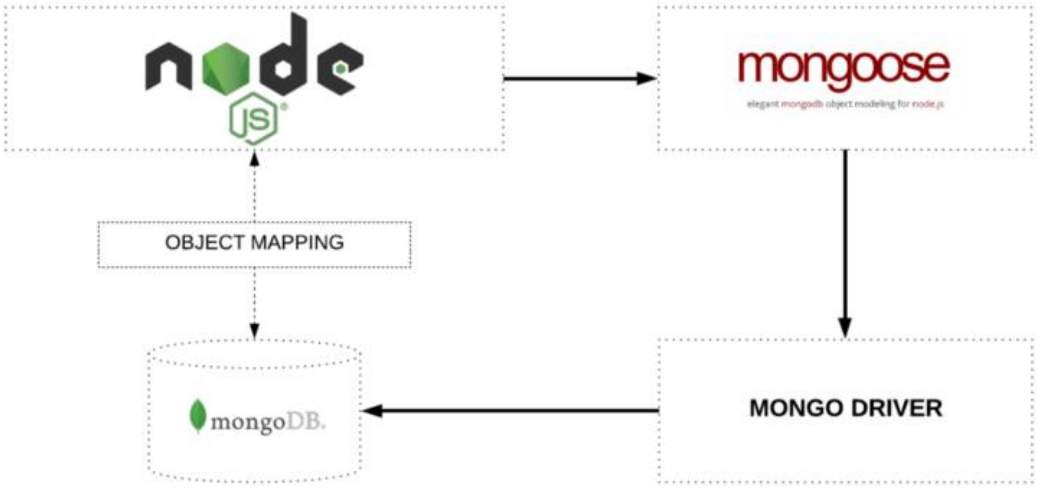
Το app Word Wizardry πρέπει να είναι διαθέσιμο εκτός από υπολογιστές και σε smartphone που χρησιμοποιούν το λειτουργικό σύστημα android. Για τη δημιουργία ενός mobile app δεν αρκεί η React, αλλά χρειάζεται και κάποια βιβλιοθήκη που να ενσωματώνει τις λειτουργίες των κινητών συσκευών στον κώδικα. Η βιβλιοθήκη που χρησιμοποιήθηκε για τη μετάφραση της εφαρμογής ώστε να λειτουργεί σε android, είναι το Capacitor της Ionic Framework.

Το Capacitor είναι ένα εργαλείο ανοικτού κώδικα για τη δημιουργία εγγενών εφαρμογών ιστού. Με το capacitor καθίσταται δυνατή η δημιουργία διαλειτουργικών εφαρμογών iOS, Android και Progressive Web Apps με JavaScript, HTML και CSS [33].Για το cross-platforming, χρησιμοποιείται μία ενιαία βάση κώδικα, συντελώντας με αυτό το τρόπο στη μείωση των κόστων ανάπτυξης και σε έναν αποδοτικότερο προγραμματισμό της εφαρμογής[34].

2.4.7 Mongoose

Η Μοντελοποίηση Δεδομένων Αντικειμένων (ODM) στην ανάπτυξη λογισμικού επικεντρώνεται στην αφαίρεση και δομήση δεδομένων με βάση τις αντιστοιχίσεις τους στον πραγματικό κόσμο. Αντίθετα από τις παραδοσιακές σχεσιακές βάσεις δεδομένων όπως η MySQL ή η PostgreSQL, οι ODMs επιτρέπουν την αντιμετώπιση των δεδομένων ως πραγματικά αντικείμενα μέσα στον κώδικα. Η Μοντελοποίηση Δεδομένων Αντικειμένων (ODM) παρέχει μείωση της ασυμφωνίας τάσης μεταξύ της βάσης δεδομένων και των γλωσσών αντικειμενοστραφούς προγραμματισμού, μεγάλη ευελιξία στη διαχείριση πολύπλοκων δομών δεδομένων καθώς και αποτελεσματικότερα και βελτιστοποιημένα ως προς την απόδοση μοτίβα πρόσβασης στα δεδομένα,χωρίς την προυπόθεση της προχωρημένης γνώσης και εμπειρίας πάνω στην SQL[35].

Μία από τις πιό δημοφιλείς βιβλιοθήκες ειδικές στη Μοντελοποίηση Δεδομένων Αντικειμένων στο Node js και στη Mongo Db είναι το Mongoose.



Σχήμα 2.4.7: Επικοινωνία Mongoose με Node και MongoDB[36]

Το Mongoose επιτρέπει την οργάνωση των δεδομένων που θέλουμε να αποστείλουμε στη βάση με τη χρήση σχημάτων (schemas) [37], που είναι απλές και κατανοητές δομές για το structuring των δεδομένων. Τα προτερήματα του Mongoose είναι:

- Η ειδίκευσή του στο χειρισμό structured data (δομημένων δεδομένων)
- Αφαιρεί την πολυπλοκότητα των εμφωλευμένων πλαισίων στη βάση (nested callbacks) μέσω της απλής και ευκατανόητης δόμησής του
- Η επιστροφή των δεδομένων σαν JSON objects, τα οποία μπορούν εύκολα να χρησιμοποιηθούν από τον κώδικα
- Η μεγάλη ποικιλία πρόσθετων functions και μεθόδων που διαθέτει, βοηθώντας τους developers να διαχειριστούν τα δεδομένα της βάσης αναλόγως των περιορισμών και των απαιτήσεων του κώδικα [38]

2.5 Εφαρμογές που χρησιμοποιήθηκαν

2.5.1 Canva

Για τη δημιουργία των γραφικών του τίτλου της εφαρμογής δεν στηριχτήκαμε στον κώδικα, αλλά επιστρατεύσαμε μία εφαρμογή που ονομάζεται Canva. Το Canva είναι ένα διαισθητικό, διαδικτυακό εργαλείο σχεδίασης, το οποίο μπορεί να χρησιμοποιηθεί για τη δημιουργία διαφόρων εικόνων και εγγράφων[39].
Διαθέτει γραμματοσειρές, γραφικά, διανύσματα και πρότυπα, φίλτρα φωτογραφιών, εικονίδια και σχήματα, γραμματοσειρές καθώς και χιλιάδες έτοιμα templates. Μόλις ολοκληρωθεί ο σχεδιασμός, μπορεί να ληφθεί σε διάφορες μορφές, όπως μορφή αρχείου εικόνας JPEG, φορητό δίκτυο γραφικών (PNG) και φορητή μορφή εγγράφου (PDF) [40].

Κεφάλαιο 3ο: Ανάκτηση των λέξεων και εναπόθεσή τους στη βάση

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τη διαδικασία ανάκτησης των λέξεων που θα χρησιμοποιηθούν στο σταυρόλεξο και την εναπόθεσή τους στη βάση MongoDB. Για την οργάνωση και αποθήκευση των λέξεων στη database στη Mongo Db,όπως και σε άλλες μη-σχεσιακές βάσεις,η συνειστώμενη μέθοδος είναι η δημιουργία ενός "schema".Το "schema" περιγράφει τη δομή των δεδομένων που θα αποθηκευτούν στη βάση.Πρόκειται δηλαδή για το "ραχοκοκαλία" της δομής,μιας και περιλαμβάνει αναπαραστάσεις πεδίων,τύπων δεδομένων,κανόνων εγκυρότητας και λοιπών πληροφοριών για τα δεδομένα που πρόκειται να αποθηκευτούν στη βάση. Η λειτουργικότητα του "schema" αποθηκεύεται σε ένα μοντέλο του Mongoose,το οποίο χρησιμοποιείται για τη δημιουργία,την ανάγνωση, την ενημέρωση και τη διαγραφή δεδομένων στο πλαίσιο της collection (της συλλογής δεδομένων που αποθηκεύουμε) στη Mongo db.

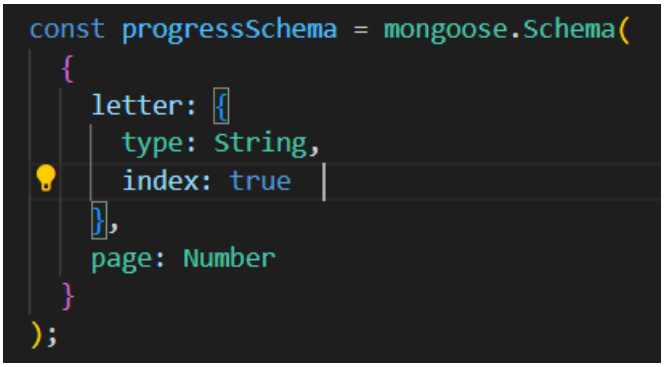
3.1.1 Progresses

Λόγω του μεγάλου όγκου δεδομένων των λέξεων, για τη διασφάλιση της σωστής αποίκισης της συλλογής των λέξεων στη βάση,κρίθηκε ότι μία σωστή και ασφαλής μέθοδος θα ήταν η δημιουργία μιας δεύτερης συλλογής, η οποία θα αφορά την κατάσταση της αποίκοισης των λέξεων, με απλά λόγια του σημείου του upload των λέξεων που βρισκόμαστε ανά πάσα στιγμή .

Η συλλογή progresses είναι αρμόδια για την επαναφορά της διαδικασίας ανεβάσματος των λέξεων στη συλλογή Words σε περίπτωση που προκύψει κάποιο error. Κάθε φορά που θα ανεβαίνει μία λέξη στη βάση,θα αποθηκεύεται η σελίδα που περιέχει τη λέξη στη συλλογή progresses. Αν προκύψουν προβλήματα στη διαδικασία ανεβάσματος λέξεων,όπως αποσύνδεση από τη βάση, χάσιμο της σύνδεσης δικτύου ή υπερφόρτωση από πληροφορία σε μικρό διάστημα της βάσης, η διαδικασία δεν τερματίζει ή αρχίζει ξανά απ την αρχή. Αντιθέτως, η διαδικασία ανεβάσματος περικείεται από την ενημέρωση του progresses · ο κώδικας επικοινωνεί με το progresses, επιστρέφει τη τελευταία σελίδα λέξεων που έχει αποθηκευτεί και αρχίζει να αποθηκεύει ξανά από αυτή τη σελίδα.

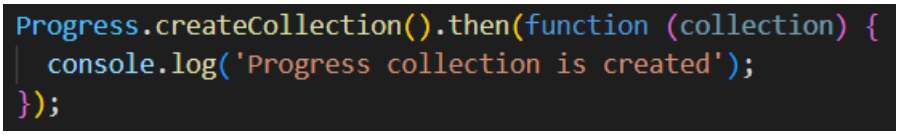
'Ετσι επιτυγχάνεται μια καλύτερη οργάνωση του κώδικα,όσο και η ανάπτυξη μιας αδιάβλητης δομής, η οποία έχει προνοήσει για τα λάθη που μπορούν να προκύψουν και έχει προσαρμοστεί ώστε να μην επηρεάζεται η εύρυθμη λειτιουργία του ανεβάσματος των λέξεων παρά τα προβήματα που δύναται να παρουσιαστούν.

Πρωτογενώς για την ανάπτυξη του collection των Progresses, πρέπει να δημιουργηθεί ένα σχήμα. Αυτό το σχήμα ορίζει ότι αποθηκεύονται τόσο το γράμμα από το οποίο ξεκινάει η δεκάδα των λέξεων που βρίσκονται στη διαδικασία αποθήκευσης, όσο και τη σελίδα.

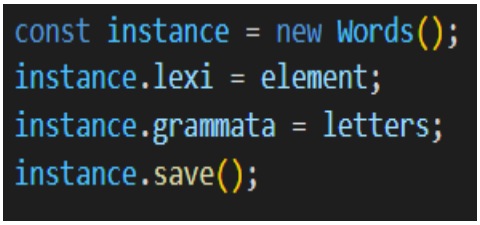


Σχήμα 3.1.1.1: Ορισμός schema

Αμέσως μετά δημιουργείται το collection Progress,το οποίο θα αποθηκεύει τις πληροφορίες που έχουμε ορίσει για αποθήκευση με βάση το schema progressSchema: το letter (γράμμα με το οποίο αρχίζουν οι λέξεις) και το page (η σελίδα των λέξεων που έχει αποθηκευτεί, εκ' των οποίων σε περίοτωση λάθους θα ξαναρχίζει η διαδικασία του ανεβάσματος λέξεων από την τελευταία αποθηκευμένη σελίδα).



Σχήμα 3.1.1.1: Δημιουργία collection για τα Progresses



Σχήμα 3.1.1.1: Δημιουργία instances για τα πεδία lexi και grammata στο collection

Μέσα στην επαναληπτική μέθοδο οικοδόμισης της βάσης με τις λέξεις μέσω του τελευταίου progress που αποθηκεύτηκε

```
catch (err) {
  if (err)
    {
    console.log(`ECONNRESET error at letter ${letter}, page ${i}. Reconnecting...`);
    await mongoose.disconnect();
    await new Promise(resolve => setTimeout(resolve, 5000));
    await mongoose.connect('mongodb+srv://stevekoulas:asfalisa1@staurolexo.zjs7exc.mongodb.n
      , { connectTimeoutMS: 120000000 });

    startLetter = letter;
    startPage = i;
    i--;
  } else {
    console.log(err.message);
  }
}
```

Σχήμα 3.1.1.4: Επανασύνδεση της διαδικασίας αποίκισης των λέξεων στη βάση με τα τελευταία στοιχεία

3.1.2 Μοντέλο,σχήμα και τρόπος αποθήκευσης των λέξεων

Παρακάτω είναι ο τρόπος δημιουργίας ενός “schema” για την περιγραφή των εγγράφων που θα χρησιμοποιηθούν στη Mongo Db:

```
const wordSchema = new mongoose.Schema({
  lexi: {
    type: String,
    required: false
  },
  grammata: {
    type: Array,
    required: false
  }
});
```

Σχήμα 3.1.2.1: Ορισμός schema για τις λέξεις

\grammata και lexi,καθώς αυτά θα είναι τα πεδία που θα συγκρίνονται με το συνδιασμό γραμμάτων που δημιουργείται,ώστε να επιστρέψουν τις λέξεις που επιβεβαιώνουν τους κανόνες που θέσαμε. Ορίζεται έτσι ότι,λοιπά πεδία (όπως για παράδειγμα το πεδίο CreatedAt που δείχνει πότε έγινε η εγγραφή της λέξης στη βάση) δεν χρειάζονται να μπαίνουν σαν εμπόδιο στην εύρεση των στοιχείων που θέλουμε. Με τα ευρετήρια, επιταχύνονται η αναζήτηση και ταξινόμηση των εγγράφων στη συλλογή, καθώς η MongoDB μπορεί να εντοπίσει τα κατάλληλα έγγραφα πολύ πιο αποτελεσματικά.

```
wordSchema.index({ grammata: 1 });
wordSchema.index({ lexi: 1 });
```

Σχήμα 3.1.2.2: Indexes στα πεδία lexi και grammata

‘Ετσι,δημιουργήθηκαν δύο ευρετήρια με αύξουσα σειρά,τόσο για τα strings των λέξεων,όσο και για τον array των γραμμάτων που περιέχει κάθε λέξη.

Κατόπιν,το schema ‘wordSchema’ αποθηκεύτηκε σε ένα μοντέλο του mongoose με όνομα ‘Word’ με σκοπό τη δημιουργία της collection για CRUD Operations στη βάση με τον κάτωθι τρόπο:

```
const Word = mongoose.model('Word', wordSchema);
```

Σχήμα 3.1.2.3: Ορισμός μοντέλου το Mongoose πάνω στο schema των λέξεων

‘Επειτα, δημιουργήθηκε το collection Word στη βάση αυτό το σχήμα και έγινε export το αρχείο με σκοπό τη χρήση του,συγκεκριμένα την ενημέρωση και παραμετροποίησης της βάσης:

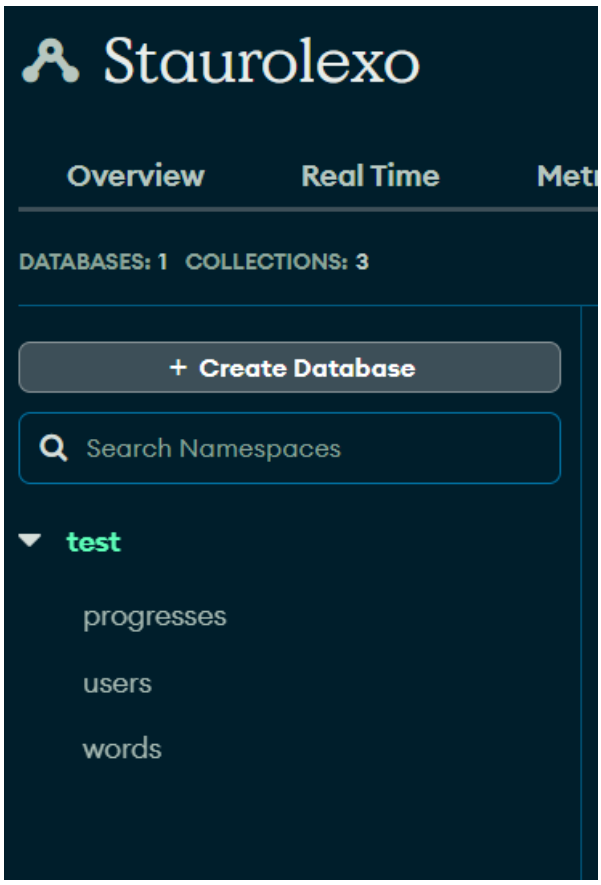
```
Word.createCollection().then(function (collection) {
    console.log('collection is created');
})

module.exports = Word;
```

Σχήμα 3.1.2.4: Δημιουργία collection για τις λέξεις βασισμένο στο μοντέλο του Mongoose

3.2 Collection λέξεων στη Mongo Db

Η database που έχει δημιουργηθεί, καθώς και τα στοιχεία αυτής,όπως οι collection και οι εγγραφές των αρχείων, είναι προσβάσιμες και από το γραφικό περιβάλλον της MongoDb. Το γραφικό περιβάλλον της Mongo είναι φιλικό προς το χρήστη,αφού παρέχει σημαντικές πληροφορίες πάνω στα δεδομένα του χρήστη,αλλά και πληθώρα άλλων εργαλείων που μπορούν να φανούν χρήσιμα στην ανάπτυξη λογισμικού. Μέσα από το ui,μπορούμε να ελέγξουμε αν οι πληροφορίες που έχουμε αποθηκεύσει έχουν όντυως αποθηκευτεί στη μορφή και με το τρόπο που θέλουμε,χωρίς την αλληλεπίδραση με κώδικα. Επίσης,μπορούμε να διακρίνουμε τυχόν λάθη που προέκυψαν κατά την οικοδόμιση της βάσης και την επίκοισή της με τα αρχεία της επιλογής μας, φερειπείν να μην έχουν προστεθεί τα ευρετήρια (indexes) που ορίσαμε, οι collection να είναι κενές κτλ.

Στο στάδιο που βρισκόμαστε, η βάση πρέπει να είναι κάπως έτσι:



Σχήμα 3.2.1: UI MongoDB με τη database και τα collections

Παρατηρούμε ότι στο περιβάλλον της MongoDb έχει δημιουργηθεί τόσο η βάση Staurolexo,όσο και οι collection που έχουμε ορίσει: η collection users η οποία διατηρεί τα στοιχεία των χρηστών που έχουν εγγραφεί στην εφαρμογή, η collection words την οποία συνθέτουν οι λέξεις του ελληνικού λεξικού που είναι διαθέσιμες για χρήση στην εφαρμογή για τη δημιουργία των σταυρολέξων, καθώς και η collection progresses, η οποία χρησιμοποιήθηκε για την διατήρηση του σημείου επίκοισης της βάσης με λέξεις,ώστε σε περίπτωση σφάλματος να μην χρειαστεί να γίνει η διαδικασία ξανά από την αρχή. Η collection αυτή, αν και δεν χρησιμεύει σε κάποιο στάδιο πέρα από το χτίσιμο της βάσης, καθώς δεν είναι μια συλλογή που θα ενημερώνεται με άξονα το user input ή μια συλλογή που

χρειάζεται κατά το τρέξιμο της εφαρμογής, παραμένει ενεργή και διαθέσιμη ώστε, σε περίπτωση που θεωρηθεί χρήσιμη η ενημέρωση της βάσης στο μέλλον, να μπορεί να απλουστεύσει αυτή τη διαδικασία, τόσο γιατί δεν θα χρειαστεί εκ νέου ανάπτυξη κώδικα, όσο και γιατί θα μειώση το χρονικό διάστημα στο οποίο οι χρήστες αποτρέπονται να εισέλθουν στην εφαρμογή καθώς αυτή ενημερώνεται.

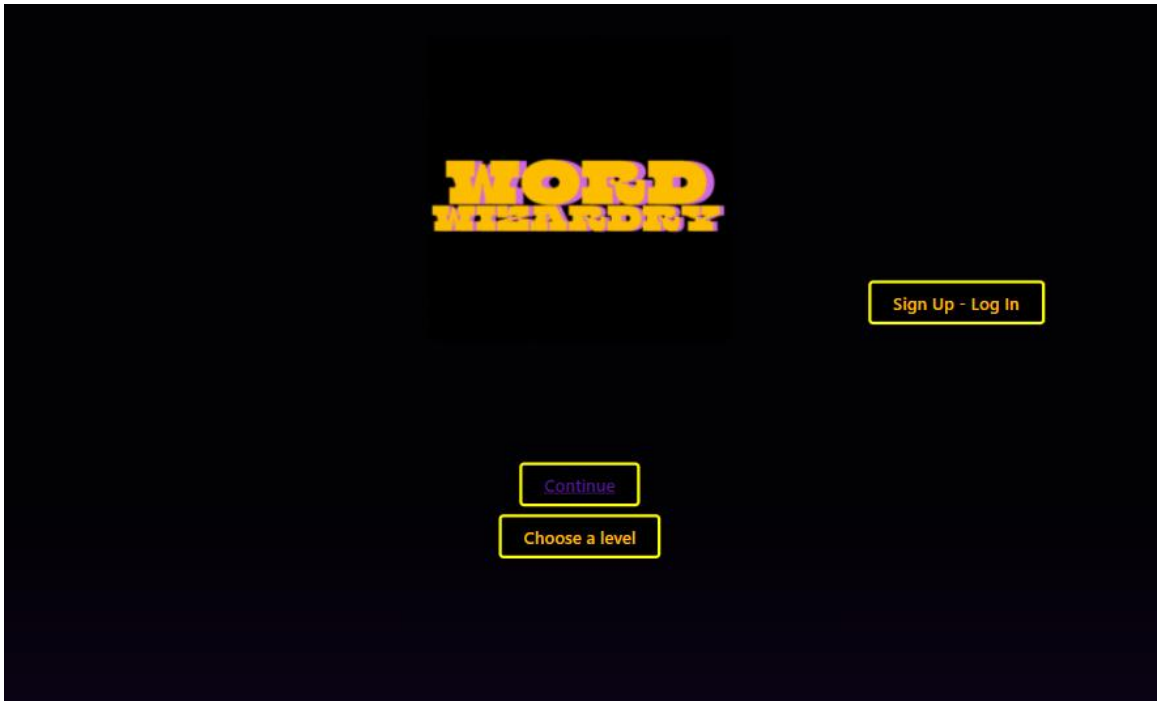
Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
progresses	17859	1.02MB	60B	480KB	2	1.04MB	532KB
users	4	729B	183B	36KB	2	72KB	36KB
words	178396	36.81MB	217B	9.25MB	3	34.29MB	11.43MB

Σχήμα 3.2.2: UI MongoDB με τα collections και πληροφορίες αυτών

Επιπροσθέτως,μπορούμε να δούμε και πληροφορίες για τα αρχεία που συνθέτουν τις συλλογές. Αυτά τα πεδία έπαιξαν σημαντικό ρόλο στη φάση κατασκευής της εφαρμογής, αλλά μπορούν να χρησιμοποιηθούν και για την επίβλεψη αυτής και τη διασφάλιση της ομαλής της λειτουργίας. Το πεδίο Documents για παράδειγμα,απεικονίζει το σύνολο των αρχείων που έχουν οι collections. ‘Ετσι, κατά την οικοδόμιση της εφαρμογής έγινε δυνατή η παρατήρηση του ανεβάσματος των λέξεων,αλλά και κατά τη χρήση της εφαρμογής η επισκόπηση των users. Το πεδίο Indexes επιτρέπει την εξέταση των ευρετηρίων που προσθέσαμε στον κώδικα για την επιστροφή πληροφοριών,καθώς και τη δημιουργλια νέων ευρετηρίων σε περιπτωση που κάτι τέτοιο θυεωρηθεί σημαντικό. Τέλος, αξίζει να αναφερθεί ότι τα πεδία που αφορούν τα sizes,με απλά λόγια τα αρχεία που αφορούν τα μεγέθη των εγγραφών,έχουν δημαντικό ρόλο στην ανάπτθυξη του κώδικα, καθώς αυτός αναπτύχθηκε βάσει των απαιτήσεων των αρχείων που διαχειρίζεται.

4 Main Menu

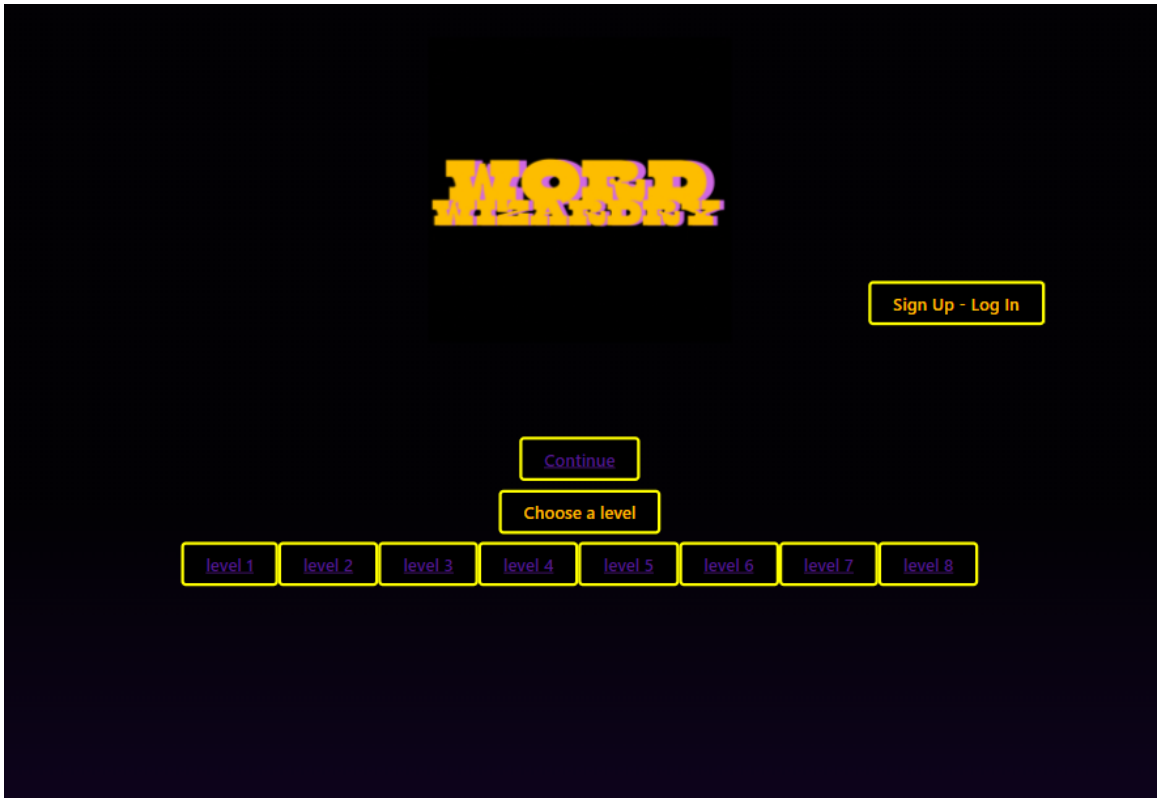
Το Main Menu αποτελεί τη main page της εφαρμογής, το οποίο σημαίνει πως πρόκειται για τη σελίδα στην οποία θα βρεθεί ο χρήστης όταν ανοίγει την εφαρμογή. Το Main menu αποτελείται από 2 διαφορετικά στοιχεία. ‘Ενα στοιχείο για το login-signup του χρήστη και ένα στοιχείο για την επιλογή επιπέδου.



Σχήμα 4.1: Main Menu

Στο πλαίσιο της δομής και της στοίχησης της σελίδας, πάνω πάνω εμφανίζεται ο τίτλος του παιχνιδιού “Word Wizardry”. Έπειτα, κάτω από το τίτλο και στα δεξιά της σελίδας εγκαταστάθηκε το component για το sign Up και το Log In. Τέλος, στο μέσο της σελίδας του Main Menu υπάρχουν τα κουμπιά για την είσοδο στο παιχίδι : το κουμπί για το continue, τη συνέχιση του παιχνιδιού από εκεί που είχε μείνει ο χρήστης, αλλά και το κουμπί “Choose a Level”, με το οποίο ο χρήστης μπορεί να επιλέξει το επίπεδο της αρεσκείας του.

Ο τίτλος “Word Wizardry” πρόκειται για ένα gif. Ο χρήστης έχει τη δυνατότητα να δει τους συντελεστές της δημιουργίας της εφαρμογής είτε σέρνοντας τον κέρσορα του ποντικιού πάνω από το τίτλο, για λειτουργικό σύστημα windows, είτε κάνοντας κλικ πάνω στο τίτλο, για κινητές συσκευές.



Σχήμα 4.2: Main Menu με ανοιγμένη την επιλογή επιπέδου

4.1 CSS του Main Menu

Το background της σελίδας, αλλά και όλης της εφαρμογής,πραγματοποιήκε με τη χρήση linear gradient της css. Τα linear gradient ή γραμμικές διαβαθμίσεις, είναι διαβαθμίσεις χρώματος που προχωρούν κατά μήκος ενός γραμμικού διανύσματος [41]. Η σελίδα αισθητικά αποτελείται από ένα μαύρο πλαίσιο το οποίο,όσο κατεβαίνουμε στον άξονα y, μετατρέπεται σε μωβ.

Τα στοιχεία της σελίδας στοιχήθηκαν το ένα κάτω απ το άλλο(flex-direction: column;), στο κέντρο της σελίδας (‘justify-content: center;’), επιβάλλοντας την ιδιότητα ‘display: flex;’, που είναι η ιδιότητα της CSS αρμόδια για τη σύνταξη του layout.Στο ‘Sign Up – Log In’ button, για να αποφευχθεί η στοίχιση του στο κέντρο της σελίδας και σαν column, δόθηκε έξτρα κώδικας CSS. Συγκεκριμένα, περικλήθηκε σε ένα div, στο οποίο δόθηκε ένα standard position ούτως ώστε να μην αλλάζει θέση στη σελίδα ανάλογα με τις αλληλεπιδράσεις του χρήστη με αυτήν (‘position: fixed;’) αλλά και συγκεκριμένες συνεταγμένες ως προς τη θέση του στον άξονα X και Y (‘top: '30%', right: '10%'). Για το responsiveness του στοιχείου ‘Sign Up – Log In’,δηλαδή για την εμφάνισή του σε διαφορετικές διαστάσεις, προστέθηκε το ‘transform: 'translateY(-50%)';’. Η εντολή transform …..Στη συγκεκριμένη περίπτωση, ορίστηκε ότι, όσο παραλλάσεται η διάσταση, τόσο θα μεταφέρεται το button στον άξονα y.

Για τη μορφοποίηση των buttons έγινε καθολική μορφοποίηση, εφαρμόζοντας θέμα. Καθολική μορφοποίηση ουσιαστικά σημαίνει μορφοποίηση από τη ‘ρίζα’ της εφαρμογής, ακριβέστερα το

αρχείο index, στο πεδίο root. Μέσα στο index.js ορίστηκε ότι τα Buttons του Mantine θα ακολουθούν όλα ένα συγκεκριμένο θέμα, ούτως ώστε να επιτευχθεί η αισθητική ομαλότητα σε όλες τις διαφορετικές σελίδες της εφαρμογής, αλλά και για μείωση του κώδικα που πρέπει να γραφτεί, καθώς δεν θα χρειάζεται σε κάθε διαφορετικό αρχείο css να ορίζονται οι ιδιότητες των Buttons. Το θέμα που επιλέχτηκε είναι: ένα κίτρινο πλαίσιο γύρω από τα Buttons πάχουν 0,2em (border: '0.2em solid yellow'), ένα εσωτερικό μαύρο χρώμα (backgroundColor: 'black') και ο τίτλος του κάθε Button σε μία κίτρινη-χρυσαφί απόχρωση (color: '#ffbe00').

4.1.2 Τίτλος

‘Οπως προσδιορίστηκε, ο τίτλος πρόκειται για ένα gif ονόματι ‘wordwizardry.gif’ που δείχνει το όνομα της εφαρμογής ‘Word Wizardry’, αλλά μεταβάλλεται όταν ο χρήστης πάει πάνω από αυτό, εμφανίζοντας τα ονόματα των συντελεστών σε ένα άλλο gif, το ‘written.gif’. Αυτό επιτεύχθηκε μέσω δύο function που ανιχνεύουν τις κινήσεις του χρήστη ‘onmouseover’, ήτοι η function της JavaScript που ενεργοποιείται όταν ο χρήστης τοποθετεί τον δείκτη του ποντικιού πάνω από ένα στοιχείο, και τη function ‘onmouseleave’ ενεργοποιείται όταν ο δείκτης του ποντικιού απομακρύνεται από το στοιχείο, επαναφέροντας την προηγούμενη κατάσταση.

```
const [hovered, setHovered] = useState(false);

const handleMouseOver = () => {
  setHovered(true);
};
const handleMouseLeave = () => {
  setHovered(false);
};
```

Σχήμα 4.1.2.1: Functions handleMouseOver, handleMouseLeave

Η function handleMouseOver καλείται όταν ο δείκτης του ποντικιού τοποθετείται πάνω από το στοιχείο στο οποίο έχει συνδεθεί το event onMouseOver. Στην προκειμένη περίπτωση, αυτή η function αλλάζει την κατάσταση hovered σε true, η οποία με τη σειρά της ενεργοποιεί την εμφάνιση μιας διαφορετικής εικόνας (συγκεκριμένα, η εικόνα αλλάζει από ‘wordwizardry.gif’ σε ‘written.gif’).

Η function handleMouseLeave καλείται όταν ο δείκτης του ποντικιού απομακρύνεται από το στοιχείο. Αυτή η function αλλάζει την κατάσταση hovered σε false, επαναφέροντας έτσι την αρχική εικόνα (wordwizardry.gif).

```
<h1
  style={{ position: 'fixed', top: '0%', left: '50%', transform: 'translateX(-50%)', zIndex: 100 }}
  onMouseOver={handleMouseOver}
  onMouseLeave={handleMouseLeave}
>
<img
  style={{ width: '6.2em', height: '6.2em' }}
  src={hovered ? require('./written.gif') : require('./wordwizardry.gif')}
  alt={hovered ? "written and directed by Kokkwnis Georgios, Stauros Koulas" : "Word Wizardry"}
/>
```

Σχήμα 4.1.2.2: Αλλαγή εικόνας ανάλογα με τις ενέργειες του χρήστη

‘Ετσι,το logo παραλλάσεται από Unhovered Logo, κοινώς το logo στο οποίο δεν έχουμε σείρει τον κέρσορα του ποντικιού από πάνω, τουτέστιν το ‘wordwizardry.gif’ ,σε Hovered Logo, που είναι το ‘written.gif’.

Σχήμα 4.1.2.3: Unhovered Logo

Σχήμα 4.1.2.4: Hovered Logo

4.2 Επιλογή επιπέδου

Για την επιλογή επιπέδου υπάρχει η επιλογή continue (η οποία παρέχει στους users τη δυνατότητα να συνεχίσουν από εκεί που βρίσκονται) καθώς και ένα choose level button,το οποίο επιτρέπει στο χρήστη την επιλογή ενός επιπέδου. Μόλις ο χρήστης πατήσει το choose level,εμφανίζονται τα επίπεδα τα οποία μπορεί να επιλέξει. Ο τρόπος λειτουργίας αυτής της δυνατότητας είναι ο εξής: έχει δημιουργηθεί μια μεταβλητή χρησιμοποιώντας το useState hook,ονόματι showLevelMenu,η οποία αρχικά έχει την τιμή false.

```
const [showLevelMenu,setShowLevelMenu] = useState(false);
```

Σχήμα 4.2.1: Αρχικοποίηση μεταβλητής για την εμφάνιση μενού με useState hook

Μόλις πατηθεί το button για την επιλογή επιπέδου,αυτή η μεταβλητή αλλάζει τιμή μέσω του showLevelHandler και παίρνει την αντίθετη τιμή από αυτή που έχει ήδη. Δηλαδή,αν το showLevelMenu έχει τιμή false(δεν εμφανίζονται τα levels), θα πάρει τη τιμή true που επιτρέπει την εμφάνιση των levels και το αντίστροφο.

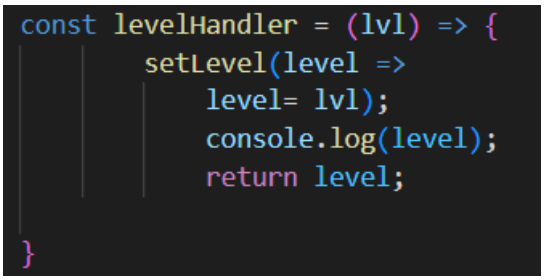
```
const showLevelHandler = () => {
    setShowLevelMenu(!showLevelMenu);
}
```

Σχήμα 4.2.2: Handler για επιλογή επιπέδου

Αυτό επιτρέπει την εμφάνιση των επιλογών των επιπέδων στο χρήστη. Αυτές οι επιλογές είναι διαφορετικά buttons με τιμή το επίπεδο. Αν η τιμή του showLevelMenu,εμφανίζονται τα buttons,τα οποία πατώντας τα ο χρήστης,οδηγείται στο κάθε επίπεδο. Το showLevelMenu === true ? ‘’ : ‘’ πρόκειται για έναν εναλλακτικό τρόπο χρήσης της if-else,καθώς αν ισχύει η συνθήκη που θέσαμε (showLevelMenu === true) τρέχει ο κωδικας που έπεται του ερωτηματικού,αλλιώς αν η συνθήκη που θέσαμε δεν ισχύει (αν showLevelMenu === false),τρέχει ο κώδικας μετά την άνω και κάτω τελεία.

Σχήμα 4.2.3: Διαφορετικά επίπεδα

Πατώντας ένα επίπεδο,καλείται μία function που ονομάζεται levelHandler. Αυτή η function χρησιμοποιείται για την αποθήκευση και «μετάδοση» του level, του επιπέδου με άλλα λόγια που επέλεξε ο κώδικας,στη συνέχεια της εφαρμογής προς χρήση από τα διαφορετικά components και functions.

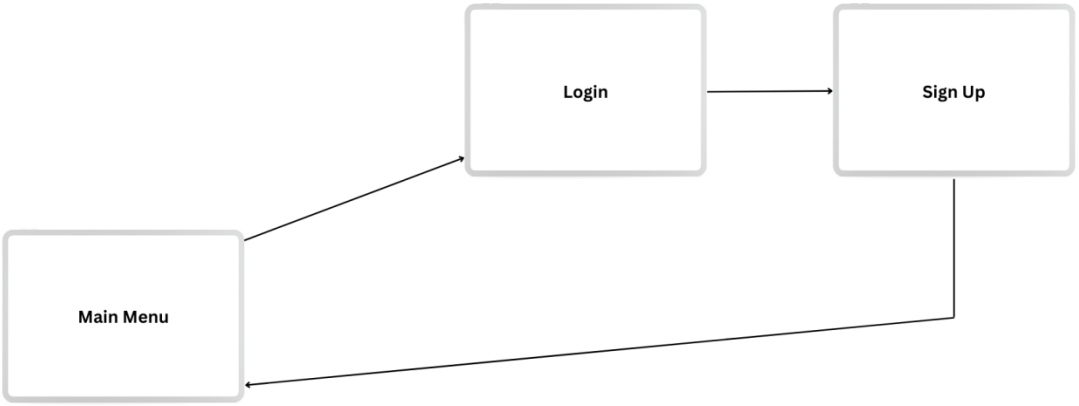


Σχήμα 4.2.4: Function για επιλογή επιπέδου

Τέλος,το κάθε button πατώντας το λειτουργεί ως ένα link για το component Staurolexo. Έτσι,οδηγούμαστε στο component Staurolexo,στο κομμάτι της σελίδας που αφορά το παιχνίδι,κρατώντας όμως την τιμή του επιπέδου, το οποίο και θα μας επιτρέψει την δημιουργία γραμμάτων και κατ' επέκταση,την επιστροφή των λέξεων και τη δημιουργία του επιπέδου.

4.3 Sign Up και Login χρήστη

Οποιοσδήποτε χρησιμοποιεί την εφαρμογή μπορεί να παίξει το παιχνίδι χωρίς να είναι εγγεγραμμένος. Αν όμως είναι εγγεγραμμένος, μπορεί να συλλέξει πόντους και να συνεχίσει την πορεία του από εκεί που έχει μείνει, είτε στην ίδια συσκευή είτε σε άλλη,επιτρέποντας το cross-platform communication. Με αυτή τη λογική, δημιουργήθηκαν τα Login και Sign up components. Το Login-Sign up είναι δύο διαφορετικά components,με το Signup να βρίσκεται εμφωλευμένο στο Login. Αυτό σημαίνει πως όταν ο χρήστης πατάει το κουμπί για Login-Signup, του εμφανίζεται το login. Αν ο χρήστης του Word Wizardry δεν είναι εγγεγραμμένος, παρέχεται μέσα στο Login η δυνατότητα της μεταφοράς στο Signup,όπου και μπορεί να πραγματοποιήσει την εγγραφή του.



Σχήμα 4.2.5: Σχήμα επικοινωνίας Main Menu – Login – Sign Up

4.3.1 Δημιουργία Μοντέλου χρηστών

Αρχικά, για την δημιουργία, αποθήκευση και επιστροφή χρηστών στην εφαρμογή, είναι αναγκαία η δημιουργία ενός μοντέλου του mongoose και ενός server.

Στο μοντέλο περιγράφονται οι πληροφορίες που χρειάζονται να αποθηκεύονται για τον κάθε χρήστη. Αυτές είναι: το όνομα (name), το email, το password που θα επιλέξει ο χρήστης. Επίσης, αρχικοποιούνται πληροφορίες οι οποίες θα ενημερώνονται με την πρόοδο του χρήστη στο ‘Word Wizardry’,όπως το επίπεδο στο οποίο βρίσκεται ανά πάσα στιγμή (level), οι πόντοι οι οποίοι έχει αποκτήσει ο χρήστης (points), οι λέξεις, που αποτελούν το τελευταίο σταυρόλεξο το οποίο κλήθηκε να ολοκληρώσει (lastChosenWords), καθώς και τα γράμματα τα οποία και θα παρέχονται με σκοπό τη σχεδίαση αυτών των λέξεων στο χρήστη (lastChosenLetters). ‘Επειτα,δημιουργείται μία κενή collection στη βάση Mongo Db η οποία αναμένει την αποίκισή της από τα δεδομένα του χρήστη.

```
const userSchema = new Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  level: { type: Number, default: 1 },
  points: { type: Number, default: 0 },
  lastChosenWords: [{ type: String }],
  lastChosenLetters: [{type: String}]
});

const User = mongoose.model('User', userSchema);
User.createCollection().then(function (collection) {
    console.log('collection is created');
  })
```

Σχήμα 4.3.1: Schema, Mongoose Model και collection χρηστών

4.3.2 Δημιουργία server χρηστών

Στη συνέχεια, δημιουργήσαμε τον server με τον οποίο θα επικοινωνεί η εφαρμογή για την δημιουργία και επιστροφή δεδομένων του χρήστη. Η διαδικασία που ακολουθήθηκε για τη σύνταξη του server περιέχει την ίδια λογική που χρησιμοποιήθηκε και για το server των λέξεων. Κατ'αρχήν, δημιουργήσαμε το κομμάτι κώδικα συνδέει τον server με μια βάση δεδομένων MongoDB χρησιμοποιώντας το Mongoose, αλλά και η σύνταξη middleware για JSON και CORS, ώστε να επιτρέπεται ο server να δέχεται JSON δεδομένα, αλλά και να επιτρέπεται η βάση να δέχεται αιτήματα από διαφορετικές διευθύνσεις.

```
const express = require('express');
const mongoose = require('mongoose');
const User = require('.//UserModel');
const cors = require('cors');
const app = express();
const port = 3001;
const uri = 'mongodb+srv://stevekoulas:asfalisa1@staur

mongoose.connect(uri)
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB connection erro


app.use(express.json());
app.use(cors({ origin: 'http://localhost:3000' }));
```

Σχήμα 4.3.2.1: imports και σύνδεση στη Mongo του server των χρηστών

‘Επειτα, δημιουργήσαμε τους δρομολογητές για τη δημιουργία χρήστη και για την επαναφορά των δεδομένων αυτού. Η app.post('/users') δημιουργεί με την κλήση της ένα νέο χρήστη στη βάση, ενώ η app.get('/users') δημιουργήθηκε με σκοπό την επιστροφή όλων των user της εφαρμογής.

```
app.post('/users', async (req, res) => {
  const user = new User({
    name: req.body.name,
    email: req.body.email,
    password: req.body.password,
    level: req.body.level || 1,
    points: req.body.points || 0,
    lastChosenWords: req.body.lastChosenWords || [],
    lastChosenLetters: req.body.lastChosenLetters || [],
  });

  try {
    const newUser = await user.save();
    res.status(201).json(newUser);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

app.get('/users', async (req, res) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});
```

Σχήμα 4.3.2.2: Post και get χρηστών

Για την ανάπτυξη της εφαρμογής είναι απαραίτητη και η δημιουργία διαδρομών αρμοδίων για την ανάκτηση των δεδομένων ενός συγκεκριμένου χρήστη, αλλά και την ενημέρωση αυτών. Αυτό χρείζεται αναγκαίο από τη στιγμή που θέλουμε να αναπτύξουμε δυνατότητες όπως η επιστροφή και ενημέρωση των λέξεων που θα απαρτίζουν το σταυρόλεξο του χρήστη, τους πόντους τους οποίους έχει συλλέξει οι οποίοι και θα ενημερώνονται σε κάθε επίλυση ενός σταυρολέξου κτλ. Με τη δημιουργία αυτών των διαδρομών μπορεί να αναπτυχθεί και η λειτουργία του ‘continue’, η συνέχισης της πορείας του χρήστη από το σημείο στο οποίο είχε μείνει.

Με αυτή τη λογική, δημιουργήθηκε ένας δρομολογητής ανάκτησης των δεδομένων του χρήστη ανάλογα με το id- μοναδικό αναγνωριστικό του χρήστη (app.get('/users/:id'), αλλά και δρομολογητές

ενημέρωσης όλων των στοιχείων του χρήστη: name, email, password, level, points, lastChosenWords, lastChosenLetters. Ενδεικτικά, στον κώδικα παρακάτω, παρέχεται η ανάπτυξη του app.get('/users/:id'), αλλά και ο κώδικας του app.post('/users/:id/points' (καθώς η λογική της ανάκτησης και ενημέρωσης και των υπολοίπων στοιχείων που απαρτίζουν τους user είναι η ίδια).

```
app.get('/users/:id', getUser, (req, res) => {
  res.json(res.user);
});

app.post('/users/:id/points', getUser, async (req, res) => {
  if (req.body.points != null) {
    res.user.points = req.body.points;
  }

  try {
    const updatedUser = await res.user.save();
    res.json(updatedUser);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
```

Σχήμα 4.3.2.3: Ανάκτηση υποπεδίων χρηστών

Με την σύνταξη του μοντέλου και του server των χρηστών, μπορούμε να προχωρήσουμε στην ανάλυση της λογικής υλοποίησης του Login και του Sign Up.

4.4 Sign Up

Για να μπορεί να αποθηκεύεται η πορεία του κάθε χρήστη της εφαρμογής, ώστε να μπορεί να συνεχίσει το σταυρόλεξο από το σημείο που είχε μείνει ανεξαιρέτως του λογισμικού της συσκευής που χρησιμοποιεί (Windows ή android), πρέπει να ακολουθήσει μία συγκεκριμένη διαδικασία για να εγγραφεί σαν user του Word Wizardry. Αυτή η διαδικασία επιτελείται μέσω του component Sign Up. Το Sign Up μπορεί να αποθηκευσει το χρήστη, αφού έχει συμπληρώσει πρώτα αυτός τα στοιχεία του στις ειδικές φόρμες που διαθέτει το component.

4.4.1 User Interface του Sign Up

Μέσα από το Login, ο χρήστης πατώντας το button με όνομα Sign Up οδηγείται στο component Sign Up, ένα modal με παρόμοιο UI με το component Login,στο οποίο, τοποθετώντας τα στοιχεία του, μπορεί να εγγραφεί στην εφαρμογή. Αρχικοποιείται και εδώ μία μεταβλητή opened σε false η οποία, πατώντας το κουμπί για την είσοδο στο SignUp, μετατρέπεται σε true και μας επιτρέπει την εμφάνιση του SignUp.

```
<Group position="center">
  <Button style={{display: 'flex'}} onClick={() => setOpened(true)}>Sign Up</Button>
</Group>
```

Σχήμα 4.4.1: 'Ανοιγμα modal για το Sign Up

Τα στοιχεία που προαπαιτούνται για την εγγραφή στο 'Word Wizardry' είναι το όνομα (name),το email και ένα password. Αν ο χρήστης δεν είναι συνδεδεμένος (!isLoggedIn), εμφανίζονται τα πεδία για την εισαγωγή ονόματος, email, και κωδικού πρόσβασης, καθώς και ένα κουμπί "Sign Up".Αυτά τα πεδία αποτελούν και το userData. Παίρνουν σαν value τις τιμές που πατάει ο χρήστης και, με το πάτημα του Sign Up button, οι τιμές που θέτει ο χρήστης σε αυτά αποθηκεύονται στη βάση.

```
  {!isLoggedIn ?
  <div>
  <p style={{marginTop : '0em'}}>Add your name</p>
  <TextInput
  placeholder="Enter your name"
  value={name}
  onChange={(e) => setName(e.target.value)}
/>
  <p>Add your email</p>
  <TextInput
  placeholder="Enter your email"
  value={email}
  onChange={(e) => setEmail(e.target.value)}
/>
<p>Add your password</p>
  <TextInput
  placeholder="Enter your password"
  value={password}
  onChange={(e) => setPassword(e.target.value)}
/>
```

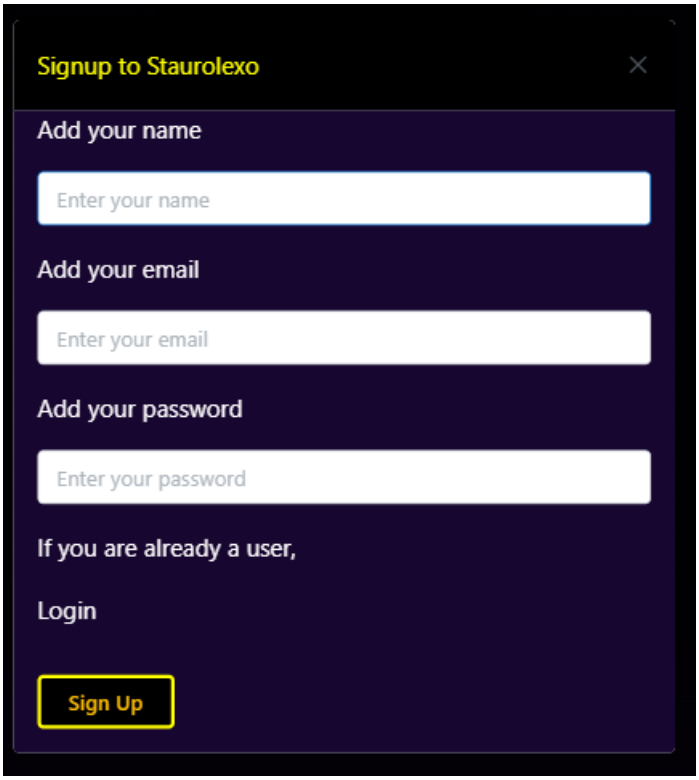
Σχήμα 4.4.2: Ορισμός πεδίων συμπλήρωσης

Αν ο χρήστης είναι συνδεδεμένος ή πραγματοποιηθεί επιτυχώς η εγγραφή του, το modal Sign Up μεταλλάσσεται. Αυτό γίνεται δημιουργώντας ένα διαφορετικό UI, με γνώμονα το αν δεν είναι εγγεγραμμένος ο χρήστης (!isLoggedIn ?) ή αν έχει καταχωρήσει επιτυχώς τα στοιχεία του (:). Αν η εγγραφή του χρήστη στεφθεί με επιτυχία, τη θέση της φόρμας συμπλήρωσης στοιχείων παίρνει μία σελίδα εμφάνισης των στοιχείων του χρήστη User Details.

```
  :
  <div>
  {userData && (
  <div>
    <h3>User Details</h3>
    <p>Name: {userData.name}</p>
    <p>Email: {userData.email}</p>
    <p>Level: {userData.level}</p>
    <p>Points: {userData.points}</p>
    <p>Last Chosen Words: {userData.lastChosenWords.join(', ')}</p>
  </div>
)}
</div>}
```

Σχήμα 4.4.3: Δεδομένα που κατέθεσε ο χρήστης στη φόρμα

‘Ετσι, οπτικά έχει δημιουργηθεί το modal SignUp, το οποίο και έχει την παρακάτω μορφή:



Σχήμα 4.4.3: Το modal Sign Up

4.4.2 Αποθήκευση Στοιχείων Sign Up

Για την αποθήκευση των στοιχείων που δίνει ο χρήστης, με το πάτημα του κουμπιού Sign Up, καλείται η ασύγχρονη function handleSignup. Αυτή αρχικά ελέγχει αν έχουν εισαχθεί τιμές στα πεδία name, email και password. Αν δεν έχει εισαχθεί τιμή σε κάποιο πεδίο, εμφανίζει ένα μήνυμα alert που ειδοποιεί το χρήστη να συμπληρώσει όλα τα πεδία.

```
const handleSignup = async () => {

  if (!name || !email || !password) {
    alert('Please fill in all fields');
    return;
  }
```

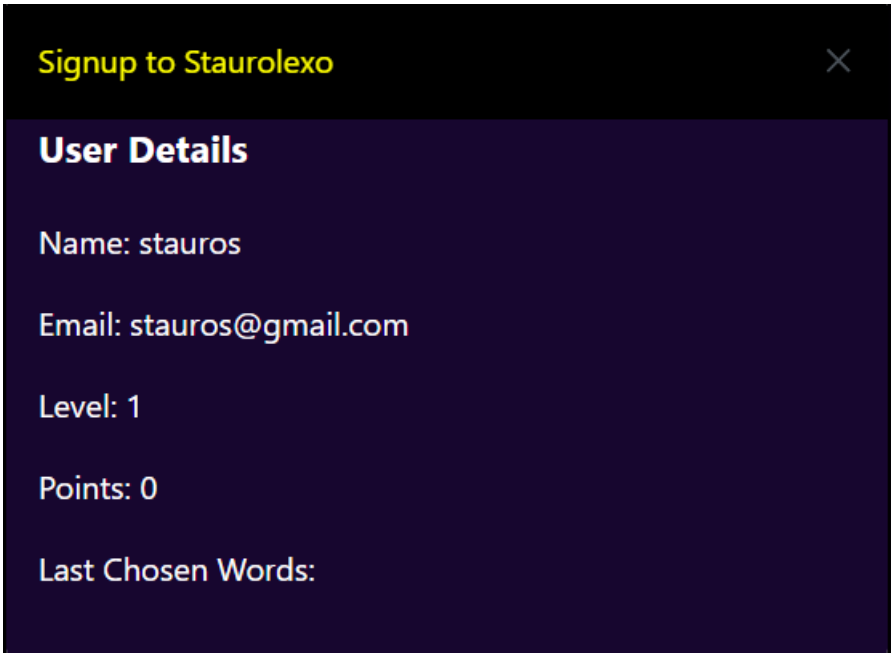
Σχήμα 4.4.2.1: 'Ελεγχος συμπλήρωσης των απαραίτητων πεδίων

Αν ο χρήστης συμπληρώσει όλα τα προαπαιτούμενα πεδία, πραγματοποιείται μία function post στο endpoint users, μέσω της οποίας αποθηκεύονται τα name,email,password σε μία νέα εγγραφή, δημιουργώντας έτσι ένα νέο χρήστη. Τέλος, ο χρήστης ειδοποιείται ότι πραγματοποιήθηκε η εγγραφή του στην εφαρμογή με επιτυχία, μέσω ενός alert.

```
  try {
  const response = await axios.post('http://localhost:3001/users', { name, email, password });
  alert('User created successfully!');

  const newUser = response.data;
  setUserData(newUser);
  alert('User created successfully!');
  setIsLoggedIn(isLoggedIn => isLoggedIn = true);
} catch (error) {
  console.error('Error creating user:', error);
  alert('Error creating user. Please try again.');

}
```

Σχήμα 4.4.2.2: Αποστολή δεδομένων χρήστη στη βάση

Αν ο χρήστης είναι συνδεδεμένος, εμφανίζονται οι πληροφορίες του χρήστη (userData) όπως το όνομα, το email, το επίπεδο, οι πόντοι, και οι τελευταίες επιλεγμένες λέξεις.



Σχήμα 4.4.2.3: Εμφάνιση δεδομένων στο χρήστη

Η δημιουργία του Sign Up έχει ολοκληρωθεί. Αν η διαδικασία έχει ολοκληρωθεί σωστά,το Sign Up δημιουργεί ένα νέο χρήστη και μας εμφανίζει τα στοιχεία του ή, σε περίπτωση λάθους στη διαδικασία, εμφανίζεται μήνυμα λάθους και δεν εγγράφεται χρήστης.

4.4.3 Εξακρίβωση Αποθήκευσης στη Βάση

Σαν δημιουργοί της εφαρμογής, αν οδηγηθούμε στο γραφικό περιβάλλον της Mongo DB, μπορούμε να εξακριβώσουμε μία εγγραφή. ‘Ετσι, βάζοντας το όνομα, το email και ένα τυχαίο password, μπορούμε να εντωπίσουμε ότι όντως έχει εγγραφεί ένας χρήστης στο collection users της Mongo με αυτά τα στοιχεία. Ακόμα, είναι παρατηρήσιμη και η λειτουργικότητα που προσθέσαμε για τον ορισμό του επιπέδου (level) σε 1 και των πόντων (points) που έχει συλλέξει ο χρήστης σε 0.

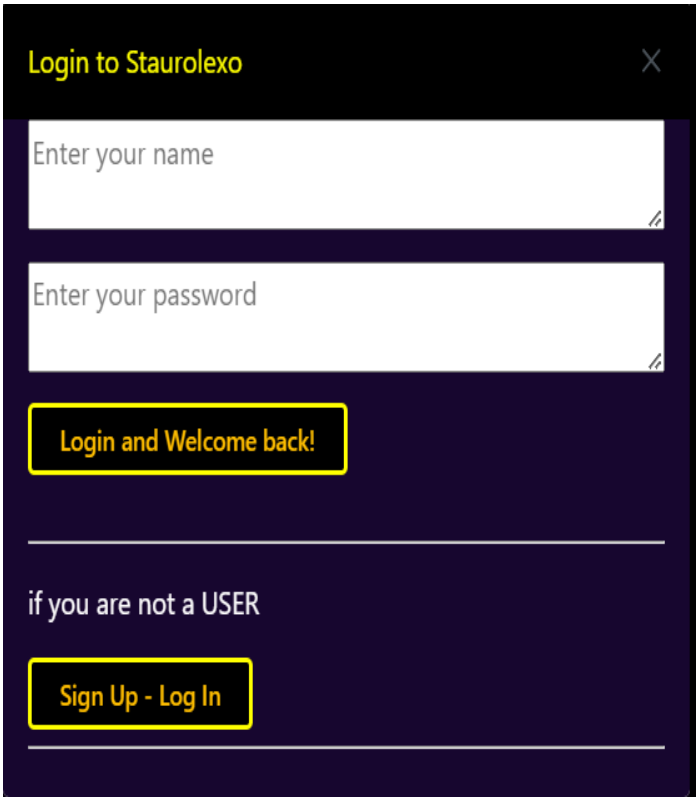
Σχήμα 4.4.3: Αποθηκευμένα δεδομένα χρήστη στη MongoDB

4.5 Login

Για να μπορέσει να επαναφέρει ο χρήστης την πορεία του και, λήνωντας σταυρόλεξα, να ενημερώσει αυτή, πρέπει να κάνει Login στο Word Wizardry, χρησιμοποιώντας τα στοιχεία που έχει ήδη δηλώσει κατά το Sign Up.

4.5.1 Τρόπος λειτουργείας Login

Πατώντας το Button ‘Log In – Sign Up’ ανοίγει ένα modal του Mantine, το οποίο και περιέχει τον κώδικα και τη λειτουργικότητα του Login. Το modal (επίσης αποκαλείται modal window ή lightbox) είναι ένα στοιχείο ιστοσελίδας που εμφανίζεται μπροστά από όλο το υπόλοιπο περιεχόμενο της σελίδας και το απενεργοποιεί. Για να επιστρέψει ο χρήστης στο κύριο περιεχόμενο πρέπει να αλληλεπιδράσει με το modal ολοκληρώνοντας μια ενέργεια ή κλείνοντάς το[42],πατώντας το κουμπί ακύρωσης και μην ολοκληρώνοντας τη διαδικασία . Το modal Login ανοίγει και ζητάει την συμπλήρωση του πεδίου του ονόματος και του password, αν ο χρήστης είναι ήδη εγγεγραμμένος στην εφαρμογή. Κάτω από το κουμπί ‘Login and Welcome back!’ υπάρχει το κουμπί sign up για τους μη εγκεκριμμένους χρήστες.



Σχήμα 4.5.1 Το modal Login

4.5.2 User Interface του Login

Για το UI του login, δημιουργήθηκαν δύο textarea, ένα για το name και ένα για το password. Τα κουμπιά τόσο για το 'Login and Welcome back!' όσο και για τη δρομολόγηση στο Sign Up αν δεν είναι εγγεγραμμένος ο χρήστης υιοθετούνται από το styling που έχουμε θέσει στο αρχείο index.js. Πάνω και κάτω από το πλαίσιο για τη μετάβαση στο Sign Up έχουν τοποθετηθεί δύο στοιχεία hr (οριζόντιες γραμμές). Για την εξασφάλιση της αρμονίας του UI του Login με το UI του MainMenu, ανατέθηκε μαύρο χρώμα στο header ('Login to Staurolexo') και μωβ-purple χρώμα στο υπόλοιπο modal.

4.5.3 Κώδικας Login

Το Login αποτελεί και αυτό ένα modal, όπως το Sign Up. Έτσι, αρχικοποιείται μία μεταβλητή opened η οποία χρησιμοποιείται για το άνοιγμα και κλείσιμο του modal, καθώς και τα πεδία name, password, userData. Τα δύο πρώτα θα τα συμπληρώνει ο χρήστης μέσω τον textarea. Το userData θα επιστρέφεται αν τα name και password που εισήγαγε ο χρήστης υπάρχουν στη βάση.

```
const [opened, setOpened] = useState(false);
const [name, setName] = useState('');
const [password, setPassword] = useState('');
const [userData, setUserData] = useState(null);
```

Σχήμα 4.5.3.1 Ορισμός μεταβλητών για τους χρήστες

Όταν ο χρήστης πατήσει το κουμπί "Login", η Πατώντας το button Login, καλείται η function handleLogin, η οποία αρχικά αναλαμβάνει την εμφάνιση ενός μηνύματος σφάλματος αν ο χρήστης δεν έχει συμπληρώσει όλα τα πεδία της φόρμας.

```
const handleLogin = async () => {
  if (!name || !password) {
    alert('Please fill in all fields');
    return;
  }
```

Σχήμα 4.5.3.2 'Ελεγχος συμπλήρωσης δεδομένων χρήστη

Στη συνέχεια, η handleLogin στέλνει ένα αίτημα POST στον διακομιστή με τα στοιχεία αυτά για έλεγχο ταυτότητας. Αν η σύνδεση είναι επιτυχής, τα δεδομένα του χρήστη αποθηκεύονται στο userData.

```
try {
  const response = await axios.post('http://localhost:3001/login', { name, password });
  setUserData(response.data);
  console.log(response.data);
  console.log("im trying to get in");
  setTimeout(() => {
    console.log("im in");
    console.log(response.data);
  }, 10000);
} catch (error) {
  console.error('Σφάλμα κατά τη φόρτωση των λέξεων:', error);
}
```

Σχήμα 4.5.3.3 Σύδεση χρήστη στη βάση

Επιπλέον, το modal περιέχει έναν σύνδεσμο για την εμφάνιση του Signup component.

5 Αρχείο Staurolexo

Επιλέγοντας ένα επίπεδο μέσω του ανάλογου κουμπιού μεταφερόμαστε στη σελίδα Staurolexo, που είναι η σελίδα που περιέχει και εκτελεί το παιχνίδι. Η σελίδα Staurolexo εμπεριέχει διαφορετικά components,όπως το σταυρόλεξο με τις λέξεις που πρέπει να βρει ο χρήστης,το πλαίσιο των γραμμάτων από τα οποία ο χρήστης μπορεί να δημιουργήσει συνδιασμούς και εν τέλει λέξεις με αυτά τα γράμματα (wordSwipe),καθώς και ένα σύστημα χρήσης των πόντων που έχουν συλλεχθεί με σκοπό την εμφάνιση μίας λέξης του σταυρολέξου.

Πριν την συσχέτιση με αυτά τα components όμως, το αρχείο Staurolexo αποτελείται από μία δική του λειτουργικότητα για την δημιουργία του παιχνιδιού,παρέχοντας σαφείς εντολές στα components για τη διαχείρισή τους, διαθέτοντάς τους τις κατάλληλες μεταβητές για σύνταξη και παραμετροποίηση, αλλά είναι και το component το οποίο λειτουργεί σαν δίαυλος μεταξύ των διαφορετικών στοιχείων του παιχνιδιού με τον server, εμπλουτίζοντας αυτά με τα κατάλληλα στοιχεία από τη βάση, αλλά και επιστρέφοντας στη βάση τις κινήσεις και τις επιλογές του χρήστη, όπως το επίπεδο στο οποίο βρίσκεται,τις λέξεις που έχει βρει κτλ. Συνοψίζοντας, η σελίδα Staurolexo λειτουργεί σαν ένα parent Component στα άλλα components, αλλά και ως ένας οδηγός χρήσης και διαχειριστής του παιχνιδιού.

5.1 Διαδικασία Δημιουργίας Γραμμάτων

Αρχικά, δημιουργήθηκαν δύο πίνακες· ο ένας απαρτίζεται από το σύνολο των φωνηέντων που απαρτίζουν το ελληνικό λεξιλόγιο και ο άλλος με το σύνολο των συμφώνων. Μετά από αυτούς τους πίνακες,δημιουργήθηκαν άλλοι δύο που αφορούν το σύνολο των συμφώνων και των φωνιέντων που θα έχει το κάθε επίπεδο ως διαθέσιμα γράμματα.

```
const Fwnienta = ['Α', 'Ε', 'Η', 'Ι', 'Ο', 'Υ', 'Ω'];
const Sumfwna = ['Β', 'Γ', 'Δ', 'Ζ', 'Θ', 'Κ', 'Λ', 'Μ', 'Ν', 'Ξ', 'Π', 'Ρ', 'Σ', 'Τ', 'Φ', 'Χ', 'Ψ'];
const numberOfSumfwna = [1, 2, 2, 3, 3, 4, 4, 5, 5,];
const numberOfFwnienta = [2, 2, 3, 3, 4, 4, 5, 5, 6,];
```

Σχήμα 5.1Δημιουργία πινάκων με φωνίεντα, σύμφωνα και αναλογίας αυτών σε κάθε επίπεδο

Η λογική της δημιουργίας των παραπάνω πινάκων είναι η εξής : Σε κάθε επίπεδο δημιουργείται ένας τυχαίος συνδιασμός γραμμάτων (φωνιέντων και συμφώνων) αναλόγως το επίπεδο που επέλεξε ο χρήστης. Το μέγεθος αυτού του συνδιασμού παραλλάσεται από επίπεδο σε επίπεδο, καθώς όσο το επίπεδο ανεβαίνει,τόσο μεγαλώνει και το μέγεθος του συνδιασμού γραμμάτων που απαρτίζουν το κάθε επίπεδο. Κατά συνέπεια, το επίπεδο 3 θα έχει περισσότερα γράμματα από το επίπεδο 1. Κατόπιν, αυτά τα γράμματα θα χρησιμοποιηθούν για την ανάκτηση λέξεων από τη βάση. Η επιστροφή δηλαδή των λέξεων βασίζεται στη δημιουργία τυχαίων γραμμάτων σε κάθε επίπεδο.

Η διαδικασία της τυχαίας επιλογής γραμμάτων γίνεται με χρήση των πινάκων numberOfSumfwna και numberOfFwnienta. Το κάθε element αυτών των πινάκων αφορά και ένα διαφορετικό επίπεδο. Ειδικότερα, το πρώτο element στους πίνακες (το 1 στον numberOfSumfwna και το 2 στον numberOfFwnienta) αφορά το πλήθος των φωνιέντων και συμφλωνων που θα έχει το πρώτο

επίπεδο: 1 σύμφωνο και 2 φωνίεντα. Αναλόγως, το επίπεδο 2 θα χρησιμοποιήσει το δεύτερο element των πινάκων,άρα θα έχει 2 φωνίεντα και 2 σύμφωνα, το τρίτο επίπεδο θα έχει 3 φωνίεντα και 2 σύμφωνα και ούτω κάθε εξής.

5.2 Πολλαπλή εμφάνιση ίδιου γράμματος

Ένα πολύ πιθανό ενδεχόμενο είναι στη τυχαία επιλογή των γραμμάτων να προκύψει ένας συνδιασμός γραμμάτων ο οποίος θα περιέχει πάνω από μία φορά το ίδιο γράμμα. Αυτό μπορεί να δημιουργήσει δύο πιθανά προβλήματα. 'Ενα προβλήμα που μπορεί να προκύψει είναι η δημιουργία τριών ή τεσσάρων επαναλήψεων του ίδιου γράμματος σε ένα χαμηλό επίπεδο (π.χ. να βρισκόμαστε στο 5^{ο} επίπεδο και τα φωνίεντα που μας διατίθενται να είναι 3 επαναλήψεις του 'Α' και ένα 'Ε'). 'Αλλο ένα πιθανό σενάριο που πρέπει να έχουμε υπ'όψην, ανεξαρτήτως επιπέδου, είναι ολόκληρη η αλληλουχία των συμφώνων ή των φωνιέντων να αποτελείται απο επαναλήψεις του ίδιου γράμματος (π.χ. να είμαστε στο επίπεδο 7 και να επιστρέφεται 5 φορές το 'Α'). Αυτές οι περιπτώσεις αποτελούν προβλήματα καθώς μειώνουν τον αριθμ'ό των λέξεων που μπορούν να δημιουργηθούν και μπορούν να καταστήσουν αδύνατη τη δημιουργία σταυρολέξου, ή να δημιουργηθεί ένα σταυρόλεξο δυσανάλογης δυσκολίας για το χρήστη. Έτσι, στη δημιουργία των τυχαίων γραμμάτων που θα χρησιμοποιηθούν στο σταυρόλεξο, πρέπει να οριστούν περιορισμοί και κανόνες για την αποφυγή των συγκεκριμλενων ζητημάτων.

5.3 Δημιουργία τυχαίων γραμμάτων για εμπλουτισμό επιπέδου

Τη δημιουργία των συνδιασμών των γραμμάτων ανέλαβε η function letterConfiguration. Το επίπεδο που επέλεξε ο χρήστης εισάγεται ως τιμή τόσο στον πίνακα των συμφώνων,όσο και σε αυτόν των φωνιέντων (η αιτία χρήσης του [level -1] αφορά το ότι τα elements ενός array ξεκινούν από το 0, ήτοι το πρώτο element είναι το στοιχείο 0 στον πίνακα).Ακόμα,αρχικοποιήθηκε ο πίνακας filteredLettersArray, ο οποίος και θα περιέχει τα γράμματα που θα παράγονται τυχαία από τη function.

'Υστερα,ο πίνακας filteredLettersArray εμπλουτίζεται με τυχαία σύμφωνα και φωνίεντα,αναλόγως του επιπέδου, μέσω των δύο for. Για κάθε αριθμό συμφώνων και φωνιέντων, παράγονται τα αντίστοιχα,τυχαία φωνίεντα η σύμφωνα. Αυτό γίνεται χρησιμοποιώντας τη μέθοδο mathRandom, δημιουργώντας ένα τυχαίο αριθμό για κάθε γράμμα. Αυτός ο τυχαίος αριθμός αντικατοπτρίζει και ένα element των πινάκων Fwnienta και Symfwna,το οποίο και θα εισάγουμε στον πίνακα filteredLettersArray. Αυτό γίνεται σύμφωνα με τους πίνακες numberOfFwnienta και numberOfSymfwna,επαναλαμβάνοντας τη διαδικασία δημιουργίας ενός τυχαίου αριθμού, ο οποίος και θα είναι με άλλα λόγια και ένα element στους πίνακες των υμφώνων και των φωνιέντων, Για παράδειγμα, αν ο numberOfSumfwna είναι 5 και ο numberOfFwnienta είναι 6, επιλέγονται, δημιουργώντας τους τυχαίους αριθμούς ως position of element στους array, 5 τυχαία σύμφωνα από τον πίνακα Symfwna και 6 τυχαία φωνίεντα από τον Fwnienta. Μετά, αυτά τα γράμματα εντάσσονται στον filteredLettersArray. Με αυτό το τρόπο, ο filteredLettersArray επιτυγχάνει να διαθέτει το

συνδιασμό των φωνιέντων και συμφώνων εκ των οποίων θα δημιουργηθεί το σταυρόλεξο και θα εισαχθούν οι λέξεις. Ο filteredLettersArray εισέρχεται ως τιμή και του wordSwipeLetters, ο οποίος είναι και ο πίνακας που θα χρησιμοποιηθεί στη σύνταξη του πλαισίου για τη δημιουργία λέξεων (των κουμπιών που θα κάνει swipe ο χρήστης).

Η function letterConfiguration φυσικά τρέχει κάθε φορά που επιλέγουμε ένα επίπεδο και οδηγούμαστε στη σελίδα Staurolexo. Κάθε φορά που ο χρήστης διαλέγει κάποιο επίπεδο, θέλουμε αυτόματα να δημιουργούνται τα τυχαία γράμματα του σταυρολέξου. Αυτό γίνεται μέσω της useEffect, η οποία, όπως προείπαμε, είναι ένα ειδικό hook της React, η οποία μας επιτρέπει σε κάθε render μίας σελίδας να τρέχουμε τις function της επιλογής μας.

5.4 Διαδικασία Επικοινωνίας με server για εύρεση και απόκτηση γραμμάτων

‘Οπως δηλώθηκε, το αρχείο Staurolexo είναι, εκτός των άλλων, το αρχείο το οποίο λειτουργεί ως γέφυρα μεταξύ του παιχνιδιού και του server. Ακριβέστερα, ο κώδικας του Staurolexo σχηματίζει τυχαία γράμματα, τα οποία και περνάει στο server, ώστε να αναζητήσει αυτός λέξεις που απαρτίζονται από αυτά τα γράμματα και να τις επιστρέψει.

```
const fetchData = async (wordswipeLetters, source) => {
  try {
    const response = await axios.post('http://localhost:3001/words', { array1: wordswipeLetters }, { cancelToken: source.token });
    setMatchingEntries(response.data);
  } catch (error) {
    if (axios.isCancel(error)) {
      console.log('Request canceled', error.message);
    } else {
      console.error('Error fetching data:', error);
    }
  }
};
```

Σχήμα 5.4.1 Ανάκτηση λέξεων από το server

Η φάση αυτή διενεργείται με τη function fetchData, μιας async function, εν ολίγοις μιας function η οποία περιμένει την επιστροφή μιας τιμής (στο στάδιο αυτό περιμένει την επιστροφή των λέξεων), ώστε να ολοκληρωθεί και να επιστρέψει τη διενέργεια της συνέχειας του κώδικα. Η fetchData προωθεί στον server τα γράμματα μέσω του post και θέτει σαν matrchingEntries τις λέξεις που αποδίδει ο server. Η διαχείριση των γραμμάτων, εύρεση λέξεων και επιστροφή αυτών από το server θα διευκρινηστεί στη συνέχεια,στο κεφάλαιο Wordserver.

Η fetchData χρησιμοποιεί το cancel token της βιβλιοθήκης axios για να σταματήσει την επικοινωνία με το server σε περίπτωση που ο χρήστης αποχωρήσει από τη σελίδα Staurolexo (axios.isCancel) ή σε περίπτωση που δεν βρεθούν αποτελέσματα (error fetching data). Αυτό γίνεται με σκοπό την εξοικονόμιση χώρου και την ευελιξία της εφαρμογής, καθώς το να τρέχει ο server ενώ ο χρήστης έχει αποχωρήσει μπορεί να καταστήσει την εφαρμογή αργή, αφού ο server θα αναζητά αποτελέσματα πάνω σε συνδιαδμούς γραμμάτων που πλέον δεν υφίστανται.

```
useEffect(() => {
  const source = axios.CancelToken.source();
  if (wordswipeLetters.length > 0) {
    fetchData(wordswipeLetters, source);
  }

  return () => {
    source.cancel('Component unmounted, canceling request');
  };
}, [wordswipeLetters]);
```

Σχήμα 5.4.2 UseEffect λήψης λέξεων

Η function fetchData τρέχει και αυτή με το rendering της σελίδας,ακριβώς αφού γεμίσει ο πίνακας wordswipeLetters, που είναι ο πίνακας που θα αποτελεί τα γράμματα του πλαισίου. Μέσα στη useEffect διαμορφώνεται ένα νέο CancelToken που χρησιμοποιείται για να ακυρώσει την αίτηση Axios αν χρειαστεί. Καλείται τη συνάρτηση fetchData, περνώντας τα wordswipeLetters και το CancelToken ως παραμέτρους. Τέλος,επιστρέφει μια συνάρτηση καθαρισμού που ακυρώνει την αίτηση Axios όταν το component αποσυνδέεται (unmounts).

5.5 WordServer

‘Οταν ένας χρήστης επιλέγει ένα επίπεδο,τρέχει ο αλγόριθμος ο οποίος δημιουργεί ένα τυχαίο πίνακα γραμμάτων clickedLetters (με αριθμό γραμμάτων ανάλογο του επιπέδου που επέλεξε). Οι λέξεις που θα υπάρχουν στο σταυρόλεξο θα είναι ένας συνδιασμός αυτών των γραμμάτων. Οπότε,έχοντας τα γράμματα που δημιουργήθηκαν αυτόματα,γίνεται μία κλήση μέσω του server στη βάση. Ο κώδικας της κλήσης θα καλεί από τη βάση τις λέξεις οι οποίες απαρτίζονται από τα γράμματα που δημιουργήθηκαν τυχαία. Θα επιστρέφονται μόνο οι λέξεις οι οποίες απαρτίζονατι από κάποια από τα γράμματα του clickedLetters, αλλά και που δεν θα περιέχουν κανένα γράμμα το οποίο δεν υπάρχει στον clickedLetters. Για παράδειγμα,θέλουμε αν τα γράμματα είναι τα ‘A’,’P’,’M’,’A’, από τη βάση να έρχεται η λέξη ‘APA’, αλλά όχι η λέξη ‘APME’ (καθώς περιέχει διαφορετικά γράμματα από αυτά που δημιούργησε ο clickedLetters και συγκεκριμένα το ‘E’).

```
const express = require('express');
const mongoose = require('mongoose');
const Word = require('./WordsSchema');
const cors = require('cors');

const app = express();
const port = 3001;

const uri = 'mongodb+srv://stevekoulas:asfalisa1@staurolexo.zjs7exc.mongodb.net/?retryWrites=true
mongoose.connect(uri)
  .then(() => {
    console.log('Connected to MongoDB');
    app.listen(port, () => {
      console.log(`Server is running on port ${port}`);
    });
  })
  .catch((error) => {
    console.error('Error connecting to MongoDB:', error);
  });

app.use(express.json());
app.use(cors({ origin: 'http://localhost:3000' }));
```

Σχήμα 5.5.1 Σχηματισμός server ανάκτησης λέξεων

Κατ' αρχήν, στον wordServer γίνεται η εισαγωγή των απαραίτητων dependencies. Αυτά είναι το express ώστε να δημιουργήσει το διακομιστή και το mongoose για σύνδεση και επικοινωνία με τη βάση, το μοντέλο των λέξεων ονόματι Word από το WordsSchema.

Διαμοίραση Πόρων μεταξύ Διαφορετικής Προέλευσης CORS [44] (cross-origin requests) είναι ένας μηχανισμός βασισμένος σε HTTP-κεφαλίδες, που επιτρέπει σε έναν διακομιστή να δηλώσει οποιεσδήποτε προελεύσεις (όπως τομέα, σχήμα ή θύρα) διαφορετικές από τη δική του, από τις οποίες ένας φυλλομετρητής θα πρέπει να επιτρέψει τη φόρτωση πόρων. 'Ετσι, με την προσθήκη του πακέτου cors μπορούμε να επιτρέψουμε την πρόσβαση στην ιστοσελίδα σε χρήστες με διαφορετικές IP.

Μετά, δημιουργήθηκε μια νέα εφαρμογή express ονόματι app,όπως και η πόρτα (port) στην οποία θα 'ακούει' αυτή η εφαρμογή.Το uri είναι η διεύθυνση και τα διαπιστευτήρια της βάσης δεδομένων που θα χρησιμοποιηθεί. Πρόκειται για το κλειδί της ένωσης του server με τη βάση που έχει σχηματιστεί στο περιβάλλον Mongo Db.

Η μέθοδος connect είναι η μέθοδος που χρησιμοποιείται για τη διασύνδεση με τη βάση,περνόντας της ως παράμετρο το uri (mongoose.connect(uri)). Λέμε λοιπόν στο mongoose να χρησιμοποιήσει τη μέθοδο connect, την ειδική εντολή για επικοινωνία με τη database, δίνοντας και τα διαπιστευτήρια αυτής, ώστε να μπορέσει να γίνει η επικοινωνία.

Για την επιστροφή των επιθυμητών δεδομένων από το server έχει γραφτεί φυσικά ο κατάλληλος κώδικας ο οποίος θα κάνει ελέγχους στα δεδομένα. 'Οπως έχουμε ήδη αναφέρει, οι λέξεις που

θέλουμε να επιστρέφονται πρέπει να αποτελούνται από τα διαθέσιμα γράμματα που σχηματίζονται. ‘Ετσι, καταλαβαίνουμε ότι δεν πρέπει να εμφανίζονται λέξεις που να περιέχουν έστω και ένα γράμμα το οποίο δεν είναι διαθέσιμο στον wordSwipeLetters. Επίσης, για να χρησιμοποιηθεί παραπάνω από μία φορά το ίδιο γράμμα, θα πρέπει να υπάρχει παραπάνω από μία φορά διαθέσιμο αυτό το γράμμα στο wordswipeLetters. Για παράδειγμα, αν τα διαθέσιμα γράμματα του wordswipeLetters είναι τα [‘Α’, ‘Π’, ‘Λ’, ‘Α’], δεν θα πρέπει να επιστρέφεται η λέξη ‘ΑΠΑΛΑ’, καθώς περιέχει τρεις φορές την εμφάνιση του γράμματος ‘Α’, ενώ ο wordswipeLetters περιέχει μόνο δύο οντότητες του ‘Α’. ‘Ετσι,είναι σημαντική η ανάπτυξη μεθόδου η οποία να ελέγχει τις επαναλήψεις των γραμμάτων ώστε ο server να επιτρέπει την επιστροφή λέξεων που χρησιμοποιούν ακριβώς τα διαθέσιμα γράμματα ή λιγότερα, ποτέ περισσότερα.

Γι’ αυτό το λόγο, αναπτύχθηκε η μέθοδος getFrequency. Η μέθοδος getFrequency παίρνει ως παράμετρο τον array των γραμμάτων wordSwipeLetters και υπολογίζει τη συχνότητα εμφάνισης του κάθε γράμματος.

Πρώτα απ’όλα, δημιουργεί μία μεταβλητή freq, η οποία θα α. ‘Επειτα, μέσω του βρόγχου επανάληψης for, ελέγχει κάθε στοιχείο του array αν υπάρχει ήδη. Αν δεν υπάρχει, του προσδιορίζει την τιμή 0 αρχικοποιώντας τον και έπειτα του αναθέτει την τιμή 1,αφού είναι η πρώτη του εμφάνιση. Αν υπάρχει, παίρνει την τιμή που είχε ήδη και την αυξάνει κατά 1. Τέλος, επιστρέφει για κάθε στοιχείο του array τη συχνότητα εμφάνισης του κάθε στοιχείου στον πίνακα. ‘Ετσι, έχει ήδη δημιουργηθεί η function η οποία μετράει τις εμφανίσεις των στοιχείων,την οποία μπορούμε να εισάγουμε στον έλεγχο ώστε να υπολογίζουμε αν ταιρίαζει ή δεν ταιρίαζει η κάθε λέξη της βάσης με τον wordSwipeLetters.

```
function getFrequency(arr) {
  const freq = {};
  for (let char of arr) {
    freq[char] = (freq[char] || 0) + 1;
  }
  return freq;
}
```

Σχήμα 5.5.2 Function εύρεσης συχνότητας εμφάνισης γραμμάτων

Για την ανάκτηση των λέξεων, αναπτύχθηκε κώδικας που αρχικά χτυπάει το κατάλληλο endpoint στη βάση που είναι το collection με τις λέξεις μέσω της post. ‘Επειτα,δεχόμενος το array με τα επιλεγμένα γράμματα clickedLetters αναζητά στη βάση τα αποτελέσματα τα οποία να επιβεβαιώνουν τον κανόνα που θέσαμε. Αρχικά, τίθεται ο περιορισμός οι λέξεις που θα επιστραφούν να έχουν μέγεθος ίσο η μικρότερο του clickedLetters μέσω του find, το οποίο είναι ένα μια μέθοδος για τους πίνακες που αναζητεί και επιστρέφει τα αποτελέσματα τα οποία ικανοποιούν τις συνθήκες που θέσαμε. Ο αριθμός των αποτελεσμάτων περιερίζεται σε 25, κάτι που σημαίνει πως θα επιστραφούν τα πρώτα 25 αποτελέσματα που ικανοποιούν τους κανόνες που θέσαμε. Αυτό γίνεται για 2 λόγους:

Πρώτον, για ελαχιστοποίηση του χρόνου απόκρισης του server και την λήψη αποτελεσμάτων. Αφού περιορίζουμε τα δεδομένα που μπορούμε να εισπράξουμε από τον server, το σταυρόλεξο

δημιουργείται πιο γρήγορα και ο χρήστης μπορεί να παίξει το παιχνίδι ταχύτατα, χωρίς ανευ λόγου καθυστερήσεις.

Δεύτερον, το όριο των 25 λέξεων είναι απολύτως αποδεκτό,καθώς τα σταυρόλεξα που δημιουργούνται έχουν λιγότερες από 25 λέξεις προς εύρεση. Ο λόγος που δίνονται 25 λέξεις είναι ώστε να αποφευχθούν παρόμοιες λέξεις ως προς το περιεχόμενο των γραμμάτων.Η ποικιλία στους συνδιασμούς γραμμάτων μετατρέπει τα σταυρόλεξα λιγότερο προβλέψημα και με μεγαλύτερο δείκτη δυσκολίας ως προς την επίλυσή τους.Ακόμα,αποφεύγεται η εμφάνιση λέξεων που δεν ταιριάζουν μεταξύ τους ή λέξεων που δυσκολεύουν τη δημιουργία του σταυρολέξου (πχ οι λέξεις ει, ευ ανήκουν σε αυτή την κατηγορία καθώς μπορούν να κολλήσουν δύσκολα με άλλες λέξεις σε μεγάλα επίπεδα).

Το batchSize(100) καθορίζει ότι το μέγεθος της παρτίδας (batch) κατά την ανάγνωση των εγγράφων από τη βάση δεδομένων θα είναι 100. Ειδικότερα, κάθε φορά θα ελέγχονται 100 έγγραφα από τη βάση (100 λέξεις) για το ενδεχόμενο του να ικανοποιούν τον κανόνα. Η μέθοδος lean ορίζει ότι τα δεδομένα της βάσης θα διαχειρίζονται ως Javascript objects,παραλείποντας τις επιπρόσθετες λειτουργίες που προσφέρει η διαχείριση εγγράφων της Mongo Db. Επίσης,μέσω του cursor,δηλώνουμε ότι θα επιστραφεί ένας cursor αντί για έναν πίνακα. Ο cursor επιτρέπει τη διαδοχική ανάγνωση των αποτελεσμάτων και δεν απαιτεί τη φόρτωση όλων των εγγράφων στη μνήμη ταυτόχρονα. Αυτές οι μεθόδοι προστέθηκαν για τη βελτιστοποίηση της επιστροφής και διαχείρισης δεδομένων από τη βάση. Καθιστούν τον κώδικα πιο απλό και διαχειρίσιμο,αποφεύγοντας την υπερφόρτωση δεδομένων. ‘Ετσι διαδικασία της ανάκτησης των λέξεων γίνεται πολύ πιο γρήγορα.

```
app.post('/words', async (req, res) => {
  const { array1 } = req.body;

  try {
    const cursor = Word.find({
      $or: [
        { $expr: { $lte: [{ $size: "$grammata" }, array1.length] } },
        { grammata: { $eq: array1 } }
      ]
    }).batchSize(100).lean().cursor();
```

Σχήμα 5.5.3 Ανάκτηση λέξεων μέσω της post

Μετά, ακολουθεί ο ορισμός των κανόνων για τις λέξεις που θα επιστραφούν,καθώς και η λήψη των λέξεων που ικανοποιούν τους κανόνες από το server. Δηλώνουμε μια κενή λίστα (array) με το όνομα matchingEntries.Η λίστα matchingEntries θα χρησιμοποιηθεί για να αποθηκεύσει τα έγγραφα (words) που πληρούν τις συνθήκες της αναζήτησης. Το σύνολο uniqueWords θα χρησιμοποιηθεί για να αποθηκεύσει τις μοναδικές λέξεις (words) που έχουν ήδη βρεθεί και αποθηκευτεί στο matchingEntries.Η χρήση του Set εξασφαλίζει ότι δεν θα υπάρχουν διπλότυπες λέξεις στη λίστα matchingEntries.

```
let matchingEntries = [];
let uniqueWords = new Set();
```

Σχήμα 5.5.4 Ορισμός μεταβλητών για λέξεις που ταιριάζουν και για έλεγχο μοναδικότητας αυτών

Καθώς ο κώδικας επεξεργάζεται τα έγγραφα από τη βάση δεδομένων, κάθε λέξη που πληροί τις συνθήκες της αναζήτησης ελέγχεται αν υπάρχει ήδη στο uniqueWords Set.Αν η λέξη δεν υπάρχει στο uniqueWords, προστίθεται στο matchingEntries και στο uniqueWords.Αυτή η διαδικασία εξασφαλίζει ότι η λίστα matchingEntries περιέχει μόνο μοναδικές λέξεις που πληρούν τα κριτήρια της αναζήτησης. Αυτή η προσέγγιση είναι χρήσιμη για την αποφυγή διπλοτύπων και τη διατήρηση μιας καθαρής λίστας αποτελεσμάτων που πληρούν τις καθορισμένες συνθήκες.

Πρώτα απ’ όλα , δημιουργείται μία for και γίνεται Δήλωση και αρχικοποίηση του βρόχου χρησιμοποιώντας cursor, που είναι ένας δείκτης που δείχνει στη σειρά εγγράφων που επιστρέφονται από τη βάση δεδομένων MongoDB. Ορίζουμε ότι βρόχος for συνεχίζει να εκτελείται όσο η τιμή του doc δεν είναι null. Όταν δεν υπάρχουν άλλα έγγραφα στον cursor, cursor.next() επιστρέφει null, και ο βρόχος σταματά.Στο τέλος κάθε επανάληψης, ο κώδικας παίρνει το επόμενο έγγραφο από τον cursor και το αποθηκεύει στη μεταβλητή doc.

```
for (let doc = await cursor.next(); doc != null; doc = await cursor.next()) {
  const freq1 = getFrequency(array1);
  const freq2 = getFrequency(doc.grammata);

  let matchesCondition = true;
```

Σχήμα 5.5.5 ‘Ευρεση συχνοτήτων

Παράδειγμα:

Αν έχουμε το array1 = ['A', 'B', 'Γ'] και doc.grammata = ['A', 'B'], η getFrequency θα επιστρέψει τα εξής:

- freq1 = { 'A': 1, 'B': 1, 'Γ': 1 }
- freq2 = { 'A': 1, 'B': 1 }

Ο κώδικας τότε θα συγκρίνει αυτές τις συχνότητες για να δει αν κάθε χαρακτήρας στο freq2 εμφανίζεται στο freq1 με ίση ή μεγαλύτερη συχνότητα.

Αυτός ο βρόχος είναι σημαντικός για την επεξεργασία και το φιλτράρισμα των εγγράφων που ανακτώνται από τη βάση δεδομένων σύμφωνα με τις καθορισμένες συνθήκες.

Στη συνέχεια, αναπτύσσεται κώδικας που συγκρίνει τους χαρακτήρες και τις συχνότητές τους στα αντικείμενα freq1 και freq2.Αν οποιοσδήποτε χαρακτήρας στο freq2 δεν υπάρχει στο freq1, ή αν η συχνότητα οποιουδήποτε χαρακτήρα στο freq2 είναι μεγαλύτερη από αυτήν στο freq1, τότε το έγγραφο doc δεν πληροί τις συνθήκες και το matchesCondition τίθεται σε false.Αν το matchesCondition γίνει false, ο βρόχος σταματά να εκτελείται και ο κώδικας έξω από τον βρόχο θα αγνοήσει το τρέχον έγγραφο doc.Αυτός ο έλεγχος εξασφαλίζει ότι το έγγραφο doc πληροί τις απαιτήσεις βάσει των συχνοτήτων των χαρακτήρων, δηλαδή οι χαρακτήρες του doc.grammata υπάρχουν στο array1 με ίση ή μεγαλύτερη συχνότητα.

```
for (let char in freq2) {
  if (!freq1[char] || freq2[char] > freq1[char]) {
    matchesCondition = false;
    break;
  }
}
```

Σχήμα 5.5.6 'Ελεγχος εφαρμογής κανόνων από κάθε υποψήφια λέξη

Τέλος, ο κώδικας ελέγχει αν κάθε έγγραφο doc πληροί τις συνθήκες (δηλαδή, αν το matchesCondition είναι true) και αν η λέξη doc.lexi δεν υπάρχει ήδη στο σύνολο uniqueWords.Αν οι συνθήκες πληρούνται, το έγγραφο προστίθεται στη λίστα matchingEntries και η λέξη του προστίθεται στο σύνολο uniqueWords. Σε περίπτωση που η λίστα matchingEntries φτάσει τα 25 έγγραφα, ο βρόχος διακόπτεται και σταματά η προσθήκη άλλων εγγράφων και επιστρέφονται τα έγγραφα. Αυτός ο κώδικας διασφαλίζει ότι η απόκριση περιέχει έως και 25 μοναδικά έγγραφα που πληρούν τις καθορισμένες συνθήκες.

```
  if (matchesCondition && !uniqueWords.has(doc.lexi)) {
    matchingEntries.push(doc);
    uniqueWords.add(doc.lexi);
    if (matchingEntries.length >= 10) {
      break;
    }
  }
}
res.json(matchingEntries);
```

Σχήμα 5.5.6 Εισαγωγή πρώτων 25 λέξεων που πληρούν τους κανόνες

Εκτός από τις λέξεις που θα επιστραφούν από το server για τη δηιουργία του σταυρολέξου,αναπτύχθηκε και κώδικας επιστροφής όλων των λέξεων που θα μπορούσαν να ταιριάζουν με τα γράμματα που δημιουργήθηκαν, δηλαδή όλες οι λέξεις που μπορούν να σχηματιστούν με τα διαθέσιμα γράμματα. Αυτή η διαφοροποίηση έγινε για λόγους ευελιξίας,λόγω του ότι περιορίζοντας τον αριθμό των λέξεων στο σχηματισμό των σταυρολέξων η δημιουργία και εμφάνιση αυτών γίνεται σε μικρότερο χρονικό διάστημα, αλλά και απλούστευσης του κώδικα και αποφυγής πιθανών σφαλμάτων, αφού περιορίζεται το μέγεθος των ελέγχων και των συνθηκών. Ο αριθμός των λέξεων για σχηματισμό σταυρολέξων (υπενθυμίζεται ότι επιστρέφονται 25 λέξεις οι οποίες θα ελεγχτούν και θα χρησιμοποιηθούν στον CrossWordArray στη δόμηση του κάθε σταυρολέξου) είναι κάτι παραπάνω από επαρκής. ‘Ολες αυτές οι λέξεις που ταιριάζουν με τα γράμματα θα προστεθούν σε ένα component ονόματι AllWords ως έξτρα λειτουργία προς το χρήστη.

Για την επιστροφή όλων αυτών των λέξεων χρειάζονται μικρές τροποποιήσεις στον κώδικα που έχει ήδη γραφτεί για την επιστροφή των λέξεων που μπορούν να σχηματίσουν το σταυρόλεξο. Στο αρχείο Staurolexo προστίθεται μια function fetchAllWords η οποία έχει ακριβώς την ίδια λειτουργικότητα και δομή με την fetchData. Η διαφορά είναι ότι θα χρυπήσει ένα διαφορετικό point στο server,αυτό του allwords.

```
const fetchAllWords = async (wordswipeLetters, source) => {
  try {
    const response = await axios.post('http://localhost:3001/allwords', { array1: wordswipeLetters }, { cancelToken: source.token });
    setMatchingEntries(response.data);
  } catch (error) {
    if (axios.isCancel(error)) {
      console.log('Request canceled', error.message);
    } else {
      console.error('Error fetching data:', error);
    }
  }
};
```

Σχήμα 5.5.6 Επιστροφή των matchingEntries

Ακολουθώντας την ίδια λογική,παραλλάσεται το useEffect,ώστε να επιτρέπεται στη σελίδα να λάβει και όλα τα αποτελέσματα του fetchAllWords.

```
useEffect(() => {
  const source = axios.CancelToken.source();
  if (wordswipeLetters.length > 0) {
    fetchData(wordswipeLetters, source);
    fetchAllWords(wordswipeLetters, source);
  }

  return () => {
    source.cancel('Component unmounted, canceling request');
  };
}, [wordswipeLetters]);
```

Σχήμα 5.5.7 Παράλλαξη useEffect

Τέλος,δημιουργείται στο server WordsServer ένα entry point app.post(‘/allwords’), το οποίο ακολουθεί κατά γράμμα τη δομή και λειτουργικότητα του entry point app.post(‘/words’) με τη μόνη διαφορά την αφαίρεση του κώδικα παύσης αναζήτησης λέξεων αν αυτές είναι πάνω από 25. ‘Ετσι, ο κώδικας παραλλάσεται με τον κατώθι τρόπο:

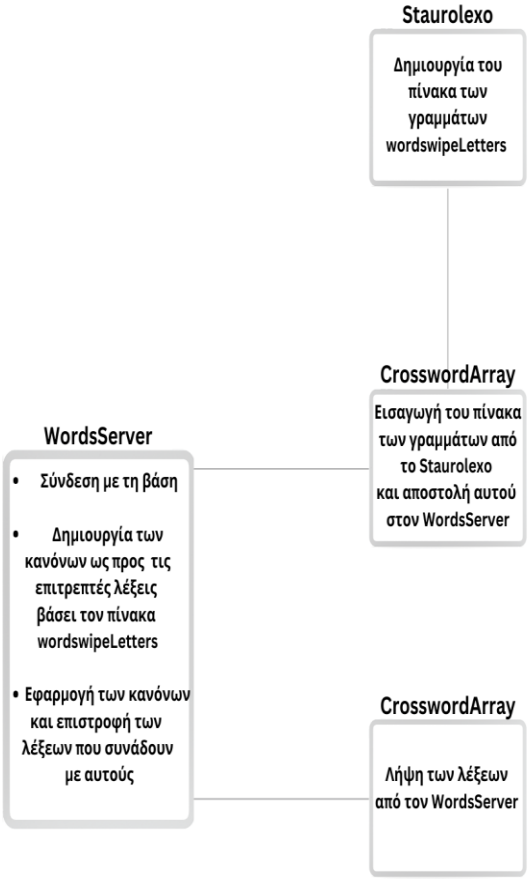
```
if (matchesCondition && !uniqueWords.has(doc.lexi)) {
  matchingEntries.push(doc);
  uniqueWords.add(doc.lexi);
  if (matchingEntries.length >= 25) {
    break;
  }
}
```

```
if (matchesCondition && !uniqueWords.has(doc.lexi)) {
  matchingEntries.push(doc);
  uniqueWords.add(doc.lexi);
}
```

Σχήμα 5.5.5 Διαφορές στη δόμηση κώδικα επιστροφής 25 λέξεων με κώδικα επιστροφής όλων των λέξεων

5.6 Σχηματική Αναπαράσταση Συσχέτισης Αρχείων

Ακολουθεί μία σχηματική αναπαράσταση η οποία επιδεικνύει,με απλουστευμένη λογική, τη συσχέτιση μεταξύ των αρχείων Staurolexo, CrosswordArray και WordServer, στην επιστροφή λέξεων καταλλήλων για τη σύνταξη ενος σταυρόλεξου.



Σχήμα 5.5.6 Σχηματική αναπαράσταση επικοινωνίας των αρχείων Staurolexo, CrosswordArray, WordServer

6. WordSwipe

Το component WordSwipe είναι το component αρμόδιο για τη δημιουργία του πλαισίου της επιλογής των γραμμάτων από το χρήστη, ώστε να δημιουργούνται λέξεις. Πρόκειται για ένα κυκλικό πλαίσιο που μέσα περιέχει όλα τα γράμματα που δημιουργούνται για κάθε επίπεδο και περνάει τους συνδιασμούς των γραμμάτων που εμφανίζει ο χρήστης στο σταυρόλεξο, ώστε, αν οι συνδιασμοί αυτοί υπάρχουν σαν λέξεις στο σταυρόλεξο, να εμφανίζονται.

6.1 User Interface του WordSwipe

Για UI του wordSwipe, δημιουργήθηκε η CSS class ‘wordSwipe’. Κατ’αρχήν, για την ομοιομορφία του πλαισίου με την υπόλοιπη σελίδα, του ορίζουμε ένα μαύρο χρώμα (background-color: black;) και ένα σταθερό περίγραμμα, συγκεκριμένου πλάτους και χρώματος (border-style: solid; border-width:

5px; border-color: #ffbe00;) Η κλάση αυτή ορίζει τόσο το ύψος και το πλάτος του πλαισίου, αλλά το μετατρέπει και σε κύκλο, μέσω του ‘border-radius : 50%’.

Με το inline-flex, το οποίο του δίνει τις ιδιότητες ενός inline στοιχείου να τοποθετείται στη γραμμή με το περιεχόμενο γύρω του και να μην ξεκινά σε μια νέα γραμμή, όπως θα έκανε ένα block στοιχείο, αλλά και τις δυνατότητες ενός flex container, το οποίο επιτρέπει την ελεύθερη διαχείριση και στοίχιση των παιδιών του στοιχείου.

```css
.wordSwipe {
  height: 10em;
  width: 10em;
  border-radius: 50%;
  position: relative;
  display: inline-flex;
  justify-content: center;
  background-color: black;
  opacity: 0.95;
  border-style: solid;
  border-width: 5px;
  border-color: #ffbe00;
  position: relative;
  overflow: hidden;
}
```

Σχήμα 6.1.1 CSS πλαισίου σχηματισμού συνδιασμών γραμμάτων

Παιδιά αυτού του div θα είναι τα γράμματα. Στην αισθητική προσέγγιση των γραμμάτων, θα εμφανίζομται μέσα σε ένα κυκλικό κίτρινο πλαίσιο (border-radius: 50%;background-color: #ffbe00;) χωρίς πλαίσιο (border: none;), με το περιεχόμενό τους, το οποίο και είναι το γράμμα, να είναι σε μαύρο χρώμα (color: black;).

```css
.letter {
  margin: 0.5em;
  border-radius: 50%;
  border: none;
  width: 1.3em;
  height: 1.3em;
  color: black;
  background-color: #ffbe00;
}
```

Σχήμα 6.1.2 CSS γραμμάτων του πλαισίου WordSwipe

Ας δούμε πως καταφέραμε όμως να ενσωματώσουμε τα γράμματα που δημιουργήθηκαν αυτόματα στο πλαίσιο του wordSwipe, να τα κάνουμε λειτουργικά, να εφαρμόζουμε τους κανόνες που προκύπτουν και να σχηματίζουμε σχεδιασμούς γραμμάτων.

6.2 Σχηματισμός συνδιασμών γραμμάτων

Το παιχνίδι, εξ'ορισμού, έχει συγκεκριμένους κανόνες. Αυτοί είναι:

- Κανένα γράμμα δεν πρέπει να χρησιμοποιηθεί πάνω από μία φορά σε ένα συνδιασμό γραμμάτων
- Ο συνδιασμός γραμμάτων πρέπει να μηδενίζει όταν έχουμε επιλέξει όλα τα διαθέσιμα γράμματα ή έχουμε βγάλει το focus από το πλαίσιο επιλογής γραμμάτων (να έχουμε βγάλει τον κέρσορα αν χρησιμοποιούμε λειτουργικό σύστημα Windows ή το δάχτυλο αν χρησιμοποιούμε λειτουργικό σύστημα Android από το πλαίσιο επιλογής)

Στον πίνακα των γραμμάτων που δημιουργούνται αυτόματα κατά την ένταξη στο επίπεδο μπορούν να δημιουργηθούν επαναλήψεις του ίδιου γράμματος. Για παράδειγμα, μπορεί τα γράμματα που δημιουργούνται αυτόματα να είναι τα 'Α','Ε','Α','Ρ','Σ'. 'Οπως βλέπουμε υπάρχει επανάληψη του γράμματος 'Α'. Αυτό σημαίνει πως για να δημιουργήσουμε ένα συνδιασμό γραμμάτων που περιέχει 2 'Α', δεν πρέπει να χρησιμοποιήσουμε 2 φορές το ίδιο 'Α', αλλά μία φορά το πρώτο και μία φορά το δεύτερο.

Γι' αυτό το λόγο δεν πρέπει να χρησιμοποιούμε το value των γραμμάτων (το ίδιο το γράμμα) σαν μεταβλητή στον κώδικα, αλλά να ορίσουμε ένα μοναδικό κλειδί (key) στο κάθε γράμμα, ούτως ώστε να χρησιμοποιούμε κάθε φορά. Ορίζοντας ένα μοναδικό κλειδί μπορούμε να μετατρέψουμε το παράδειγμα των δημιουργημένων γραμμάτων από [Α','Ε','Α','Ρ','Σ'], σε έναν πίνακα που να περιέχει και το γράμμα και τον αύξων αριθμό του γράμματος σαν key. 'Ετσι, ορίζουμε μία καινούργια μεταβλητή, τον arrayLettersAndKeys, και τον κάνουμε populate με ένα object για κάθε γράμμα. Το object αυτό περιλαμβάνει δύο τιμές, το letterKey, που είναι ο αύξων αριθμός του γράμματος στον πίνακα των δημιουργημένων γραμμάτων και το letterValue, που είναι το γράμμα το ίδιο.

```
for (const key of iterator) {

  arrayLettersAndKeys.push( {letterKey: key, letterValue: filteredLettersArray[key]} );

}
```

Σχήμα 6.2.1 Διαχωρισμός γραμμάτων με value, key

Στο παράδειγμα δηλαδή των γραμμάτων [Α','Ε','Α','Ρ','Σ'], η μεταβλητή arrayLettersAndKeys θα έχει την παρακάτω μορφή:

[{letterKey: 0,letterValue: A},

{letterKey: 1, letterValue: E},

{letterKey: 2, letterValue: A,},

{letterKey: 3, letterValue: P,},

{letterKey: 4, letterValue: Σ}]

Μετά, φτιάξαμε μία function createLetterSequence. Η function αυτή ελέγχει αν υπάρχει το key του γράμματος ήδη στο letterKeySequence. Αν δεν υπάρχει, ενημερώνει μία μεταβλητή που ονομάζεται clickedLetter με το value του γράμματος. Έτσι, μπορούμε να δημιουργήσουμε συνδιασμούς γραμμάτων, χωρίς να επαναχρησιμοποιούμε τα ίδια keys.

```
function createLetterSequence(element) {
  let letterKey = element["letterKey"];
  let letterValue= element["letterValue"];
  console.log(letterKey + "letter key" + letterValue + "letter value" + element + "element");

  if (!letterKeySequence.includes(letterKey)) {



    setClickedLetters(clickedLetters => {
     clickedLetters = [...clickedLetters,letterValue]
     console.log(clickedLetters + "clicked letters with added new letter")
     return clickedLetters;
    })
```

Σχήμα 6.2.2 Η function createLetterSequence

Ο συνδιασμός των γραμμάτων θα πρέπει να μηδενίζει όταν έχουμε χρησιμοποιήσει όλα τα διαθέσιμα γράμματα. Αυτό επιτυγχάνεται με μία απλή σύγκριση του clickedLetters (του πίνακα με τα γράμματα που έχουμε συνδέσει) και του filteredLettersArray (του πίνακα με τα γράμματα προς χρήση στο επίπεδο). Αν το μέγεθος του συνδιασμού που φτιάξαμε είναι μεγαλύτερο ή ίσο του αριθμού των διαθέσιμων γραμμάτων, ο συνδιασμός μηδενίζει.

```
if (clickedLetters.length >= filteredLettersArray.length) {
 clickedLetters.length = 0;
 letterKeySequence.length = 0;


 return clickedLetters;
}
```

Σχήμα 6.2.3 Μηδενισμός γραμμάτων συνδιασμού

Εκτός από τον μηδενισμό του συνδιασμού όταν χρησιμοποιούμε όλα τα διαθέσιμα γράμματα, θέλουμε ο συνδιασμός να μηδενίζει αυτομάτως όταν φεύγει το focus από το πλαίσιο, όπως έχουμε ορίσει στο τρίτο κανόνα. Για να το επιτύχουμε αυτό, ορίζουμε μία function clearTypingHandler, η οποία μηδενίζει τα clickedLetters (δηλαδή το συνδιασμό που έχουμε δημιουργήσει).

```
function clearTypingHandler() {

    setClickedLetters(clickedLetters => {
      clickedLetters = [];
      return clickedLetters
    },
    setLetterKeySequence(letterKeySequence =>
      {
        letterKeySequence = [];
        return letterKeySequence;
      }
    )

    )

}
```

Σχήμα 6.2.4 Η function clearTypingHandler υπεύθυνη για το μηδενισμό συνδιασμού

Και τη βάζουμε να τρέχει όταν φεύγει το focus από το WordSwipe (onMouseLeave).

```
<div className="wordSwipe" clickedLetters={clickedLetters} onMouseLeave={() => clearTypingHandler()}>
```

Σχήμα 6.2.5 Χρήση της clearTypingHandler onMouseLeave

Μαζί με το component WordSwipe, ορίζουμε ένα υπο-component που ονομάζεται ShownWord. Αυτό το component παίρνει σαν prop την τιμή του clickedLetters και, αν αυτή είναι 0, δηλαδή δεν υπάρχει συνδιασμός γραμμάτων, εμφανίζει την πρόταση ‘Σχηματίστε μία καινούργια λέξη’. Αν αυτή δεν είναι 0, δηλαδή αν έχουμε σχηματίσει κάποιο συνδιασμό γραμμάτων, το ShownWord εμφανίζει την τιμή του clickedLetters, που είναι ο ίδιος ο συνδιασμός.

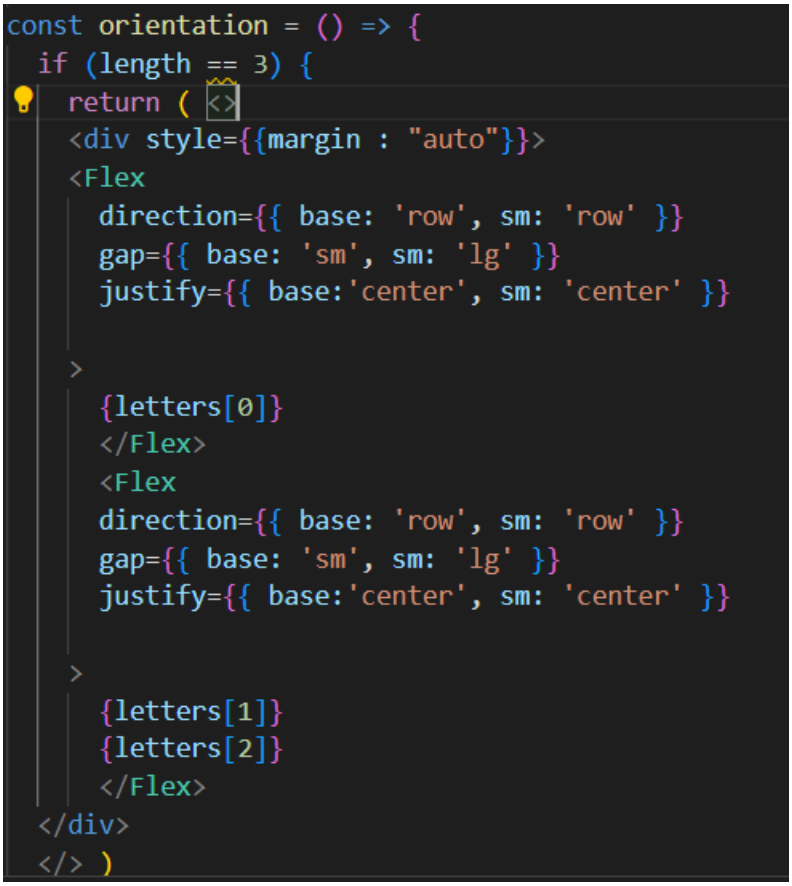
```
export default function ShownWord({clickedLetters}) {
    return (
     <div style={{margin : "1em"}}>
     {clickedLetters.length !== 0 ? <div>

         {clickedLetters}

       </div>


         : <div>Σχηματίστε μία καινούργια λέξη</div>
       }
       </div>
    )
}
```

Σχήμα 6.2.6 ShownWord

Η λειτουργικότητα και το θέμα του WordSwipe είναι έτοιμα.Το μόνο που απομένει είναι η δημιουργία ενός structure για τα γράμματα στο πλαίσιο WordSwipe, ώστε αυτά να παρουσιάζονται με μία όμορφη και ευδιάκριτη δομή.

Για τη δημιουργία συγκεκριμένων θέσεων που μπορούν να απικοίσουν τα γράμματα, δημιουργούμε μία function που ονομάζεται orientation. Η function orientation δέχεται τα γράμματα που δημιουργήθηκαν (filteredLettersArray) και το μέγεθος του πινάκου των γραμμάτων (length). Με αυτά τα δύο στοιχεία, δημιουργεί συγκεκριμένες θέσεις για το κάθε γράμμα αναλόγως του επιπέδου, μέσω του component Flex του Mantine, το οποίο και υιοθετεί τη χρήση του flexbox της CSS: δημιουργεί ένα container και προσδιορίζει τις θέσεις που θα πάρουν όλα τα στοιχεία του μέσα σε αυτό. Για κάθε διαφορετικό length, δηλαδή για κάθε διαφορετικό level, φτιάξαμε και ένα διαφορετικό structure για το orientation.

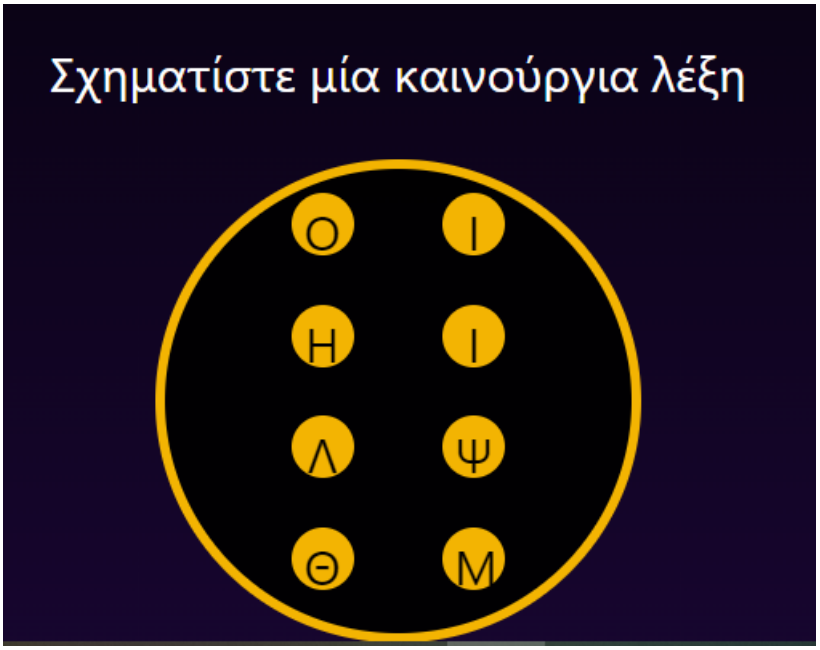
Στο παράδειγμα που ακολουθεί παρουσιάζεται το structure της ανάπτυξης του πρώτου level. Το πρώτο γραμμα τοποθετείται στο κέντρο του WordSwipe και τα δύο επόμενα γράμματα στην επόμενη γραμμή, το ένα στα αριστερά και το άλλο στα δεξιά, συνθέτοντας ένα παραστατικό τρίγωνο.



```
const orientation = () => {
  if (length == 3) {
    return ( <>
    <div style={{margin : "auto"}}>
    <Flex
      direction={{ base: 'row', sm: 'row' }}
      gap={{ base: 'sm', sm: 'lg' }}
      justify={{ base:'center', sm: 'center' }}

    >
      {letters[0]}
      </Flex>
      <Flex
      direction={{ base: 'row', sm: 'row' }}
      gap={{ base: 'sm', sm: 'lg' }}
      justify={{ base:'center', sm: 'center' }}

    >
      {letters[1]}
      {letters[2]}
      </Flex>
  </div>
  </> )
```

Σχήμα 6.2.7 UI εμφάνισης γραμμάτων αναλόγως του επιπέδου

Η διαδικασία του WordSwipe έχει ολοκληρωθεί.



Σχήμα 6.2.8 Τελικό πλαίσιο WordSwipe

7. CrossWord Array

Μετά την ολοκλήρωση του πλαισίου WordSwipe για τη δημιουργία λέξεων, αναλαμβάνει ρόλο στην υλοποίηση η ανάπτυξη του crossword. Το αρχείο CrosswordArray πρόκειται ουσιαστικά για την αναπαράσταση του σταυρολέξου και η απόδοση αυτού στο χρήστη. Μέσα σε αυτό το αρχείο πραγματοποιείται τόσο το στήσιμο του σταυρολέξου, όσο και η επίλυσή του, δεχόμενο το input από το χρήστη. Το πρώτο όμως βήμα είναι η επιβολή κανόνων για το στήσιμο του περιβάλλοντος δημιουργίας και εμφάνισης του σταυρολέξου, όπως και το χτίσιμο ενός κενού πίνακα, ο οποίος θα είναι αρμόδιος για την αναπαράσταση των δεδομένων του σταυρολέξου.

7.1 Στήσιμο του πίνακα του Σταυρολέξου

Για να στηθεί ο array, πρέπει πρώτα να λάβουμε τις λέξεις που ταιριάζουν με τα γράμματα. Υπενθυμίζεται ότι τα γράμματα δημιουργούνται αυτόματα κατά την επιλογή ενός επιπέδου και δίνονται σαν props τόσο στο WordSwipe, το πλαίσιο επιλογής συνδιασμών γραμμάτων, όσο και στο parent component που ονομάζεται Staurolexo. Το Staurolexo δίνει σαν props αυτά τα γράμματα αλλά και τα matchingEntries, τις λέξεις που επέστρεψε ο server, οι οποίες και αποτελούνται από αυτά τα γράμματα. Έτσι, στο αρχείο CrosswordArray,χτίζεται μία επαναλαμβανόμενη function useEffect, η οποία αναμένει τη λήψη των λέξεων του σταυρολέξου (matchingEntries) από το server. Μόλις ληφθούν αυτά τα matchingEntries, γίνεται ένα sort από το μεγαλύτερο στο μικρότερο. Συγκεκριμένα, ελέγχονται ένα προς ένα τα elements που αποτελούν τον array MatchingEntries (τον πίνακα με τις λέξεις που θα μπορούν να αποικίσουν το σταυρόλεξο) και κατανέμονται από το μεγαλύτερο στο

μικρότερο, ανάλογα με το length που έχει το κάθε στοιχείο,που είναι το μέγεθος της κάθε λέξης. Ο σκοπός πίσω από αυτό το sort είναι να διαβαθμίζονται οι λέξεις που μπαίνουν στο σταυρόλεξο από τη μεγαλύτερη στη μικρότερη, καθώς, όσο περισσότερο μέγεθος έχουν οι λέξεις, τόσο ανεβαίνει η πολυπλοκότητα και η δυσκολία επίλυσης του σταυρολέξου.

```
useEffect(() => {
  setResults(wordsGrammata);
  console.log(arrayWithWords + "ARRAY WITH WORDS");
}, [matchingEntries, wordsGrammata]);

const arrayGrammata = matchingEntries.map((entry) => entry.grammata);
arrayGrammata.sort((a, b) => b.length - a.length);
```

Σχήμα 7.1.1 Λήψη λέξεων στο CrosswordArray και σορτάρισμά τους από τη μεγαλύτερη στη μικρότερη

‘Επειτα, αφού έχουν ληφθεί και κατανεμηθεί τα matchingEntries από το μεγαλύτερο στο μικρότερο, ξεκινά η διαδικασία χτίσης του σταυρολέξου. Το σταυρόλεξο πρόκειται ουσιαστικά για έναν δισδιάστατο πίνακα με 10 στήλες (columns),τις κάθετες σειρές του πίνακα και 10 γραμμές (rows),τις οριζόντιες σειρές του πίνακα. ‘Ετσι, δημιουργείται ένας πίνακας με 100 διαθέσιμα κελία, τα οποία περιέχουν το καθένα μία διαφορετική τιμή (i,j) ανάλογα της θέσης του κελιού. Η λογική της δημιουργίας του πίνακα με τα i,j ως κελιά είναι το ότι το κάθε κελί είναι διαθέσιμο προς κατοίκιση από ένα γράμμα μίας λέξης.

Ως προς την πλευρά της υλοποίησης μέσω κώδικα αυτής της προσέγγισης, δημιουργήθηκε μία if που θα ελέγχει αν το μέγεθος των λέξεων είναι μεγαλύτερος από το 0. Αυτό μας επιτρέπει το χτίσιμο του πίνακα μόνο από τη στιγμή που έχουν ληφθεί οι λέξεις (matchingEntries > 0). ‘Ετσι, δεν θα παρουσιαστεί ποτέ στο χρήστη το ενδεχόμενο της εμφάνισης ενός κενού σταυρολέξου προς επίλυση. ‘Επειτα, δημιουργούμε ένα πίνακα με μήκος rows (που είναι 10), όπου όλα τα στοιχεία είναι null (κενά). Με το ‘map((_, rowIndex) =>’ παίρνουμε αυτά τα κενά στοιχεία και δημιουργούμε ένα πίνακα, κρατόντας το rowIndex,δηλαδή την τιμή του row για κάθε στοιχείο. Μετά, μέσω του ‘new Array(columns).fill(null).map((_, colIndex) => (’, δημιουργούμε έναν νέο πίνακα με κενά (null) στοιχεία και, με το map, για κάθε στοιχείο αυτού του πίνακα, κρατάμε το colIndex, που είναι η δεύτερη μεταβλητή που θα δωθεί σαν παράμετρος για την τιμή του κάθε κελιού τουδισδιάστατου πίνακα, αφού, όπως αναφέρθηκε παραπάνω, το κάθε κελί θα έχει συντεταγμένες (rowIndex,colIndex). Τέλος, δημιουργούμε μία <p>, κατ’ουσίαν μία παράγραφο της jsx, για κάθε κελί, δίνοντας σαν key(κλειδί),id (μοναδικό αναγνωριστικό που αποδίδεται σε ένα HTML στοιχείο) και περιεχόμενο τα (rowIndex, colIndex) κάθε κελιού.

```
useEffect(() => {
  if (arrayGrammata.length > 0) {

    const initialArray = new Array(rows).fill(null).map((_, rowIndex) =>
      new Array(columns).fill(null).map((_, colIndex) => (
        <p key={`${rowIndex}-${colIndex}`} id={`${rowIndex}-${colIndex}`}>{rowIndex}, {colIndex}</p>
      ))
    );
```

Σχήμα 7.1.2 Στήσιμο array

Σε οπτική μορφή, αυτό που έχει δημιουργηθεί είναι ο δισδιάστατος πίνακας, αποικημένος με τα rowIndex-colIndex κάθε κελιού, όπως παρουσιάζεται στο παρακάτω σχήμα.

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

Σχήμα 7.1.3 Παράδειγμα στημένου array

7.2 Τοποθέτηση πρώτης λέξης στο σταυρόλεξο

Αφού δημιουργήθηκε ο δισδιάστατος πίνακας 10x10, ήρθε η ώρα αποίκισής του με τις λέξεις. Η διαδικασία που ακολουθείται είναι γραμμική. Λέγοντας γραμμική, εννοούμε πως πρώτα τοποθετείται η πρώτη λέξη και μετά οι υπόλοιπες λέξεις ελέγχονται μία προς μία για το αν ικανοποιούν τους κανόνες της ένταξης στο σταυρόλεξο. Η πρώτη λέξη τοποθετείται πάνω πάνω στον πίνακα, οριζόντια, ξεκινώντας από τα αριστερά και αναπτίζεται στον άξονα x. Πιό απλά, το πρώτο κελί της πρώτης λέξης είναι πάντα το (0,0) και, ανάλογα με τα γράμματα, τόσο αυξάνεται το colIndex. Ακολουθεί ένα παραστατικό παράδειγμα όπου με τη τυχαία περίπτωση που η πρώτη λέξη του sortedArray των λέξεων είναι η λέξη ‘ΛΕΞΟΜΑΓΕΙΑ’, επιδεικνύεται παραστατικά η εναπόθεσή της στον πίνακα.

Λ	Ε	Ξ	Ο	Μ	Α	Γ	Ε	Ι	Α
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

Σχήμα 7.2.1 Παράδειγμα με τοποθετημένη την πρώτη λέξη

Με την εναπόθεση της μεγαλύτερης λέξης πρώτη στο σταυρόλεξο έχουμε τα εξής προτερήματα:

- Δημιουργούμε ένα βασικό σκελετό για το σταυρόλεξο. Αυτό μας βοηθά στην ανάπτυξη του κώδικα στησίματος του, αφού πολύ εύκολα μπορούν να παρατηρηθούν λογικά σφάλματα (σφάλματα στην υλοποίηση της προγραμματισμένης λογικής του προγραμματιστή, το οποίο οδηγεί σε λάθος αποτελέσματα κατά την εκτέλεση του προγράμματος. Το πρόγραμμα μεταγλωττίζεται και εκτελείται χωρίς σφάλματα ή προβλήματα, αλλά η έξοδος δεν είναι αυτή που αναμένενει ο προγραμματιστής. [45])
- Μπορούμε να προσδιορίσουμε καλύτερα τα σημεία όπου οι άλλες λέξεις μπορούν να εγκατασταθούν χωρίς να προκαλέσουν αλληλοεπικαλύψεις ή προβλήματα.
- Η εναπόθεση της μεγαλύτερης λέξης πρώτης μειώνει τις πιθανότητες σύγκρουσης με άλλες λέξεις. Εάν μια μεγαλύτερη λέξη έχει εισαχθεί στον πίνακα, οι μικρότερες λέξεις θα βρουν πιο εύκολα τη θέση τους γύρω από αυτήν.
- Η μεγαλύτερη λέξη συχνά καταλαμβάνει αρκετό χώρο, αφήνοντας λιγότερες επιλογές για τις υπόλοιπες λέξεις. Με την εισαγωγή της ως πρώτη λέξη στον πίνακα, μπορούμε να αποφύγουμε τη δημιουργία ανεπιθύμητων περιορισμών για τις υπόλοιπες λέξεις. Οι υπόλοιπες λέξεις έχουν μεγάλο περιθώριο να αναπτυχθούν, αφού ολόκληρος ο πίνακας πέρα από την πρώτη γραμμή διατείθεται κενός.

Για την εναπόθεση της πρώτης λέξης στον πίνακα, αρχικά ορίζεται μία μεταβλητή initialExecution = useRef(true). Λόγω του ότι χρησιμοποιούμε το επαναλαμβανόμενο hook useEffect(), χρειαζόμαστε μία μεταβλητή που να παίρνει τιμή όταν τοποθετείται η πρώτη λέξη και να μην επιτρέπει την εναπόθεση άλλης λέξης κατά την επανάληψη. Ορίζεται μία for, η οποία, ξεκινώντας με i=0 (που σημαίνει colIndex= 0), αυξάνει το colIndex για κάθε γράμμα της πρώτης λέξης. Καθ’ αυτόν τον τρόπο, δημιουργούμε ένα διαφορετικό colIndex για κάθε γράμμα της πρώτης λέξης και, κατ’επέκταση, για κάθε κελί που θα χρησιμοποιηθεί από αυτήν. Μετά, ενημερώνουμε τον updatedArray. Σαν συντεταγμένες, του ορίζουμε το [0][i], να ξεκινάει δηλαδή η λέξη από το στοιχείο [0,0] του πίνακα και να αναπτύσσεται οριζοντίως. Αφού ενημερώσουμε τα key και id του array με τις τιμές που έχει το κάθε γράμμα σ΄αυτόν, εισχωρούμε τη λέξη στον array. Έπειτα, ενημερώνουμε τη μεταβλητή setLexeisKaiTheseis,ώστε να δέχεται ένα object με το κάθε κελί που χρησιμοποιήθηκε, τη τιμή που περιέχει (το γράμμα της λέξης) και αρχικοποιούμε σ αυτό μια property ονόματι found.Αυτή θα χρησιμοποιηθεί κατά τη διαδικασία επίλυσης του σταυρολέξου, ώστε να μπορεί ο κώδικας να αναγνωρίζει ποιές λέξεις έχει καταφέρει να βρει ο χρήστης. ‘Ετσι, σε αυτό το property αναθέτουμε το

value false, που σημαίνει ότι δεν έχει βρεθεί ακόμα αυτή η λέξη από το χρήστη. Τέλος, μετά το πέρας της εισαγωγής της πρώτης λέξης στον updatedArray και της αποθήκευσης των δεδομένων στη lexeisKaiTheseis, θέτουμε τη μεταβλητή initialExecution.current = false, ώστε να μην μπορεί άλλη λέξη να εισαχθεί σαν πρώτη στον πίνακα.

```
for (let i = 0; i < arrayGrammata[0].length; i++) {
  if (initialExecution.current) {
    console.log(`j: ${0},i: ${i}, γράμμα: ${arrayGrammata[0][i]}`);

    setLexeisKaiTheseis(prevLexeisKaiTheseis => [...prevLexeisKaiTheseis, { thesi: [0, i], gramma: arrayGrammata[0][i], found: false }])
  }

  updatedArray[0][i] = <p key={`0-${i}`} id={`0-${i}`}>{arrayGrammata[0][i]}</p>;
}
initialExecution.current = false;
```

Σχήμα 7.2.2 Τοποθέτηση πρώτης λέξης στον πίνακα

Καθ'αυτόν τον τρόπο, έχουμε εναποθέσει την πρώτη λέξη στον πίνακα updatedArray και έχουμε αποθηκεύσει τις συντεταγμένες της για αποθήκευση στη βάση. Αποθηκεύουμε και την πρώτη λέξη του σταυρολέξου σε μία μεταβλητή lexeisStaurolexou. Με τις lexeisStaurolexou θα διασταυρώνουμε αν κάθε λέξη που θα προσπαθήσουμε στη συνέχεια να εναποθέσουμε στον πίνακα υπάρχει ή όχι ήδη σε αυτόν.

```
if (!lexeisStaurolexou.includes(arrayGrammata[0])) {
  setLexeisStaurolexou(prevLexeisStaurolexou => [...prevLexeisStaurolexou, arrayGrammata[0]]);
}
```

Σχήμα 7.2.3 Αποθήκευση θέσης πρώτης λέξης

7.3 Τοποθέτηση δεύτερης λέξης στο σταυρόλεξο

Αφού βάλαμε την πρώτη,η οποία είναι και η μεγαλύτερη, λέξη του πίνακα arrayGrammata στον πίνακα του σταυρολέξου, θα προσπαθήσουμε να βάλουμε και μία δεύτερη λέξη σε αυτόν. Η δεύτερη λέξη που θα μπει στο σταυρόλεξο πρέπει πρώτα να ικανοποιεί κάποιους κανόνες. Αυτοί είναι:

- Να μην είναι επανάληψη της πρώτης λέξης.
- Να έχει σαν πρώτο γράμμα, δηλαδή να αρχίζει με, ένα γράμμα που έχει και η πρώτη λέξη.

Με λίγα λόγια, όχι μόνο δεν θέλουμε να επαναλαμβάνουμε την ίδια λέξη, αλλά δεν θέλουμε και να εναποθέσουμε στον πίνακα μία λέξη που να έχει απλά ένα γράμμα κοινό με την πρώτη λέξη,αλλά να αρχίζει με ένα κοινό γράμμα. Αυτός ο κανόνας ορίστηκε διότι η δεύτερη λέξη θα τοποθετηθεί καθέτως στην πρώτη. 'Ετσι, αν το κοινό γράμμα που έχουν οι δύο λέξεις δεν είναι το αρχικό της δεύτερης, η δεύτερη λέξη θα βγει εκτός ορίων πίνακα. Ακολουθεί ένα παράδειγμα της λογικής αυτής

που θέλουμε να αποφύγουμε καθώς με την εγκατάσταση της λέξης ‘ΕΛΕΥΘΕΡΙΑ’ στο γράμμα ‘Λ’ της λέξης ‘ΛΕΞΟΜΑΓΕΙΑ’ στον πίνακα, βγαίνουμε εκτός ορίως αυτού.

Ε									
Λ	Ε	Ξ	Ο	Μ	Α	Γ	Ε	Ι	Α
Ε	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
Υ	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
Θ	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
Ε	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
Ρ	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
Ι	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
Α	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

Σχήμα 7.3.1 Εναπόθεση λέξης που δεν εξυπηρετεί τον δεύτερο κανόνα

Αφού έγιναν κατανοητοί οι κανόνες της εγκατάσταση της δεύτερης λέξης, ας δούμε τη διαδικασία που ακολουθήθηκε για να μπει μία δεύτερη λέξη στον πίνακα. Στην αρχή, δηλώνουμε μία μεταβλητή secondWordPlaced= false. Αυτή η μεταβλητή θα πάρει τιμλη true με την εύρεση και εγκατάσταση της δεύτερης λέξης στον πίνακα, δηλαδή μόλις βρεθεί κάποια λέξη που να μπορεί να μπεί κάθετα στην πρώτη λέξη, ώστε να σταματήσει η διαδικασία αναζήτησης δεύτερης λέξης από τη useEffect. ‘Επειτα, δημιουργήσαμε μία for, που κάνει parse κάθε λέξη των matchingEntries, εν προκειμένου του arrayGrammata, για να εξετάσει το ενδεχόμενο να μπορεί η λέξη να οριστεί ως δεύτερη στον array. Η for, φυσικά, πρέπει να ελέγχει όλες τις διαθέσιμες λέξεις πέρα από την πρώτη λέξη που έχει ήδη τοποθετηθεί. Γι’ αυτό το λόγο, ξεκινάμε από τη δεύτερη λέξη την αναζήτηση (let secondLexi = 1). Στη συνέχεια, ορίζουμε δύο μεταβλητές, το grammaFirstLexis, που αφορά το κάθε γράμμα της πρώτης λέξης και το firstLetterOfWord, το οποίο λαμβάνει τη τιμή του πρώτου γράμματος της κάθε λέξης πέρα από την πρώτη.

```
let secondWordPlaced = false;
for (let secondLexi = 1; secondLexi < arrayGrammata.length; secondLexi++) {
  if (secondWordPlaced) break;

  for (let firstLexiCommonIndex = 0; firstLexiCommonIndex < arrayGrammata[0].length; firstLexiCommonIndex++) {
    if (secondWordPlaced) break;

    let grammaFirstLexis = arrayGrammata[0][firstLexiCommonIndex];
    let firstLetterOfWord = arrayGrammata[secondLexi][0];
```

Σχήμα 7.3.2 Εύρεση συντεταγμένων δεύτερης λέξης

‘Οπως γίνεται κατανοητό, αν κάποιο γράμμα της πρώτης λέξης υπάρχει και σαν πρώτο γράμμα μίας οποιασδήποτε άλλης λέξης, έχουμε δεύτερη λέξη στο σταυρόλεξο. Αυτό μπορεί να γίνει πιο

κατανοητό με το σχήμα που ακολουθεί. Στο σχήμα αυτό, η λέξη ‘ΛΕΞΟΜΑΓΕΙΑ’ έχει κοινό γράμμα το δεύτερό της γράμμα, που είναι το ‘Ε’, με το πρώτο γράμμα της λέξης ‘ΕΛΕΥΘΕΡΙΑ’, που είναι επίσης το ‘Ε’.

Λ	Ε	Ξ	Ο	Μ	Α	Γ	Ε	Ι	Α
1,0	Λ	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	Ε	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	Υ	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	Θ	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	Ε	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	Ρ	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	Ι	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	Α	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

Σχήμα 7.3.3 Παράδειγμα εύρεσης και εναπόθεση δεύτερης λέξης με την επιβεβαίωση των κανόνων

Με τη λογική αυτή, δημιουργούμε μια if στον κώδικα, που ελέγχει αν το firstLetterOfWord είναι ίδιο με οποιοδήποτε γράμμα της πρώτης λέξης. Αν είναι ίδιο, έχουμε βρει τη δεύτερη λέξη που θα μπει στο σταυρόλεξο.

```
if (firstLetterOfWord === grammaFirstLexis) {

  let secondWord = arrayGrammata[secondLexi];
```

Σχήμα 7.3.4 ‘Ελεγχος πρώτου γράμματος υποψήφιας λέξης με οποιοδήποτε γράμμα της πρώτης λέξης

Αυτό που απομένει είναι το να βάλουμε τη δεύτερη λέξη στον array. Για να γίνει αυτό, θα πρέπει να αυξάνουμε τον άξονα j ανάλογα με τον αριθμό των γραμμάτων που απαρτίζουν τη δεύτερη λέξη. Σαν i, αναθέτουμε το i του κοινού στοιχείου που έχουν η πρώτη με τη δεύτερη λέξη. Όπως και στην εναπόθεση της πρώτης λέξης, αποθηκεύουμε τα κελιά που απαρτίζουν τη δεύτερη λέξη και τις τιμές αυτών στη lexeisKaiTheseis.

```
for (let j = 0; j < arrayGrammata[secondLexi].length; j++) {
  if (secondExecution.current) {
    console.log(`j: ${j},i: ${firstLexiCommonIndex}, γράμμα: ${arrayGrammata[j][firstLexiCommonIndex]}`);

    setLexeisKaiTheseis(prevLexeisKaiTheseis => [...prevLexeisKaiTheseis, { thesi: [j, firstLexiCommonIndex],
      gramma: arrayGrammata[j][firstLexiCommonIndex] }])
  }

  updatedArray[j][firstLexiCommonIndex] = <p key={`${j}-${firstLexiCommonIndex}`} id={`${j}-${firstLexiCommonIndex}`}>
    {arrayGrammata[secondLexi][j]}</p>;
  secondExecution.current = false
}
```

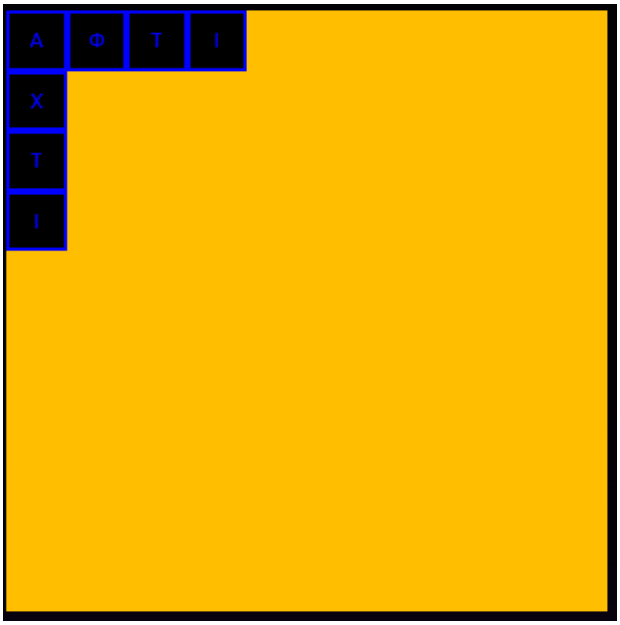
Σχήμα 7.3.5 Τοποθέτηση δεύτερης λέξης

Ενημερώνουμε τις lexeisStraurolexou ώστε η τρίτη λέξη που θα τοποθετήσουμε στον πίνακα να μην είναι ίδια ούτε με την πρώτη, αλλά ούτε με τη δεύτερη. Τέλος, θέτουμε την μεταβλητή secondWordPlaced = true, αποτρέποντας την useEffect να αναζητήσει δεύτερη λέξη και ενημερώνουμε το σταυρόλεξο.

```
if (!lexeisStaurolexou.includes(secondWord)) {
  setLexeisStaurolexou(prevLexeisStaurolexou => [...prevLexeisStaurolexou, secondWord]);
}

secondWordPlaced = true;
console.log(lexeisKaiTheseis + "lexeis klai theseis sti deuteri lexi")
setTwoDimensionalArray(updatedArray);
```

Σχήμα 7.3.6 Ενημέρωση πινάκων ελέγχου με τα στοιχεία της δεύτερης λέξης

Με τη μεθοδολογία που αναλύσαμε, έχουμε καταφέρει να αναπτύξουμε ένα σταυρόλεξο δύο λέξεων.



Σχήμα 7.3.7 Σταυρόλεξο δύο λέξεων

7.4 Τοποθέτηση τρίτης λέξης στο σταυρόλεξο

Η Τρίτη λέξη που θέλουμε να εισάγουμε στο σταυρόλεξο θα εγκαθίσταται και αυτή οριζόντια στη δεύτερη λέξη, όπως και η πρώτη. Θέλουμε, με την είσοδο της τρίτης λέξης, να έχουμε ένα σταυρόλεξο με 2 οριζόντιες και 1 κάθετη λέξη.

Για την επίτευξη της τοποθέτησης και τρίτης λέξης στο σταυρόλεξο, ήρθαμε αντιμέτωποι με τους κανόνες που πρέπει να σεβαστούμε.Αυτοί οι κανόνες είναι:

- Η Τρίτη λέξη δεν μπορεί να μπει στα κελιά που έχει ήδη εισαχθεί η πρώτη λέξη
- Αν το κοινό στοιχείο της τρίτης λέξης με την δεύτερη λέξη δεν είναι το πρώτο γράμμα, πρέπει με αφαιρέσεις στον δείκτη i να βρούμε το κοινό στοιχείο (όπως φαίνεται στο παρακάτω σχήμα, η λέξη 'ΡΕΜΑ' ξεκινά με i=0, κάτι που είναι αποτέλεσμα της αφαίρεσης του κοινού στοιχείου της δεύτερης και τρίτης λέξης, με τη θέση του κοινού γράμματος στη Τρίτη λέξη)

Λ	Ε	Ξ	Ο	Μ	Α	Γ	Ε	Ι	Α
1,0	Λ	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	Ε	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	Υ	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	Θ	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
Ρ	Ε	Μ	Α	5,4	5,5	5,6	5,7	5,8	5,9
6,0	Ρ	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	Ι	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	Α	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

Σχήμα 7.4.1 Τοποθέτηση τρίτης λέξης με σωστές τιμές

'Ετσι, πρέπει να αναπτυχθεί κώδικας που να τοποθετεί μια τρίτη λέξη στο σταυρόλεξο με βάση τη σύνδεση της τρίτης λέξης με τη δεύτερη λέξη, λαμβάνοντας υπόψη την κοινή θέση γραμμάτων. Αν βρει ένα κοινό γράμμα μεταξύ της δεύτερης και της τρίτης λέξης, ο κώδικας πρέπει να υπολογίζει τη σωστή θέση στο πλέγμα και ορίζει την τρίτη λέξη εκεί.

Για την επιβεβαίωση αυτής την σκέψης, όπως και των κανόνων που θέσαμε, δημιουργήσαμε ξανά μία μεταβλητή που να έχει τη τιμή false (thirdWordPlaced = false) που, σε περίπτωση που γίνει true, να δείχνει στη useEffect να σταματήσει την αναζήτηση τρίτης λέξης, όπως και μίας επαναληπτικής for για κάθε στοιχείο του arrayGrammata πέρα από το πρώτο. Γίνεται έλεγχος και για το αν η τρίτη λέξη που σκοπεύουμε να τοποθετήσουμε στο σταυρόλεξο έχει ήδη χρησιμοποιηθεί.

```
for (let thirdLexi = 1; thirdLexi < arrayGrammata.length; thirdLexi++) {
    if (thirdWordPlaced) break;
    if (lexeisStaurolexou.includes(arrayGrammata[thirdLexi])) continue;
    if (thirdWordPlaced) break;
```

Σχήμα 7.4.2 'Ελεγχος τρίτης διαθέσιμης λέξης

Κατόπιν, πρέπει να γίνουν κάποιοι ελέγχοι για να δούμε αν μπορούμε να βάλουμε τρίτη λέξη. Θα πρέπει να ελέγξουμε αν η τρίτη λέξη που θέλουμε να εισάγουμε έχει κάποιο κοινό γράμμα με τη δεύτερη λέξη, αλλά και που, ώστε με τις κατάλληλες προσθαφεραίσεις, να βρούμε το σημείο εκκίνησης της τρίτης λέξης στο πίνακα. Την εύρεση του cell εκκίνησης την αναλαμβάνει η μεταβλητή i.

Καταλαβαίνουμε ότι, για την επίτευξη της εισαγωγής της τρίτης λέξης στον array, θα ακολουθηθεί μία διαφορετική διαδρομή από αυτές για τη τοποθετηση της πρώτης και δεύτερης λέξης. Εκτός από μία μεταβλητή i που θα χρησιμοποιηθεί για την εύρεση του σημείου εκκίνησης της τρίτης λέξης, πρέπει και να επιστρατεύσουμε μία έξτρα μεταβλητή, η οποία θα αυξάνεται για κάθε γράμμα της τρίτης λέξης, ώστε να ορίζεται η θέση που θα πάρει το κάθε γρλαμμα της λέξης στον array. Αυτό το ρόλο αναλαμβάνει η μεταβλητή k.

‘Ετσι, σαν thirdWord ορίζουμε την κάθε υποψήφια λέξη από τον πίνακα arrayGrammata και ορίζουμε το βρόχο επανάληψης for.

```
let thirdWord = arrayGrammata[thirdLexi];
for (let i = thirdStartingPoint, k = 0; i < (arrayGrammata[thirdLexi].length + thirdStartingPoint); i++, k++) {
```

Σχήμα 7.4.3 Εύρεση συντεταγμένων για εισαγωγή τρίτης θέσης στο σταυρόλεξο

Στη συνέχεια, όσο η thirdExecution είναι true, κάτι που σημαίνει πως δεν έχει εισαχθεί ακόμα τρίτη λέξη, ενημερώνουμε τη lexeisKaiTheseis με τη νέα λέξη που θα βάλουμε στο σταυρόλεξο.

```
if (thirdExecution.current) {
    console.log(`j: ${secondLexiCommonIndex},i: ${i}, γράμμα: ${arrayGrammata[secondLexiCommonIndex][i]}`);

    setLexeisKaiTheseis(prevLexeisKaiTheseis =>
    [...prevLexeisKaiTheseis, { thesi: [secondLexiCommonIndex, i], gramma: arrayGrammata[secondLexiCommonIndex][i] }])
}
```

Σχήμα 7.4.4 Ενημέρωση lexeisKaiTheseis με τα στοιχεία της τρίτης λέξης

‘Επειτα, ορίζουμε στον array τα κελιά που θα κατοικήσει το κάθε γράμμα της τρίτης λέξης στον πίνακα.

```
updatedArray[secondLexiCommonIndex][i] =
<p key={`${secondLexiCommonIndex}-${i}`}
id={`${secondLexiCommonIndex}-${i}`}>
    {arrayGrammata[thirdLexi][k]}
</p>;
thirdExecution.current = false;
console.log(thirdWord)
```

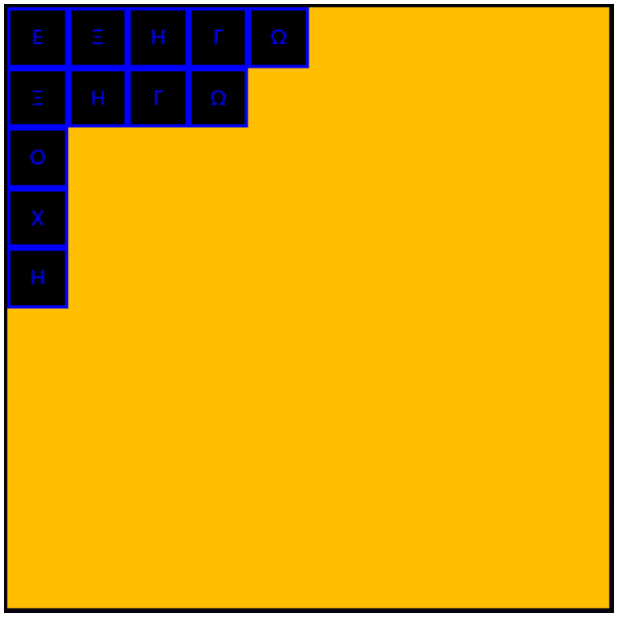
Σχήμα 7.4.5 Ορισμός τρίτης θέσης στον πίνακα

Τέλος, ορίζουμε στη useEffect ότι μπήκε μία τρίτη λέξη (thirdWordPlaced = true) και ενημερώνουμε τον array με τις τελευταίες πληροφορίες.

```
thirdWordPlaced = true;
setTwoDimensionalArray(updatedArray);
```

Σχήμα 7.4.6 Αποθήκευση τρίτης λέξης και αποφυγή επανάληψης αναζήτησής της

‘Ετσι, έχει ολοκληρωθεί και η εισαγωγή μίας τρίτης λέξης στο σταυρόλεξο.



Σχήμα 7.4.7 Σταυρόλεξο τριών λέξεων

7.5 Τοποθέτηση τέταρτης λέξης στο σταυρόλεξο

Αφού βάλαμε τη τρίτη λέξη στο πίνακα, ο κώδικας συνεχίζει στη τοποθέτηση της τέταρτης λέξης. Για να βάλουμε τέταρτη λέξη στον πίνακα, η λογική που θα ακολουθηθεί είναι παρόμοια με τη λογική που ακολουθήθηκε στην εγκατάσταση της τρίτης λέξης. ‘Ετσι, δημιουργούμε μία επαναληπτική for.

```
for (let fourthLexi = 1; fourthLexi < arrayGrammata.length; fourthLexi++) {
    if (fourthWordPlaced) break;
    if (lexeisStaurolexou.includes(arrayGrammata[fourthLexi])) continue;
    if (fourthWordPlaced) break;
```

Σχήμα 7.5.1 Αρχικές κινήσεις εγκατάστασης τέταρτης λέξης

Εξετάζεται κάθε γράμμα της τέταρτης λέξης (fourthLexiLetter) για να δούμε αν ταιριάζει με γράμμα της τρίτης λέξης στη θέση thirdLexiCommonIndex.Ορίζεται μία μεταβλητή metritis, η οποία χρησιμοποιείται για να βρει τη θέση της τέταρτης λέξης σε σχέση με τη θέση της τρίτης λέξης.Εάν το γράμμα της τρίτης λέξης στη θέση thirdLexiCommonIndex ταιριάζει με το γράμμα της τέταρτης λέξης και η θέση είναι έγκυρη (δηλαδή δεν είναι αρνητική και η λέξη ταιριάζει), τότε η τέταρτη λέξη μπορεί να τοποθετηθεί στο grid στην κατάλληλη θέση.

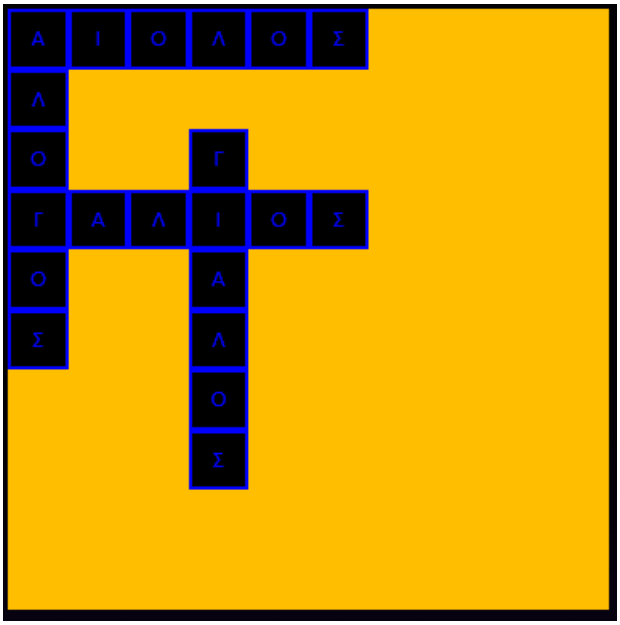
```
for (let thirdLexiCommonIndex = 1; thirdLexiCommonIndex < thirdWord.length; thirdLexiCommonIndex++) {
    if (fourthWordPlaced) break;

    arrayGrammata[fourthLexi].some((fourthLexiLetter) => {
        let metritis = secondLexiCommonIndex - arrayGrammata[fourthLexi].indexOf(fourthLexiLetter);

        if (thirdWord[thirdLexiCommonIndex] === fourthLexiLetter &&
            thirdLexiCommonIndex >= arrayGrammata[fourthLexi].indexOf(fourthLexiLetter) &&
            metritis > -1
```

Σχήμα 7.5.2 Ορισμός των περιορισμών για την εύρεση συντεταγμένων τέταρτης λέξης

‘Επειτα, ορίζονται τα νέα x και y, δηλαδή i,j που θα έχει η τέταρτη λέξη.Μόλις η λέξη μπει στον πίνακα, το grid ενημερώνεται και το σύστημα θεωρεί ότι η τέταρτη λέξη έχει τοποθετηθεί επιτυχώς, σταματώντας περαιτέρω ελέγχους και αναζητήσεις για την ίδια λέξη.

```
if (thirdWord !== fourthWord) {

    let newy = thirdStartingPoint + thirdLexiCommonIndex;
    let newx = secondLexiCommonIndex - arrayGrammata[fourthLexi].indexOf(fourthLexiLetter);
    console.log(newx + " newx " + newy + " newy");
    console.log("-------");

    for (let j = newx, k = 0; j < (arrayGrammata[fourthLexi].length + newx); j++, k++) {
        console.log(arrayGrammata[fourthLexi][0] + ": arrayGrammata[fourthLexi][0} ," + JSON.stringify(updatedArra
        updatedArray[j][newy] = <p key={`${j}-${newy}`} id={`${j}-${newy}`}>{arrayGrammata[fourthLexi][k]}</p>;
        console.log(fourthWord + " Τέταρτη ΛΕΞΗ!!!");
    }

    fourthWordPlaced = true;
    setTwoDimensionalArray(updatedArray);
}
```

Σχήμα 7.5.3 Εγκατάσταση τέταρτης λέξης



Σχήμα 7.5.4 Σταυρόλεξο τεσσάρων λέξεων

7.6 Τοποθέτηση πέμπτης λέξης στο σταυρόλεξο

Με το τρόπο που έχουμε επιδείξει και παραπάνω γίνεται και η εναπόθεση της πέμπτης και τελικής λέξης στο σταυρόλεξο, ξεκινώντας με τον ορισμό των παραμέτρων αναζήτησης.

```
for (let fifthLexi = 1; fifthLexi < arrayGrammata.length; fifthLexi++) {
    if (fifthWordPlaced) break;
    if (lexeisStaurolexou.includes(arrayGrammata[fifthLexi])) continue;
    if (fifthWordPlaced) break;

    for (let fourthWordIndex = 0; fourthWordIndex < arrayGrammata[fourthLexi].length; fourthWordIndex++) {
        if (fifthWordPlaced) break;
```

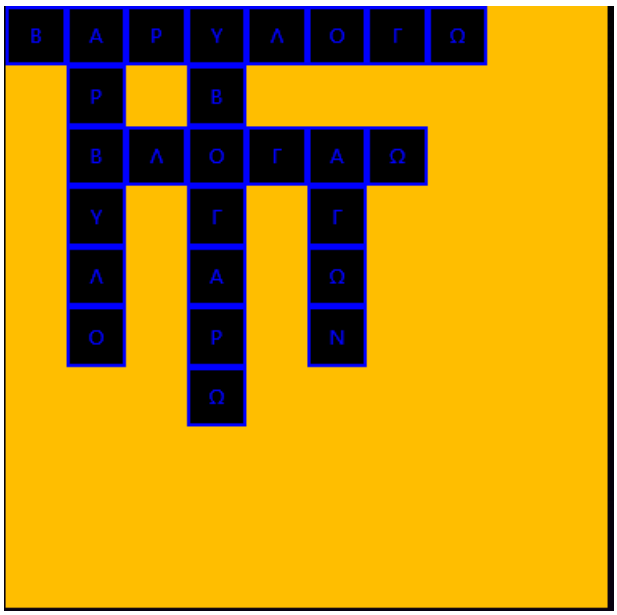
Σχήμα 7.6.1 Αρχικές κινήσεις εγκατάστασης πέμπτης λέξης

‘Επειτα, βρίσκονται οι συντεταγμένες που θα έχει η πέμπτη λέξη, η οποία και θα τοποθετηθεί και αυτή οριζόντια, όπως η πρώτη και η τρίτη λέξη. Βρίσκεται το κοινό σημείο με τη τέταρτη λέξη και η θέση από την οποία θα ξεκινήσει η πέμπτη (δηλαδή το κελί στο οποίο θα τοποθετηθεί το πρώτο γράμμα της πέμπτης λέξης). Με την εύρεση αυτών των στοιχείων, εγκαθίσταται και η πέμπτη λέξη στον πίνακα, ενημερώνεται ο array για την εγκατάστασή της και διατάζεται η useEffect να μην αναζητήσει άλλη λέξη.

```
arrayGrammata[fifthLexi].some((fifthLexiLetter) => {

    let fourthLetter = arrayGrammata[fourthLexi][fourthWordIndex];
    let y = fourthStartingPoint + fourthWordIndex;

    if (fifthLexiLetter === fourthLetter) {
        let fifthWord = arrayGrammata[fifthLexi];
        let fifthStartingPoint = y - arrayGrammata[fifthLexi].indexOf(fifthLexiLetter);

        if (fifthStartingPoint >= 0 && fifthStartingPoint + fifthWord.length <= columns) {
            for (let j = fifthStartingPoint, k = 0; j < (fifthStartingPoint + fifthWord.length); j++, k++) {
                updatedArray[y][j] = <p key={`${y}-${j}`} id={`${y}-${j}`}>{arrayGrammata[fifthLexi][k]}</p>;

            }
            fifthWordPlaced = true;
            setTwoDimensionalArray(updatedArray);
        }
    }
});
```

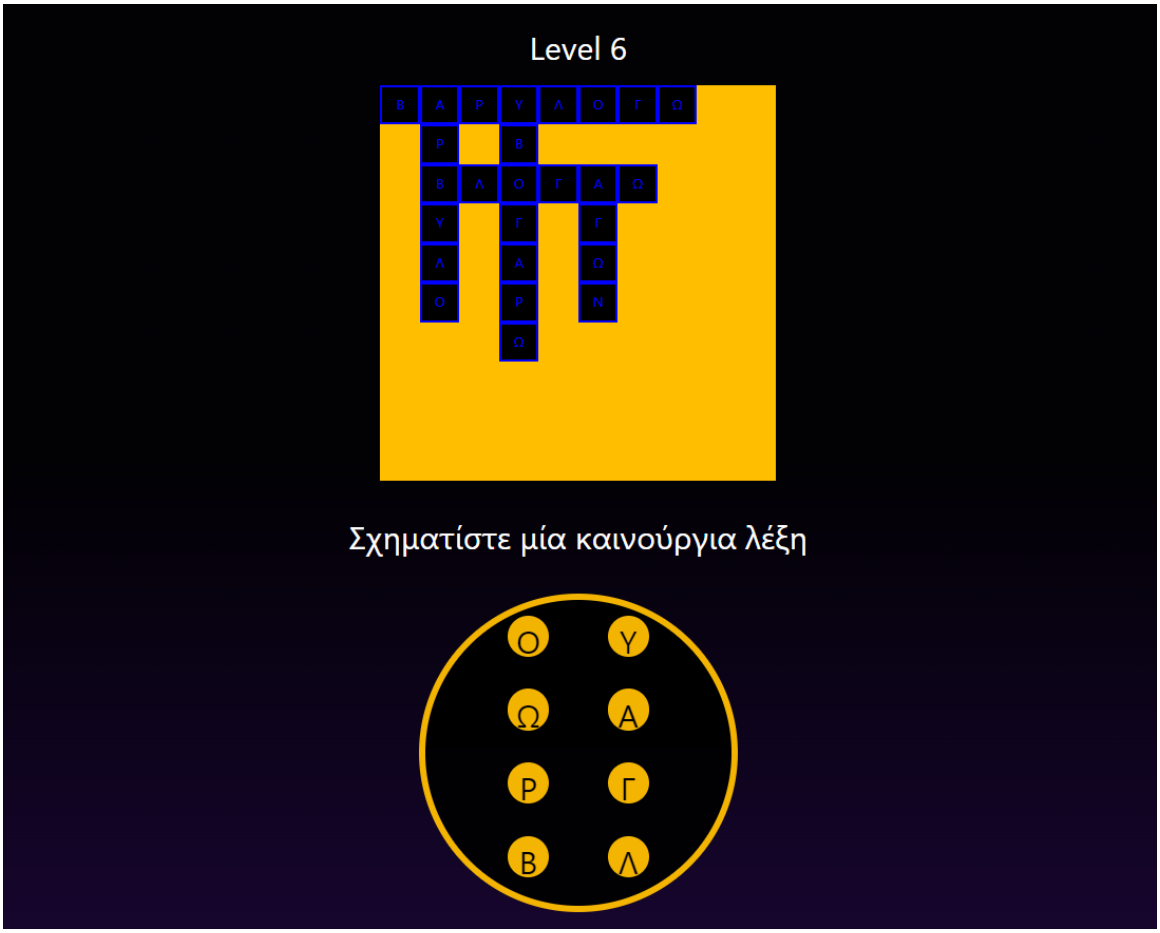
Σχήμα 7.6.2 Εύρεση και εγκατάσταση πέμπτης λέξης

Το στήσιμο του σταυρολέξου με τις λέξεις έχει ολοκληρωθεί. Πλέον, έχει δημιουργηθεί ένας αλγόριθμος ικανός να κάνει populate ένα σταυρόλεξο μέχρι και με 5 λέξεις.



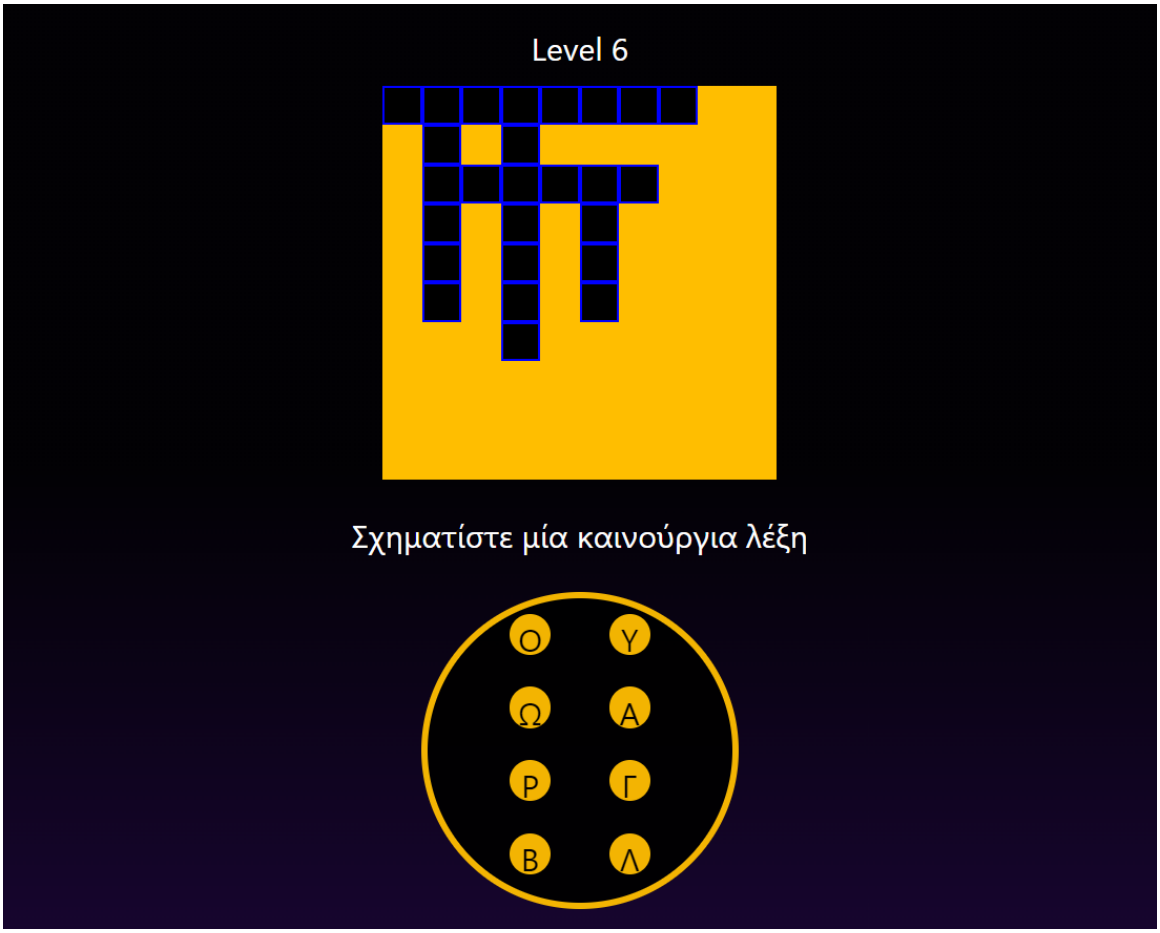
Σχήμα 7.6.3 Σταυρόλεξο πέντε λέξεων

Με την ολοκλήρωση του crosswordArray γίνεται και εισαγωγή του component στο component Staurolexo, μαζί με το πλαίσιο wordSwipe.



Σχήμα 7.6.4 Το component Staurolexo με το πλαίσιο των γραμμάτων και το σταυρόλεξο

Τέλος, θέτουμε σε όλα τα κελιά του πίνακα τη τιμή ‘color: black’. Αυτό θα κάνει το περιεχόμενο του array αόρατο στους users και, αφού ορίζεται πως με την εύρεση κάποιας λέξης να αλλάζει το color σε μπλε, έχουμε την ολοκλήρωση του σταυρολέξου.



Σχήμα 7.6.5 Κρύψιμο των λέξεων

Κεφάλαιο 8ο: Επίλυση σταυρολέξου, All Words και επιλογή Continue

Αφού έχουμε ολοκληρώσει τη διαδικασία εναπόθεσης των λέξεων στο σταυρόλεξο, απομένει η ανάπτυξη του κώδικα που θα ορίζει τον τρόπο που ο χρήστης θα μπορεί να παίζει και να ολοκληρώνει το σταυρόλεξο.

8.1 Εύρεση λέξεων

Για να μπορέσει να βρεί ο χρήστης μία λέξη του σταυρολέξου, αρκεί να δημιουργήσει το σωστό συνδιασμό γραμμάτων στο πλαίσιο WordSwipe. ‘Οπως έχουμε ήδη επισημάνει, κάθε συνδιασμός που δημιουργεί ο χρήστης ορίζεται στη μεταβλητή clickedLetters, η οποία και περνάει σαν prop από το WordSwipe στον CrossWordArray.

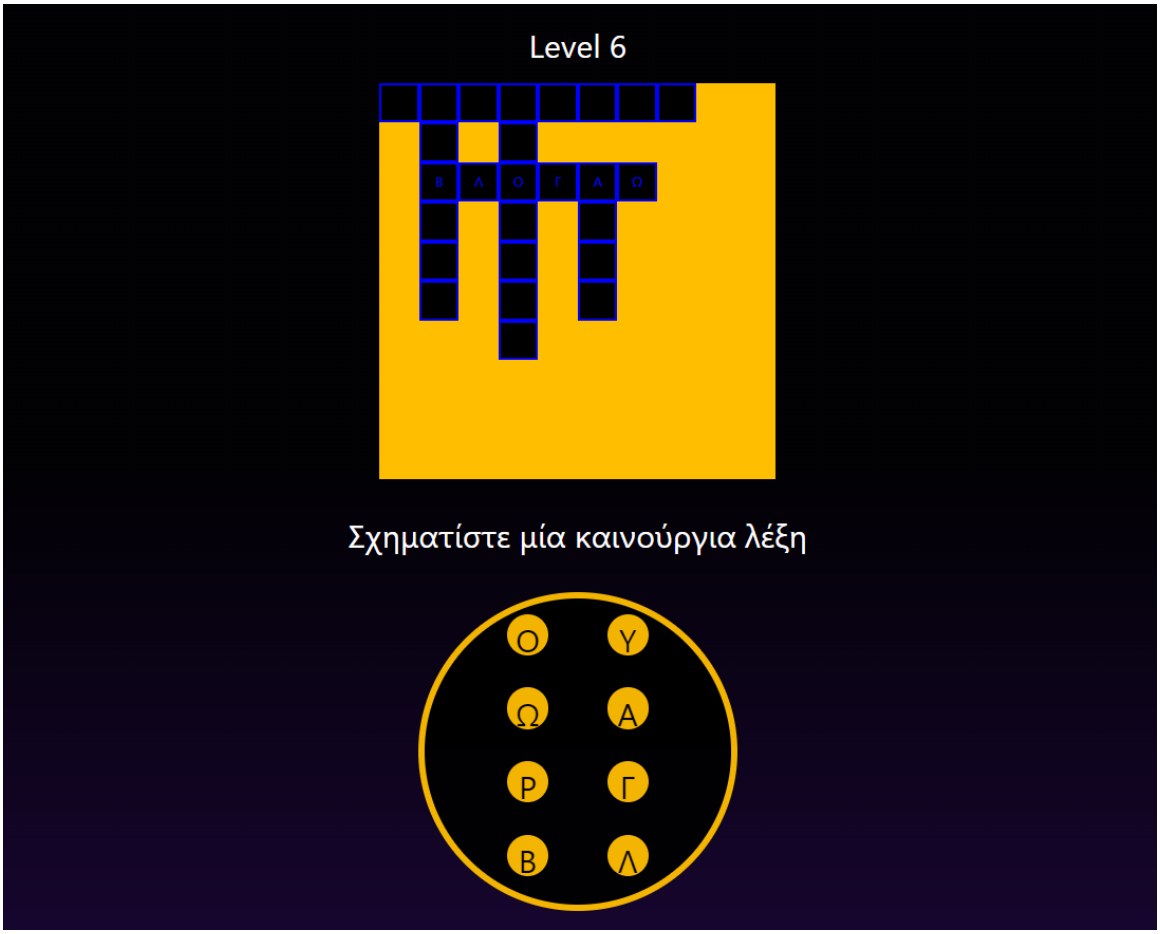
Μετά, δημιουργήσαμε κώδικα που να ελέγχει αν ο συνδιασμός των γραμμάτων του WordSwipe που έχει δημιουργήσει ο χρήστης απαρτίζουν μία λέξη του σταυρολέξου. Για να γίνει αυτό, απλά χρειάζεται να ελέγξουμε αν η μεταβλητή clickedLetters είναι ίση με κάποια από τις λέξεις που υπάρχουν στον πίνακα lexeisKaiTheseis.

Για να γίνει αυτό, ορίστηκε μία forEach, η οποία κάνει parse μία προς μία τις λέξεις που έχουν μπει στον array και, αν είναι ίσες με το clickedLetters, μετατρέπει το πεδίο found σε true.

Για κάθε λέξη που έχουμε εισχωρήσει στον πίνακα, έχουμε δώσει το style backgroundColor : black, κάτι που κάνει τις λέξεις να είναι στο ίδιο χρώμα με το κελί που τους περιβάλλει, δηλαδή εμφανισιακά αόρατες. Ελέγχουμε μία προς μία τις λέξεις που έχουμε βάλει στον CrossWordArray και, αν κάποια από αυτές έχει τη τιμή found: true, αλλάζουμε το backgroundColor. Με αυτό τον τρόπο, η κάθε λέξη που έχει βρει ο χρήστης γίνεται ευδιάκτιτη σε αυτόν, κάτι που του δείχνει ποιές λέξεις του έχουν απομείνει προς επίλυση, αλλά και μπορούν να τον βοηθήσουν στην επίλυση αυτών, αφού οι λέξεις που έχει βρει μπορεί να έχουν κοινά γράμματα με κάποια λέξη που δεν έχει καταφέρει ακόμα να εντοπίσει.

8.2 Ολοκλήρωση σταυρολέξου

Για να ολοκληρώσει ένας χρήστης το σταυρόλεξο θα πρέπει να έχει βρεί και τις 5 λέξεις που υπάρχουν σε αυτό. ‘Οπως δηλώσαμε στο κομμάτι της εύρεσης μίας λέξης, όταν ο χρήστης βρίσκει μία λέξη, η τιμή αυτής στον πίνακα lexeisKaiTheseis μετατρέπεται σε found: true. Επίσης, αλλάζει το color, δηλαδή το χρώμα των γραμμάτων στα κελιά που είναι η λέξη, από black σε blue, ώστε να είναι εμφανής η λέξη στο χρήστη.



Σχήμα 8.2.1 Σχηματισμός λέξης στο wordSwipe και εύρεση λέξης από το σταυρόλεξο

Γι' αυτό το λόγο, συντάξαμε ένα κώδικα ο οποίος ελέγχει όλα τα entries του lexeisKaiTheseis και εντοπίζει τη τιμή τους στο πεδίο found. Αν όλα τα entries του lexeisKaiTheseis έχουν στο πεδίο found τη τιμή 'true', αυτές έχουν βρεθεί από το χρήστη.

Για τον ορισμό αυτού του κώδικα, αρχικοποιούμε μία μεταβλητή foundLexeisKaiTheseis σε 0. Αυτή η μεταβλητή θα χρησιμοποιείται σαν counter, σαν μετρητή, που η τιμή του θα υξάνεται με κάθε λέξη που θα βρίσκει ο χρήστης.

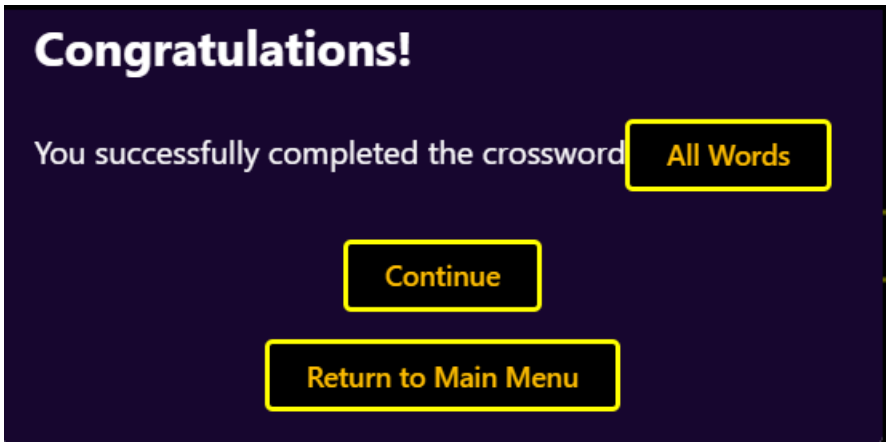
```
const [foundLexeisCount,setFoundLexeisCount] = useState(0);
```

Σχήμα 8.2.2 Ορισμός counter για εύρεση λέξεων σταυρολέξου

```
lexeisKaiTheseis.forEach(entry => {
    if(entry.found === 'true') {
        setFoundLexeisCount(count => count++)
    }
})
```

Σχήμα 8.2.3 Αύξηση counter με κάθε λυμμένη λέξη

Τέλος, με την εύρεση των 5 λέξεων του σταυρολέξου (όταν το foundLexeisCount ισούται με 5) καλούμε στη React ένα modal. Το modal αυτό ενημερώνει το χρήστη ότι έχει καταφέρει να επιλύσει επιτυχώς το σταυρόλεξο. Του δίνει την επιλογή 'Continue' ώστε να συνεχίσει σε κάποιο άλλο σταυρόλεξο. Στο modal ολοκλήρωσης παρουσιάζεται και ένα button 'Return to Main Menu'. Αυτό το button λειτουργεί ως ένα hyperlink, το οποίο οδηγεί το χρήστη ξανά στο component MainMenu. Τέλος, στο modal της ολοκλήρωσης, παρουσιάζεται ένα button με όνομα 'All Words'.



Σχήμα 8.2.4 Modal ολοκλήρωσης σταυρολέξου

8.3 All Words

Με την επίλυση του σταυρολέξου, εμφανίζεται στο χρήστη το modal που δείχνει ότι το σταυρόλεξο λύθηκε. Αυτό,εκτός του ότι επιτρέπει στο χρήστη τη συνέχιση σε επόμενο σταυρόλεξο ή την επιστροφή στο κύριο μενού, εμφανίζει ένα button με τιμή 'All Words' στο χρήστη.
Πατώντας αυτό το button, ανοίγει ένα ακόμα modal, το οποίο εμφανίζει όλες τις λέξεις που θα μπορούσαν να χρησιμοποιηθούν από το σταυρόλεξο, καθώς απαρτίζονται από τα γράμματα που δημιουργήθηκαν αυτόματα κατά την εισαγωγή στο επίπεδο.

Για την εισαγωγή σε αυτό το modal, αρχικοποιούμε μία μεταβλητή 'const [opened, setOpened] = useState(false);' και εισάγουμε ένα button το οποίο, πατώντας το ο χρήστης, μετατρέπει τη μεταβλητή σε true και ανοίγει το modal.

```
<Group position="center">
  <Button onClick={() => setOpened(true)}>All Words</Button>
</Group>
```

Σχήμα 8.3.1 Κώδικας ανοίγματος modal για όλες τις λέξεις

Το modal που ανοίγει διαθέτει το ίδιο UI με τα modals για το Login και το Sign Up, υιοθετώντας τις κλάσσεις αυτών με σκοπό την ομοιομορφία.

```
<Modal
  opened={opened}
  classNames={{
    modal: 'custom-modal',
    header: 'custom-modal-header',
    title: 'custom-modal-title',
    body: 'custom-modal-body',
  }}
  onClose={() => setOpened(false)}
  title="All Words"
>
```

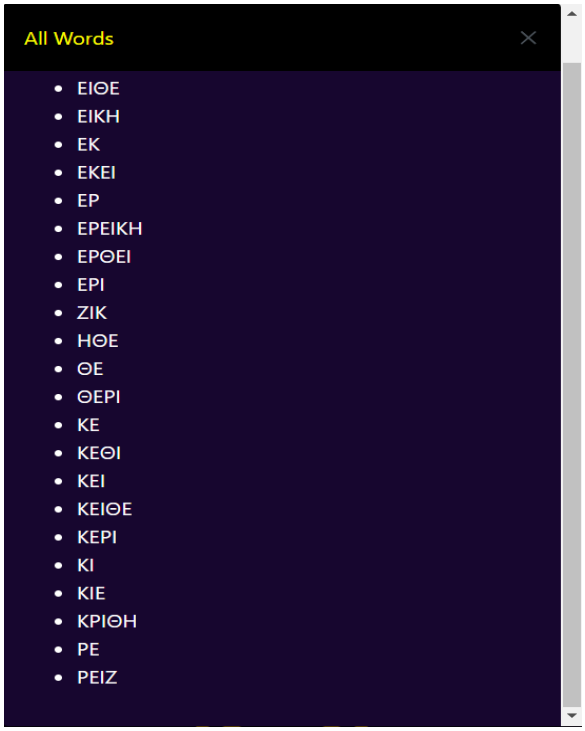
Σχήμα 8.3.2 Κλάσεις του modal All Words

Για την εμφάνιση των λέξεων, δημιουργήσαμε μία HTML unordered list, η οποία ορίζεται χρησιμοποιώντας το tag <ul> για τη σύνταξή της και το tag <li> για κάθε item αυτής [1]. Σαν items της unordered list θα εμφανίζεται η κάθε λέξη που αποτελείται από τα γράμματα που μας δόθηκαν. Γιά την επίτευξη αυτού, δημιουργήθηκε μία map, η οποία μετατρέπει κάθε matchingEntry σε ένα <li> στοιχείο.

```
<ul>
  {matchingEntries.map((entry, index) => (
    <li key={index}>
      {entry.grammata}
    </li>
  ))}
</ul>
```

Σχήμα 8.3.3 Κλάσεις του modal All Words

Σαν αποτέλεσμα αυτών των ενεργειών, το modal AllWords εμφανίζει όλες τις λέξεις που θα μπορούσαν να χρησιμοποιηθούν στο σταυρόλεξο στο χρήστη με τη μορφή λίστας.



Σχήμα 8.3.4 Το All Words με όλες τις λέξεις που θα μπορούσαν να χρησιμοποιηθούν στο σταυρόλεξο

8.4 Continue

Η επιλογή ‘Continue’ του modal ολοκλήρωσης του σταυρολέξου, περνάει τη τιμή του επιπέδου που έχει επιλέξει ο χρήστης (level) ξανά στο αρχείο Staurolexo. Με αυτό το τρόπο, επιτυγχάνουμε να φορτώσουμε ξανά τη σελίδα Staurolexo, η οποία, όπως αναλύσαμε προηγουμένως, δημιουργεί τυχαία γράμματα και με αυτά αναζητά λέξεις. Έπειτα, διασυνδέει τις λέξεις που βρήκε με αυτά τα γράμματα και τις ‘χτίζει’ πάνω σε ένα νέο CrossWordArray, δηλαδή ένα νέο σταυρόλεξο. Τουτέστην, ο χρήστης με την ολοκλήρωση του σταυρολέξου, πατώντας την επιλογή ‘Continue’, εισάγεται σε ένα καινούργιο σταυρόλεξο προς επίλυση και το gaming experience του συνεχίζεται.

Κεφάλαιο 9ο: Συμπεράσματα και Μελλοντικές επεκτάσεις

Στο πλαίσιο της παρούσας πτυχιακής εργασίας, αναδείξαμε τον τρόπο υλοποίησης ενός σταυρολέξου. ‘Εγινε κατανοητή η ευρεία κατανόηση των βασικών τεχνολογιών ιστού που χρειάζεται για την ανάπτυξη ενός τέτοιου εγχειρήματος. Επίσης, είδαμε ότι για να μπορέσει να είναι δυναμική μία τέτοια εφαρμογή, είναι απαραίτητη η χρήση των τεχνολογιών του MERN stack, όπως και η κατανόηση της λογικής της γραμμικής άλγεβρας.

Υπάρχουν αρκετές ιδέες και προοπτικές για μελλοντική ανάπτυξη της παρούσας εργασίας. Η μετάβαση της και στο ios, ώστε να είναι διαθέσιμη σε προϊόντα Apple μπορεί να ανοίξει νέους ορίζοντες. Η δημιουργία ακόμα περισσότερων επιπέδων, με μεγαλύτερη πληθώρα διαθέσιμων λέξεων προς εύρεση, φαντάζει ως μία απολύτως δυνατή λύση για τη βελτιστοποίηση του Word Wizardry. Ακόμα, μπορούν να εισαχθούν διαφορετικά λεξικά, ώστε να μπορούν να δημιουργηθούν σταυρόλεξα σε διαφορετικές γλώσσες, κάτι που μπορεί να καταστήσει το app διαθέσιμο σε ένα ευρύτερο κοινό. Τέλος, με την προσθήκη ενός campaign mode με το δικό του storyline, το Word Wirardry μπορεί να διαφοροποιηθεί ακόμα περισσότερο από τις εφαρμογές.

ΒΙΒΛΙΟΓΡΑΦΊΑ

[1] HTML: HyperText Markup Language, Available:
https://developer.mozilla.org/en-US/docs/Web/HTML

[2] Modern CSS Master the Key Concepts of CSS for Modern Web Development, Joe Attardi, 2020, Available:
https://link.springer.com/book/10.1007/978-1-4842-6294-8

[3] Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics, Jennifer Niederst Robbins, 2018, Available:
https://books.google.gr/books?hl=el&lr=&id=FJkVxtXr7n0C&oi=fnd&pg=PR11&dq=css&ots=lQCOS-9G3D&sig=Z7aBKY3LaAZgbrYIp2Nux39_N1E&redir_esc=y#v=onepage&q=css&f=false

[4] JavaScript: The Definitive Guide , David Flanagan, Available:
https://www.kufunda.net/publicdocs/JavaScript%20The%20Definitive%20Guide%20(David%20Flanagan).pdf

[5] An analysis of the dynamic behavior of JavaScript programs, Gregor Richards, Sylvain Lebresne, Brian Burg, Jan Vitek, June 05 2010, Available:
https://dl.acm.org/doi/pdf/10.1145/1806596.1806598

[6] Visual Studio Documentation, Available:
https://visualstudio.microsoft.com/

[7] Visual Studio Documentation, Available:
https://code.visualstudio.com/

[8] Ilustration about Mern stack, March 21 2021, Available:
https://medium.com/techiepedia/what-exactly-a-mern-stack-is-60c304bffbe4

[9] Introduction to React, Cory Gackenheimer, 2015, Available:
https://link.springer.com/chapter/10.1007/978-1-4842-1245-5_1

[10] https://s3.amazonaws.com/assets.fullstack.io/books/fullstackreact/349fb8eb5ebbe5b97/sample-chapter/fullstackreact-sample-chapter-book-sample-chapter.pdf?inf_contact_key=aa8f567be7ae92131a3df7fe08ed0245842e902fbefb79ab9abae13bfcb46658

[11] Fullstack React The Complete Guide to ReactJS and Friends , Anthony Accomazzo, Nate Murray, Ari Lerner, Clay Allsopp, David Guttman, and Tyler McGinnis, 2020, Available:
https://books.google.gr/books?hl=el&lr=&id=7KVYDwAAQBAJ&oi=fnd&pg=PT20&dq=react&ots=gFVkJtSr7I&sig=H6ZwxBUojGJUwvWzt-a2J5w5NF0&redir_esc=y#v=onepage&q=react&f=true

[12] State: A Component's Memory, Available:
https://react.dev/learn/state-a-components-memory

[13] React Hooks: useState (With Practical Examples), Tito Adeoye, Feb 5 2024, Available:
https://medium.com/@titoadeoye/react-hooks-usestate-with-practical-examples-64abd6df6471

[14] React Key Concepts: Consolidate your knowledge of React's core features, Maximilian Schwarzmuller, 2022, Available:
https://books.google.gr/books?hl=el&lr=&id=ykqkEAAAQBAJ&oi=fnd&pg=PP1&dq=react&ots=2cnSbOEFZl&sig=NFhFnKqwlQFFwDmG-kI80EbkwFU&redir_esc=y#v=onepage&q=react&f=false

[15] React state management and side-effects – A Review of Hooks , Krutika Patil, Sanath Dhananjayamurty Javagal, Dec 2022, Available:
https://d1wqtxts1xzle7.cloudfront.net/96239902/IRJET_V9I1225-libre.pdf?1671784340=&response-content-disposition=inline%3B+filename%3DReact_state_management_and_side_effects.pdf&Expires=1725532476&Signature=aFRpNq-j~cUOa2p19WgbLviPcvBBNlosdtubLmCERdyzfIj9-4KG2ezMYxJ3ouH5fLSzsiBlBWCpIwT2oneWqubSzSm-5qMcpo34kEY-tkddV6AfQjnmKxTDlPo3yEDQem7jcDy74~NI1vVoVzsgrH1SYwXZUu6GhEJNpl4Sa24zqJuzPAfcW6zf6QSrCtT8asUoFm87eeJbm5Smds2yam0Nt5~jJW-U3L8kSm7-VZLhi09SinHjxKQIaaiUk7q2oB8P59vf5JQLdkjS~vGUHdwxFbD2Ak3zmaA7Cj0Cic9oeHYm8vXK519DLj8dG2MklR85tXxl3UKB-ZV1wpksuA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[16] USABILITY AND ADAPTATION OF REACT HOOKS, Tuomas Luojus, April 2021, Available:
https://trepo.tuni.fi/bitstream/handle/10024/130986/LuojusTuomas.pdf?sequence=2&isAllowed=y

[17] Built-in React Hooks
https://react.dev/reference/react/hooks

[18] A comparative study: MongoDB vs. MySQL, Cornelia Győrödi, Robert Győrödi, George Pecherle, Andrada Olah, 12 June 2015, Available:
https://ieeexplore.ieee.org/abstract/document/7158433?casa_token=q6q57G4N5GQAAAAA:-EJnO8zDlYUVgLLyv703FxKne-IsMblsOiyzfn5m0TOEpvSUj2jd-NhTjp0jipdIvmd05hRUFfM

[19] Node.js Challenges in Implementation, Hezbullah Shah, Tariq Rahim Soomro, May 2017, Available:
https://www.researchgate.net/profile/Hezbullah-Shah/publication/318310544_Nodejs_Challenges_in_Implementation/links/59634e42458515a3575451a6/Nodejs-Challenges-in-Implementation.pdf

[20] Constructing a Study Buddy Using MERN (MongoDB, Express.js, React, Node.js) Stack Technologies , Chamalla Sravani,Pudi Kumar,Sanapala Priya,Sujith Kumar Yadav,Madina Jayanthi Rao and Urlam Devi Prasan, 17 July 2024, Available:
https://www.mdpi.com/2673-4591/66/1/27

[21] Express in Action: Writing, building, and testing Node.js applications, Evan Hahn, April 19, 2016, Available:
https://books.google.gr/books?hl=el&lr=&id=czkzEAAAQBAJ&oi=fnd&pg=PT13&dq=express+js&ots=8BjIUpBsn0&sig=TLDhazATmk2ojKqVyUFb4dE4AqY&redir_esc=y#v=onepage&q=express%20js&f=false

[22] The React Handbook – Learn React for Beginners, Flavio Copes, January 8, 2019, Available:
https://www.freecodecamp.org/news/the-react-handbook-b71c27b0a795/#the-virtual-dom

[23] Jotai Documentation, available:
https://jotai.org/

[24] Jotai Documentation, available:
https://github.com/pmndrs/jotai

[25] Impact of React component libraries on developer experience, Ossian Rajala, July 23, 2024, Available:
https://aaltodoc.aalto.fi/server/api/core/bitstreams/5f0f95f6-2342-4f33-836c-b408f3b597dd/content

[26] Mantine Documentation, available:
https://mantine.dev/

[27] BUILDING AN E-COMMERCE WEBSITE USING NEXT JS,MANTINE, AND STRAPI, An Nguyen, May 2022, Available:
https://www.theseus.fi/bitstream/handle/10024/751226/Nguyen_An.pdf?sequence=2&isAllowed=y

[28] How To Restart Your Node.js Apps Automatically with nodemon, Bradley Kouchi, April 1, 2024, Available:
https://www.digitalocean.com/community/tutorials/workflow-nodemon

[29] Nodemon Documentation, available:
https://www.npmjs.com/package/nodemon

[30] Essential Typescript 4, Adam Freeman, 2021, Available:
https://link.springer.com/chapter/10.1007/978-1-4842-7011-0_3

[31] Υλοποίηση ενός serious educational web-based game for java programming tutorial, Νίκος Νταντινάκης, 2019, available:
https://apothesis.lib.hmu.gr/bitstream/handle/20.500.12688/9410/NtantinakisNikolaos2019.pdf?sequence=1&isAllowed=y

[32] WEB APPLICATION DEVELOPMENT with React, Node.js and SQL, Duy Le, 2024, Available:
https://www.theseus.fi/bitstream/handle/10024/851908/Duy_Le.pdf?sequence=2&isAllowed=y

[33] Capacitor Documentation, available:
https://capacitorjs.com/

[34] A complete Guide: “Building a CapacitorJS application using pure JavaScript and Webpack”, Luca R., Aug 26, 2021, Available:
https://medium.com/@SmileFX/a-complete-guide-building-a-capacitorjs-application-using-pure-javascript-and-webpack-37d00f11720d

[35] Building A CRUD Application Using Node JS And MongoDB Atlas, Altaf Shaikh, Posted on Apr 18, 2021, Updated on Sep 6, Available:
https://dev.to/ialtafshaikh/building-a-crud-application-using-node-js-and-mongodb-atlas-2df5

[36] What is Mongoose?, Monib Bormon, Mar 4, 2022, Available:
https://monib-bormon.medium.com/what-is-mongoose-c1bc3031cc08

[37] Web Development with MongoDB and NodeJS, Mithun Satheesh, Bruno Joseph D'mello, Jason Krol, October 30 2015, Available:
https://www.google.gr/books/edition/Web_Development_with_MongoDB_and_NodeJS/4QKACwAAQBAJ?hl=el&gbpv=1&dq=mongoose+js&printsec=frontcover

[38] Mongoose for Application Development, Simon Holmes, August 26 2013, Available:
https://www.google.gr/books/edition/Mongoose_for_Application_Development/DxTFXO771tIC?hl=el&gbpv=1&dq=mongoose&printsec=frontcover

[39] Canva, Klug Brandy, Williams Ursula, 1 April 2016, Available:
https://annurev.publisher.ingentaconnect.com/contentone/annurev/tca/2016/00000017/00000004/art00006

[40] Canva, Alison Paige Gehred, Apr 1 2020, Available:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7069818/

[41] CSS the Definitive Guide Web Layout and Presentation, Eric A. Meyer, Estelle Weyl, 2023, Available: https://books.google.gr/books?hl=el&lr=&id=ILHBEAAAQBAJ&oi=fnd&pg=PP1&dq=linear+gradient+css&ots=Cdu6WdpRdz&sig=WQTIyNimHA2uFvK5F9H9-_IZ28c&redir_esc=y#v=onepage&q=linear%20gradient%20css&f=false

[42] What Is a Modal and When Should I Use One?, Jamie Juviler, Published: June 27, 2021,Updated: April 01, 2022, Available: https://blog.hubspot.com/website/modal-web-design

[43] The LaTex Web Companion: Integrating TeX, HTML, and XML, Michel Goossens, S. P. Q. Rahtz, Sebastian Rahtz, 1999, Available: https://books.google.gr/books?hl=el&lr=&id=JJANltqWOM0C&oi=fnd&pg=PR11&ots=dYbM_Mshxh&sig=bCjpgHDuJ2Z8SQ4GtRXBUMo2I5A&redir_esc=y#v=onepage&q=ul&f=false

[44] Cross-Origin Resource Sharing (CORS), Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[45] Logical Error, Available: https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/logical-error/