ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Σχεδιασμός και ανάπτυξη rest api με συλλογή δεδομένων για τα χιονοδρομικά στην Ελλάδα»

Του φοιτητή
Πάντσος Ξενοφών Νικόλαος
Αρ. Μητρώου: 185251

Επιβλέπων
Χαράλαμπος Μπράτσας
Επίκουρος καθηγητής

Σεπτέμβριος 2024

Τίτλος Δ.Ε. **Σχεδιασμός και ανάπτυξη rest api με συλλογή δεδομένων για τα χιονοδρομικά στην Ελλάδα**

Κωδικός Δ.Ε. **24110**

Ονοματεπώνυμο φοιτητή: **Πάντσος Ξενοφών Νικόλαος**

Εισηγητής: **Χαράλαμπος Μπράτσας**

Ημερομηνία ανάληψης Δ.Ε. **25-01-2024**

Ημερομηνία περάτωσης Δ.Ε. **10-09-2024**

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

«Στην οικογένειά μου και σε όλους μου τους φίλους»

Πρόλογος

Ο λόγος που επέλεξα να αναπτύξω αυτήν τη πτυχιακή εργασία ήταν η επιθυμία μου να συνδυάσω τις γνώσεις μου στην ανάπτυξη λογισμικού με την αγάπη μου για το σκι. Η δημιουργία ενός REST API που παρέχει πληροφορίες σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας, φάνηκε μια πρόκληση που θα μου επέτρεπε να εφαρμόσω τις γνώσεις μου σε πραγματικά δεδομένα και να συμβάλλω στην ανάπτυξη ενός εργαλείου που θα δώσει την δυνατότητα να δημιουργηθούν εφαρμογές για το διαδίκτυο και τα κινητά που θα χρησιμοποιούνται από τους λάτρεις των χειμερινών αθλημάτων.Μέσα από αυτήν την εργασία, απέκτησα βαθύτερη κατανόηση της ανάπτυξης REST API, της διαχείρισης βάσεων δεδομένων και της φιλοξενίας εφαρμογών σε production environment. Επιπλέον, βελτίωσα τις δεξιότητές μου στην επίλυση προβλημάτων στην διαχείριση μεγάλων ποσοτήτων δεδομένων και στη βελτιστοποίηση της απόδοσης (performance optimization). Το έργο αυτό όχι μόνο με εξόπλισε με νέες τεχνολογικές δεξιότητες, αλλά μου έδωσε και την ικανοποίηση ότι συνέβαλα σε έναν τομέα που με ενδιαφέρει προσωπικά.

Περίληψη

Η παρούσα πτυχιακή εργασία ασχολείται με την ανάπτυξη ενός REST API για την παροχή πληροφοριών σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας. Το API δημιουργήθηκε με τη χρήση του PHP framework Laravel και μιας MySQL βάσης δεδομένων, ενώ φιλοξενείται σε έναν VPS server της Digital Ocean. Η συλλογή των δεδομένων πραγματοποιείται με την μέθοδο του web scraping από μια αξιόπιστη διαδικτυακή εφαρμογή το snowreport.Ο κύριος στόχος ήταν να αναπτυχθεί ένα εργαλείο που θα παρέχει στους χρήστες δεδομένα σχετικά με τις διαθέσιμες πίστες, τα lift που λειτουργούν καθώς και άλλες σημαντικές πληροφορίες για τα χιονοδρομικά κέντρα σε πραγματικό χρόνο. Το API οργανώθηκε σε διαφορετικά endpoints, καθένα από τα οποία εξυπηρετεί συγκεκριμένες πληροφορίες, όπως τη λίστα των χιονοδρομικών κέντρων με λεπτομέρειες για κάθε κέντρο, τις τρέχουσες συνθήκες σε αυτά, τα lifts τα οποία είναι σε λειτουργία αλλά και διαδικασίες όπως την αγορά εισιτηρίου για το εκάστοτε χιονοδρομικό .Ιδιαίτερη έμφαση δόθηκε στη σωστή διαχείριση και αποθήκευση των δεδομένων, καθώς και στη βελτιστοποίηση της απόδοσης του συστήματος ώστε να είναι σε θέση να διαχειριστεί μεγάλο όγκο αιτήσεων από τους χρήστες. Κατά τη διάρκεια της ανάπτυξης, αντιμετωπίστηκαν προκλήσεις που αφορούσαν την ασφάλεια του API, την αποτελεσματική διαχείριση των δεδομένων και την επίτευξη υψηλής διαθεσιμότητας και απόδοσης. Τα αποτελέσματα της εργασίας επιβεβαιώνουν ότι το API μπορεί να χρησιμοποιηθεί ως ένα αξιόπιστο εργαλείο για την παροχή πληροφοριών στους χρήστες, συμβάλλοντας στη βελτίωση της εμπειρίας τους στα χιονοδρομικά κέντρα της χώρας. Η εργασία αυτή προσφέρει μια ολοκληρωμένη λύση για τη διασύνδεση και την παροχή δεδομένων που αφορούν τα χιονοδρομικά κέντρα, με δυνατότητες επέκτασης και προσαρμογής, ώστε να καλύπτει τις μελλοντικές ανάγκες και τεχνολογικές εξελίξεις.

SnowHub rest api

Pantsos Xenofon Nikolaos

Abstract

This thesis focuses on the development of an API that provides information about the ski resorts in Greece. The API was developed using the PHP framework Laravel and a MySQL database, and it is hosted on a Digital Ocean VPS server. One of the primary goals of the project was to automate the collection and processing of data from various online sources using web scraping methods to ensure that the information provided is accurate and up-to-date.The data collection was carried out through web scraping from the snowreport.gr .The scraped data includes information about ski slopes, weather conditions, ticket prices, and other relevant details. The API is organized into multiple endpoints, offering users access to information such as a list of ski resorts, details about each resort, and current operating conditions.The development of the API involved addressing challenges such as data security, performance optimization, and ensuring continuous data updates through automated scraping processes. The results of this project demonstrate that the API can be used as a reliable tool for providing users with up-to-date information, making it easier for them to access important details about ski resorts in Greece.This project offers a comprehensive solution for collecting, organizing, and delivering data using web scraping techniques. It also provides opportunities for future expansion and adaptation to meet evolving needs and technological advancements.

Ευχαριστίες

Θα ήθελα να εκφράσω τις βαθύτατες ευχαριστίες μου σε όλους όσους συνέβαλαν στην ολοκλήρωση αυτής της πτυχιακής εργασίας.

Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου,Κ. Χαράλαμπο Μπράτσα, για την αμέριστη καθοδήγηση, την πολύτιμη υποστήριξη και τα κίνητρα που μου προσέφερε καθ' όλη τη διάρκεια της εργασίας. Οι πολύτιμες συμβουλές του ήταν καθοριστικές για την πορεία και την επιτυχή ολοκλήρωση αυτού του έργου.

Ευχαριστώ θερμά την οικογένειά μου, που με στήριξε σε κάθε βήμα, με ενθάρρυνε και μου παρείχε το απαραίτητο περιβάλλον για να αφοσιωθώ στις σπουδές μου.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους και συμφοιτητές μου, που στάθηκαν δίπλα μου σε αυτή την απαιτητική περίοδο. Οι συζητήσεις, οι ανταλλαγές ιδεών και η φιλική τους υποστήριξη έπαιξαν σημαντικό ρόλο στη διαμόρφωση της τελικής εργασίας.

Σε όλους σας, ένα μεγάλο ευχαριστώ!

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

vps	Virtual Private Server
api	Application Programming Interfaces
IDE	Integrated Development Environment

Κεφάλαιο 1ο: Εισαγωγή στο REST API και Τεχνολογίες

1.1 Εισαγωγή

Η τεχνολογία των API (Application Programming Interfaces) έχει αναδειχθεί ως ένας από τους πιο σημαντικές μεθόδους στην ανάπτυξη λογισμικού. Στο 2024 τα rest api αποτελούν μια σταθερά στην ανάπτυξη εφαρμογών καθώς προσφέρουν ευελιξία επαναχρησιμοποιησιμότητα διευκολύνοντας την δημιουργία σύνθετων εφαρμογών .Η συγκεκριμένη πτυχιακή εργασία επικεντρώνεται στην ανάπτυξη ενός api που παρέχει πληροφορίες σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας .Τις πληροφορίες όπου χρειάζεται για να είναι ολοκληρωμένο τις συλλέγει μέσο web scraping εξασφαλίζοντας με μια αυτοματοποιημένη μέθοδο ότι οι χρήστες λαμβάνουν αξιόπιστες πληροφορίες σχετικά με το ποια χιονοδρομικά και λιφτς σε αυτά είναι ανοιχτά .Η ανάπτυξη του API έγινε χρησιμοποιώντας το Laravel, ένα δημοφιλές PHP framework, σε συνδυασμό με MySQL για την αποθήκευση των δεδομένων. Η εφαρμογή φιλοξενείται σε VPS της Digital Ocean, εξασφαλίζοντας σταθερότητα και υψηλή διαθεσιμότητα. Το κεφάλαιο αυτό θα εισάγει τον αναγνώστη στην έννοια του rest api τις βασικές τεχνολογίες που έγινε χρήση τους για την ανάπτυξη του επίσης και τις μεθόδους συλλογής δεδομένων .

1.2 Στόχοι και Αντικείμενο Έρευνας

Ο κύριος στόχος αυτής της πτυχιακής εργασίας είναι η ανάπτυξη ενός πλήρους και λειτουργικού εργαλείου όπου θα γίνει χρήση του για την ανάπτυξη σχετικών εφαρμογών με τα χιονοδρομικά .Μετά από προσωπική εμπειρία έγινε κατανοητό ότι οι υπάρχων εφαρμογές είναι απαρχαιωμένες και δεν προσφέρουν μια ολοκληρωμένη εμπειρία στον χρήστη.Ωστοσο τα δεδομένα που κάναμε χρήση από υπάρχων εφαρμογές είναι πλήρως αξιόπιστα.

Ειδικότερα, οι στόχοι της εργασίας περιλαμβάνουν:

- **Ανάπτυξη ενός API με τη χρήση του Laravel**: Η εργασία αποσκοπεί στη δημιουργία ενός API που να είναι ευέλικτο, ασφαλές και εύκολο στη χρήση, χρησιμοποιώντας το PHP framework Laravel. Το Laravel επιλέχθηκε για τη δομή, την ευκολία στη διαχείριση και τη δυνατότητά του να υποστηρίζει σύνθετες λειτουργίες αλλά και λόγο εμπειρίας στο συγκεκριμένο framework και γλώσσα .
- **Συλλογή και Διαχείριση Δεδομένων μέσω Web Scraping**: Κύριος στόχος είναι η αυτόματη συλλογή των δεδομένων που είναι μεταβαλλόμενα όπως το ποια λιφτ είναι ανοιχτά αλλά και ποια χιονοδρομικά .Αυτη η διαδικασία είναι απαραίτητη καθώς σε μια μέρα λόγω καιρικών συνθηκών ή μιας βλάβης μπορεί να αλλάξει αναπάσα στιγμή η λειτουργία του χιονοδρομικού. Αυτή η διαδικασία απαιτεί την ανάπτυξη ενός αποτελεσματικού συστήματος scraping, το οποίο θα εξασφαλίζει την ακρίβεια και την επικαιρότητα των δεδομένων.
- **Αποθήκευση και Οργάνωση Δεδομένων**: Η εργασία στοχεύει επίσης στη δημιουργία μιας καλά δομημένης βάσης δεδομένων χρησιμοποιώντας MySQL. Η οργάνωση των δεδομένων θα επιτρέπει την αποτελεσματική αποθήκευση, ανάκτηση και επεξεργασία των πληροφοριών που συλλέγονται.
- **Διασφάλιση Ασφάλειας και Απόδοσης του API**: Ένας άλλος στόχος είναι η ανάπτυξη μηχανισμών που θα εξασφαλίζουν την ασφάλεια των δεδομένων και την προστασία από κακόβουλες επιθέσεις, καθώς και η βελτιστοποίηση του API ώστε να μπορεί να ανταποκρίνεται γρήγορα και αξιόπιστα σε μεγάλους όγκους αιτήσεων.Μεγάλη συμβολή σε

αυτό παίζει το framework της laravel καθώς έχει διάφορες αυτοματοποιημένες διαδικασίες και έτοιμες λύσεις ώστε να διασφαλίσει την απόδοση και την ασφάλεια στο api .

1.3 Δομή της πτυχιακής Εργασίας

Η συγκεκριμένη πτυχιακή εργασία είναι φτιαγμένη σε τέσσερα βασικά κεφάλαια, τα οποία καλύπτουν συνολικά την ανάπτυξη, υλοποίηση και αξιολόγηση του API για τα χιονοδρομικά κέντρα της Ελλάδας.

- **Κεφάλαιο 1: Εισαγωγή στο API και Τεχνολογίες** Στο πρώτο κεφάλαιο, παρουσιάζεται η γενική εισαγωγή στο θέμα της εργασίας, οι στόχοι της έρευνας, καθώς και το αντικείμενο της μελέτης. Προβάλλονται επίσης οι βασικές τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του API, καθώς και οι μέθοδοι συλλογής δεδομένων μέσω web scraping.
- **Κεφάλαιο 2: Ανάπτυξη και Υλοποίηση API** Στο δεύτερο κεφάλαιο, περιγράφεται η διαδικασία ανάπτυξης του API. Καλύπτονται οι λεπτομέρειες του σχεδιασμού της βάσης δεδομένων, η υλοποίηση των endpoints, ο τρόπος όπου τα δεδομένα που συλλέχθηκαν μέσω web scraping τοποθετήθηκαν στην βάση , και η φιλοξενία του API σε VPS server της Digital Ocean.
- **Κεφάλαιο 3: Αποτελέσματα και Αξιολόγηση** Το τρίτο κεφάλαιο στοχεύει στην αξιολόγηση του API που αναπτύχθηκε. Παρουσιάζονται τα αποτελέσματα από τη χρήση του API, αξιολογείται η απόδοσή του όταν βρίσκεται public .
- **Κεφάλαιο 4: Συμπεράσματα και Προτάσεις Βελτίωσης** Στο τελευταίο κεφάλαιο, παρουσιάζονται τα τελικά συμπεράσματα.Εξετάζονται οι περιορισμοί της τρέχουσας υλοποίησης και προτείνονται βελτιώσεις και επεκτάσεις που θα μπορούσαν να γίνουν στο μέλλον για να αυξηθεί η λειτουργικότητα και η αποτελεσματικότητα του .

Τέλος, παρατίθενται η βιβλιογραφία .

1.4 Τεχνολογίες Ανάπτυξης API

Για την ανάπτυξη του API που παρέχει πληροφορίες σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας, χρησιμοποιήθηκαν σύγχρονες και αξιόπιστες τεχνολογίες που εξασφαλίζουν την απόδοση, την ασφάλεια και την ευελιξία του συστήματος. Οι κύριες τεχνολογίες που χρησιμοποιήθηκαν περιγράφονται παρακάτω.

1.4.1 PHP

Η php είναι μια γλώσσα προγραμματισμού για την δημιουργία εφαρμογών διαδικτύου. Χρησιμοποιείται από την μεριά του σέρβερ π.χ apache ή nginx.Η ιστορία της ξεκινάει από το 1993 όπου την δημιούργησε ένας φοιτητής .Πλέον βρίσκεται στην έκδοση 8 και είναι μια από τις γνωστότερες γλώσσες προγραμματισμού με πληθώρα από βιβλιοθήκες και frameworks .

1.4.3 Laravel Framework

Το Laravel είναι ένα δημοφιλές PHP framework που χρησιμοποιήθηκε για την ανάπτυξη του API. Έγινε χρήση του καθώς διευκολύνει την ανάπτυξη σύνθετων εφαρμογών με λιγότερο κώδικα αλλά και με λιγότερα πιθανά κενά ασφαλείας . Το laravel έχει έτοιμες βιβλιοθήκες όπως το eloquent ORM που

αφορά την αλληλεπίδραση με την βάση δεδομένων και το artisan command για την αυτοματοποίηση εργασιών όπως δημιουργία μοντέλων και άλλων .

1.4.4 MySQL

Για την αποθήκευση των δεδομένων του API, χρησιμοποιήθηκε η βάση δεδομένων MySQL.Η MySQL είναι από τα πιο γνωστα συστήματα σχεσιακων βάσεων δεδομένων .Χρησιμοποιήθηκε ώστε να γίνει η αποθήκευση των δεδομένων και αυτών που συλλέγονται μέσω του web scraping.

1.4.5 VPS

Η φιλοξενία της εφαρμογής έγινε σε έναν Virtual Private Server (VPS) της Digital Ocean που προσφέρει τα χρήσιμα εργαλεία ώστε να γίνει public στο διαδίκτυο .Επιλέχθηκε η συγκεκριμένη εταιρεία καθώς είναι από τις δημοφιλέστερες στον χώρο .Χρησιμοποιείται ένα σύστημα με 1 gb ram με δυνατότητα transfer 1000Gib και ssd 25 Gib.

1.4.6 Postman

Το postman χρησιμοποιήθηκε ως εργαλείο δοκιμών κατά τη διάρκεια της ανάπτυξης του API. Είναι ένα εργαλείο όπου επιτρέπει την δημιουργία request προς ένα σέρβερ και να λαμβάνει απαντήσεις από αυτόν .Με αυτόν το τρόπο δοκιμάστηκε η σωστή λειτουργία των endpoints του api καθώς και αν επιστρέφει σωστά αποτελέσματα .Με αυτόν το τρόπο βελτιώθηκε η απόδοση του και η αξιοπιστία του.

1.4.7 GitHub

Το **GitHub** και τα git commands χρησιμοποιήθηκαν για τη διαχείριση του κώδικα της εφαρμογής. Το git είναι ένα σύστημα version control. Το work flow που ακολουθήθηκε είναι η δημιουργία ενός brunch από το main project, ανάπτυξη του τασκ σε αυτό ,pull request και έλεγχος του ώστε να γίνει merge στο main προτζεκτ .Με αυτόν το τρόπο επιτευχθεί η διατήρηση του κώδικα η αποφυγή λαθών και η καλύτερη διαχείριση του προτζεκτ.

Η επιλογή αυτών των τεχνολογιών συνεισέφερε στην ανάπτυξη ενός αξιόπιστου, αποδοτικού και ασφαλούς API, το οποίο μπορεί να ανταποκριθεί στις απαιτήσεις των χρηστών και να παρέχει ακριβείς πληροφορίες για τα χιονοδρομικά κέντρα της Ελλάδας.

1.4.8 PhpStorm

Το PhpStorm είναι ένα δημοφιλές IDE (Integrated Development Environment) που αναπτύσσεται από την JetBrains και χρησιμοποιείται κυρίως για την ανάπτυξη εφαρμογών σε PHP. Είναι σχεδιασμένο να προσφέρει ισχυρά εργαλεία για τον προγραμματιστή, διευκολύνοντας τη διαδικασία ανάπτυξης λογισμικού μέσω αυτοματοποίησης, debugging και υποστήριξης πολλών γλωσσών προγραμματισμού.

Το PhpStorm είναι ένα ισχυρό εργαλείο για τον προγραμματιστή, ειδικά σε έργα που σχετίζονται με PHP, προσφέροντας πλούσια χαρακτηριστικά για να αυξήσει την παραγωγικότητα και να βελτιώσει την ποιότητα του κώδικα.

1.5 Μέθοδοι Συλλογής Δεδομένων

Τα δεδομένα αποτελούν το πιο κρίσιμο σημείο σε ένα api καθώς από αυτά εξαρτάται η αξιοπιστία της εφαρμογής όπου θα χτιστεί .Στη παρούσα πτυχιακή εργασία και πρότζεκτ τα δεδομένα συλλέγονται χειροκίνητα από τις επίσημες ιστοσελίδες των χιονοδρομικών ωστόσο τα μεταβαλλόμενο δεδομένα συλλέγονται με την διαδικασία του web scraping στην ιστοσελίδα του snowreport.gr

1.5.1 Web scraping

Το web scraping είναι μια τεχνική η οποία επιτρέπει την αυτόματη εξαγωγή δεδομένων από ιστοσελίδες.Μέσω ειδικά διαμορφωμένων scripts, τα δεδομένα εξήχθησαν σε τακτά χρονικά διαστήματα, αποθηκεύτηκαν στη βάση δεδομένων του API. Συγκεκριμένα για την λήψη των πληροφοριών για τους διαθέσιμους αναβατήρες γίνεται scraping μετά από ένα requests στο api και εφόσον έχουν περάσει 10 λεπτά από την τελευταία εγγραφεί στην βάση δεδομένων .Δηλαδη γίνεται χρήση της διαδικασίας το πολύ ανά 10 λεπτα και εφόσον αυτό απαιτηθεί .

Η διαδικασία του web scraping περιλαμβάνει τα εξής στάδια:

- **Αναγνώριση Πηγών**: Αξιολόγηση της ιστοσελίδας ή των ιστοσελίδων που θα γίνει η προαναφερθείσα διαδικασία .
- **Ανάλυση Δομής Ιστοσελίδων**: Αναλύθηκε η δομή των ιστοσελίδων ώστε να καθοριστούν τα απαραίτητα στοιχεία που πρέπει να εξαχθούν.
- **Ανάπτυξη Web Scraping Scripts**: Αναπτύχθηκαν scripts σε γλώσσα προγραμματισμού php .
- **Καθαρισμός και Επεξεργασία Δεδομένων**: Τα δεδομένα που συλλέχθηκαν υποβλήθηκαν σε διαδικασίες καθαρισμού και επεξεργασίας ώστε να είναι έτοιμα για αποθήκευση και χρήση.

Η κλάση Scraping είναι υπεύθυνη για την εξαγωγή δεδομένων (scraping) από την ιστοσελίδα του Snow Report και την ενημέρωση των δεδομένων για τα χιονοδρομικά κέντρα στην εφαρμογή. Αναλύει το περιεχόμενο της σελίδας και επιστρέφει πληροφορίες σχετικά με τους αναβατήρες (lifts) αν είναι ανοιχτά ή κλειστά.

1. **Μέθοδος getSnowReportPage($skiCenter)**

   Η μέθοδος αυτή αναλαμβάνει να στείλει ένα HTTP αίτημα σε συγκεκριμένη σελίδα του Snow Report χρησιμοποιώντας την cURL. Η παράμετρος $skiCenter καθορίζει το χιονοδρομικό κέντρο από το οποίο θα γίνει το scraping.

2. **Μέθοδος ScrapingPage($page)**

   Η μέθοδος ScrapingPage αναλύει το HTML της σελίδας που λαμβάνει από το Snow Report και εξάγει τα δεδομένα που αφορούν την κατάσταση των lifts.

```php
public function ScrapingPage($page)
{
    libxml_use_internal_errors(true);  // Επιτρέπει τη διαχείριση των σφαλμάτων

    $lifts = [];
    $doc = new DOMDocument();
    $doc->loadHTML($page);
    $selector = new DOMXPath($doc);
    $result = $selector->query('//div[@class="lift-block"]');
```

```
    if ($result->length >= 2) { // Ελέγχει αν υπάρχουν τουλάχιστον 2 στοιχεία
        $lifts['today'] = $this->getOpenLifts($result->item(0));
        $lifts['tomorrow'] = $this->getOpenLifts($result->item(1));
    }

    return $lifts;
}
```

- **DOMDocument**: Δημιουργείται ένα αντικείμενο DOMDocument για να φορτώσει το HTML της σελίδας.
- **DOMXPath**: Χρησιμοποιείται για την εκτέλεση XPath queries και την εύρεση των στοιχείων που περιέχουν πληροφορίες για τα lifts.
- **Αποτελέσματα**: Εξάγονται τα δεδομένα για τα lifts της τρέχουσας ημέρας και της επόμενης.

3. **Μέθοδος getOpenLifts($result)**

   Η μέθοδος αυτή εξάγει τις πληροφορίες για τα αναβατήρια (lifts) από το αποτέλεσμα του XPath query, διαχωρίζοντας τα ανοιχτά και κλειστά lifts.

   - Κόκκινα Lifts: Τα lifts με κόκκινο χρώμα θεωρούνται κλειστά και αποθηκεύονται με τιμή 0.
   - Πράσινα Lifts: Τα πράσινα lifts είναι ανοιχτά και αποθηκεύονται με τιμή 1.

```
public function getOpenLifts($result)
{
    $lifts = [];
    $newDoc = new DOMDocument();
    $newDoc->appendChild($newDoc->importNode($result, true));
    $newSelector = new DOMXPath($newDoc);

    // Εξαγωγή κόκκινων lifts
    $redLifts = $newSelector->query('//span[@class="lift-name red"]');
    foreach ($redLifts as $element) {
        $liftname = $element->textContent;
        $liftname = str_replace("\n", "", $liftname);
        $lifts[$liftname] = 0; // Κλειστό lift
    }

    // Εξαγωγή πράσινων lifts
    $greenLifts = $newSelector->query('//span[@class="lift-name green"]');
    foreach ($greenLifts as $element) {
        $liftname = $element->textContent;
        $liftname = str_replace("\n", "", $liftname);
        $lifts[$liftname] = 1; // Ανοιχτό lift
```

```
    }

    return $lifts;
}
```

Επιπλέων η κλάση isopen επεκτείνει την εντολή Command του Laravel. Η κύρια λειτουργία της εντολής είναι να κάνει scraping από τον ιστότοπο Snow Report και να ενημερώνει τη βάση δεδομένων σχετικά με την κατάσταση (ανοιχτό ή κλειστό) των χιονοδρομικών κέντρων.Αναλυτικά οι μέθοδοι οι από τις οποίες αποτελείται:

1. **Μέθοδος getSnowReportPage()**

   Αυτή η μέθοδος χρησιμοποιεί το cURL για να κατεβάσει τη σελίδα από το Snow Report

2. **Μέθοδος ScrapingPage($page)**

   Αυτή η μέθοδος αναλαμβάνει να αναλύσει την HTML της σελίδας. Χρησιμοποιεί το DOMDocument για να φορτώσει το HTML και το DOMXPath για να εντοπίσει τα στοιχεία που περιέχουν τις πληροφορίες για τα χιονοδρομικά κέντρα. Διαδικασία:

   - Μέθοδος ScΕντοπίζει τα χιονοδρομικά κέντρα με τη χρήση XPath (//ul[@class="submenu"]/li/a).rapingPage($page)
   - Αναλύει την κατάσταση του κάθε χιονοδρομικού κέντρου, ανάλογα με το χρώμα της γραμματοσειράς (κόκκινο = κλειστό, οποιοδήποτε άλλο χρώμα = ανοιχτό).
   - Αποθηκεύει το όνομα και την κατάσταση του κάθε χιονοδρομικού κέντρου σε έναν πίνακα $resorts.
   - Στη συνέχεια, καλεί τη μέθοδο StoreStatus για να αποθηκεύσει τις πληροφορίες στη βάση δεδομένων.

```
public function ScrapingPage($page): void
{
    libxml_use_internal_errors(true);

    $doc = new \DOMDocument();
    $doc->loadHTML($page);
    $selector = new \DOMXPath($doc);

    // Βρίσκει όλα τα στοιχεία του μενού που αντιστοιχούν στα χιονοδρομικά
κέντρα
    $resortNodes = $selector->query('//ul[@class="submenu"]/li/a');
    $i=0;
    foreach ($resortNodes as $node) {
        $url = $node->getAttribute('href');
        $statusNode = $node->getElementsByTagName('font')->item(0);
        $status = $statusNode->getAttribute('color') === 'red' ? 'Closed'
: 'Open';
```

```
        $resorts[] = [
            'name' =>  basename(parse_url($url, PHP_URL_PATH)),
            'status' => $status,
        ];
        if($i==21){
            break;
        }
        $i++;
    }

    $this->StoreStatus($resorts);
}
```

3. Μέθοδος StoreStatus($resorts)

Η μέθοδος StoreStatus είναι υπεύθυνη για την ενημέρωση της βάσης δεδομένων:

- Αναζητά στη βάση δεδομένων το όνομα του χιονοδρομικού κέντρου.
- Εάν το κέντρο υπάρχει, ενημερώνει την κατάσταση του (ανοιχτό ή κλειστό).
- Εάν δεν υπάρχει, προσθέτει νέα εγγραφή.

```
public function StoreStatus($resorts): void
{
    foreach ($resorts as $resort) {
        $existingResort = DB::table('snow_resorts')
            ->where('name_en', $resort['name'])
            ->first();

        if ($existingResort) {
            DB::table('snow_resorts')->updateOrInsert(
                ['name_en' => $resort['name']],
                ['status' => $resort['status']]
            );
        }
    }
}
```

Η κλάση isopen έχει ως κύριο στόχο τη λήψη δεδομένων από τον ιστότοπο Snow Report και την αποθήκευση της κατάστασης των χιονοδρομικών κέντρων στη βάση δεδομένων. Αυτή η εντολή μπορεί να εκτελείται περιοδικά για να ενημερώνει τα δεδομένα, διασφαλίζοντας έτσι την επικαιροποίηση της πληροφορίας.

1.6 Επίλογος

Στο πρώτο κεφάλαιο, παρουσιάστηκε το γενικό πλαίσιο και οι στόχοι της πτυχιακής εργασίας, καθώς και οι βασικές τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του API. Αναλύθηκαν οι μέθοδοι συλλογής δεδομένων (web scraping) και παρουσιάστηκαν οι κύριες τεχνολογίες που χρησιμοποιήθηκαν ώστε να αποτελεί ένα αξιόπιστο και αποδοτικό εργαλείο ώστε να είναι η βάση για διάφορες εφαρμογές (web app ,android app, iOS app). Ολοκληρώνοντας τον επίλογο, ο αναγνώστης έχει αποκτήσει μια σαφή εικόνα των βασικών αρχών και τεχνικών που διέπουν την ανάπτυξη του API.Τα επόμενα κεφάλαια θα εμβαθύνουν στις τεχνικές λεπτομέρειες και τα αποτελέσματα της πτυχιακής εργασίας, παρέχοντας μια πλήρη εικόνα της διαδικασίας και των αποτελεσμάτων της.

Κεφάλαιο 2ο: Ανάπτυξη και Υλοποίηση API

2.1 Εισαγωγή

Στο δεύτερο κεφάλαιο, θα αναλυθεί η διαδικασία ανάπτυξης και υλοποίησης του API, το οποίο δημιουργήθηκε για να παρέχει πληροφορίες σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας. Η διαδικασία περιλαμβάνει όλα τα στάδια από την υλοποίηση της βάσης δεδομένων ,την ενσωμάτωση των δεδομένων ,την ανάπτυξη του με laravel, την αρχιτεκτονική ,την ασφάλεια ,την φιλοξενία στο σέρβερ και τα τελικά endpoints. Ο στόχος αυτού του κεφαλαίου είναι να παράσχει μια λεπτομερή περιγραφή της τεχνικής προσέγγισης που ακολουθήθηκε για την κατασκευή του API. Με τον τρόπο αυτό, θα γίνει κατανοητή η πλήρης διαδικασία υλοποίησης και θα παρουσιαστούν τα εργαλεία και οι μέθοδοι που εξασφάλισαν την επιτυχή ολοκλήρωση του έργου.

2.2 Σχεδιασμός της Βάσης Δεδομένων

Ο σχεδιασμός της βάσης δεδομένων είναι ένα κρίσιμο στάδιο στην ανάπτυξη του API, καθώς καθορίζει τον τρόπο με τον οποίο τα δεδομένα θα αποθηκεύονται, θα οργανώνονται και θα ανακτώνται. Πριν από τον σχεδιασμό της βάσης δεδομένων, πραγματοποιήθηκε ανάλυση των απαιτήσεων που έπρεπε να καλυφθούν. Αυτές περιλάμβαναν τον καθορισμό των τύπων δεδομένων που θα αποθηκεύονται, όπως οι πληροφορίες για τα χιονοδρομικά κέντρα, τις πίστες, τις καιρικές συνθήκες, και τα δεδομένα χρήστη. Επιπλέον, αναλύθηκαν οι σχέσεις μεταξύ αυτών των δεδομένων, ώστε να εξασφαλιστεί η ακεραιότητα και η συνέπεια των πληροφοριών.

2.2.1Σχεδίαση Σχημάτων και Πινάκων

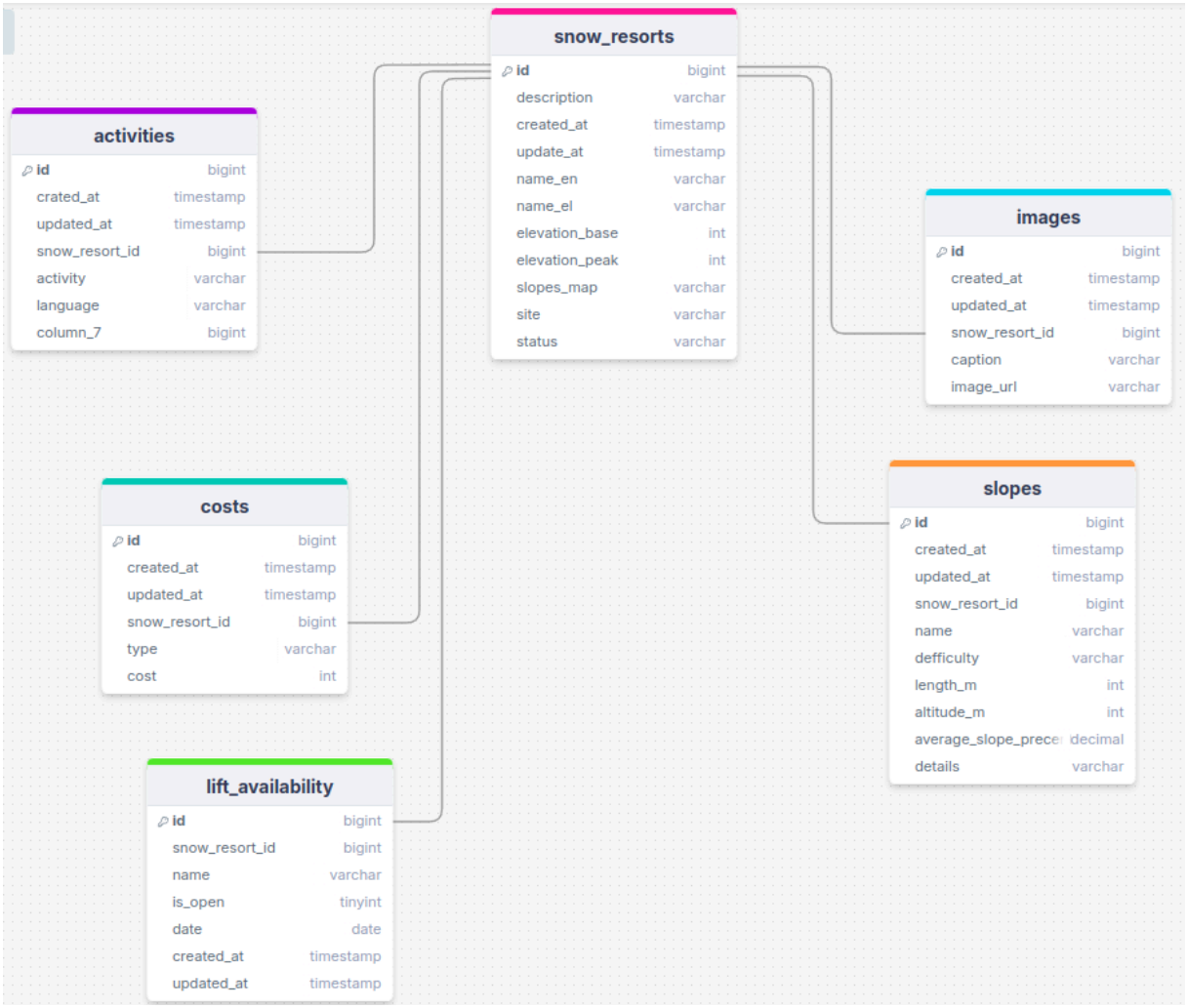
Με βάση την ανάλυση των απαιτήσεων, σχεδιάστηκαν τα σχήματα και οι πίνακες της βάσης δεδομένων. Η δομή της βάσης δεδομένων περιλαμβάνει πίνακες όπως:

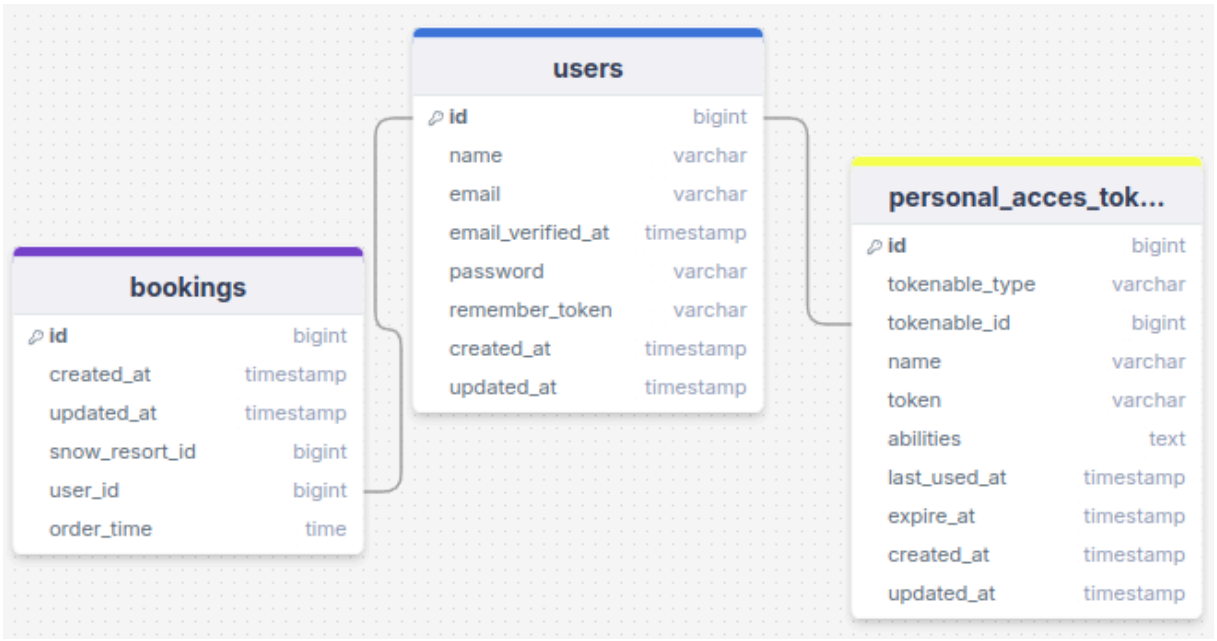
- snow_resorts: Περιέχει πληροφορίες όπως το όνομα, η τοποθεσία, η βάση και η κορυφή του χιονοδρομικού , γεωγραφικές συντεταγμένες , το στάτους του κέντρου και την επίσημη ιστοσελίδα του.

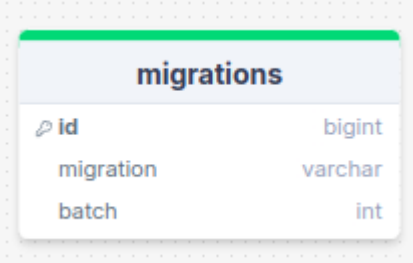
- activities:Καταγράφει δεδομένα στην ελληνική και στην αγγλική γλώσσα για τις δραστηριότητες του resort.
- bookings:Αποθηκεύει πληροφορίες που έχουν να κάνουν με την αγορά εισιτηρίου του χρήστη όπως την ώρα της αγοράς, το id του user και το id του εισιτηρίου .
- costs: Περιλαμβάνει τον τύπου του εισιτηρίου και το κοστος του .
- images:Περιέχει τα url από φωτογραφίες που χρησιμοποιούμε όπως και το caption της φωτογραφίας .
- lift_availability: Καταγράφει τους αναβατήρες για κάθε resort καθώς και το στάτους του και ανανεώνεται κάθε 10 λεπτά ώστε να μην κατέχει παρωχημένες πληροφορίες .
- migrations:Λειτουργία της laravel για την βάση.
- personal_access_tokens: Καταγράφονται ευαίσθητες πληροφορίες σχετικά με την πρόσβαση του χρήστη κατά την σύνδεση του στην εφαρμογή όπως tokenble_id κλπ.
- slopes:Αφορά τις πίστες όπου έχει το κάθε χιονοδρομικό με το όνομα τους ,το επίπεδο δυσκολίας , το μήκος, το ύψος που ξεκινάει ,την κλίση της και γενικές πληροφορίες για αυτήν.
- users:Αποθηκεύει πληροφορίες σχετικα με τον χρήστη όπως το όνομα ,το email και τον κωδικό του αποκρυπτογραφημένο.

Οι πίνακες συνδέονται μεταξύ τους μέσω σχέσεων, όπως οι σχέσεις "ένα προς πολλά" (one-to-many) μεταξύ των χιονοδρομικών κέντρων και πινάκων των πιστών , των αναβατήρων, των φωτογραφιών των κρατήσεων , των δραστηριοτήτων και αυτόν του κόστους. . Χρησιμοποιήθηκαν πρωτεύοντα και ξένα κλειδιά (primary and foreign keys) για τη διασφάλιση της ακεραιότητας των δεδομένων.

Ο σχεδιασμός της βάσης δεδομένων ήταν καθοριστικός για την επιτυχία του API, επιτρέποντας την αποθήκευση, ανάκτηση και διαχείριση δεδομένων με αποδοτικό και ασφαλή τρόπο. Οι παραπάνω αρχές και πρακτικές αποτέλεσαν τη βάση για τη δημιουργία ενός συστήματος που μπορεί να ανταποκριθεί στις απαιτήσεις των χρηστών και να προσφέρει αξιόπιστες υπηρεσίες.


snow_resorts
id bigint
description varchar
created_at timestamp
update_at timestamp
name_en varchar
name_el varchar
elevation_base int
elevation_peak int
slopes_map varchar
site varchar
status varchar
activities
id bigint
crated_at timestamp
updated_at timestamp
snow_resort_id bigint
activity varchar
language varchar
column_7 bigint
images
id bigint
created_at timestamp
updated_at timestamp
snow_resort_id bigint
caption varchar
image_url varchar
costs
id bigint
created_at timestamp
updated_at timestamp
snow_resort_id bigint
type varchar
cost int
slopes
id bigint
created_at timestamp
updated_at timestamp
snow_resort_id bigint
name varchar
defficulty varchar
length_m int
altitude_m int
average_slope_prece decimal
details varchar
lift_availability
id bigint
snow_resort_id bigint
name varchar
is_open tinyint
date date
created_at timestamp
updated_at timestamp


users
id bigint
name varchar
email varchar
email_verified_at timestamp
password varchar
remember_token varchar
created_at timestamp
updated_at timestamp
bookings
id bigint
created_at timestamp
updated_at timestamp
snow_resort_id bigint
user_id bigint
order_time time
personal_acces_tok...
id bigint
tokenable_type varchar
tokenable_id bigint
name varchar
token varchar
abilities text
last_used_at timestamp
expire_at timestamp
created_at timestamp
updated_at timestamp



Σχήμα 2.1: SnowHub database

2.3 Ανάπτυξη του API με Laravel

Η ανάπτυξη του API για τα χιονοδρομικά κέντρα της Ελλάδας πραγματοποιήθηκε με τη χρήση του Laravel, ενός δημοφιλούς PHP framework που προσφέρει εργαλεία και βιβλιοθήκες για την εύκολη και ασφαλή δημιουργία web εφαρμογών. Η επιλογή του Laravel έγινε λόγω των προηγμένων χαρακτηριστικών του, της υποστήριξής του για σύγχρονες πρακτικές ανάπτυξης λογισμικού, και της μεγάλης κοινότητας χρηστών που το υποστηρίζει αλλά και προσωπικής εμπειρίας .

2.3.1 Ρύθμιση και Διαμόρφωση του Laravel

Αρχικά, εγκαταστάθηκε το Laravel σε έναν τοπικό σύστημα. Η ρύθμιση περιελάμβανε τη διαμόρφωση των αρχείων περιβάλλοντος (.env) για τη σύνδεση με τη βάση δεδομένων MySQL και τη διαχείριση άλλων παραμέτρων, όπως οιρυθμίσεις email.Επιπλέον, καθορίστηκε η δομή του project, οργανώνοντας τους controllers, models, και views σε αντίστοιχους φακέλους, σύμφωνα με το πρότυπο MVC (Model-View-Controller) που προτείνει το Laravel.

2.3.2 Δημιουργία Μοντέλων και Μηχανισμών ORM

Η ανάπτυξη του API ξεκίνησε με τη δημιουργία των μοντέλων (models) που αντιπροσωπεύουν τις οντότητες της βάσης δεδομένων, όπως τα χιονοδρομικά κέντρα, οι πίστες, οι αναβατήρες ,φωτογραφίες κλπ. Με τη βοήθεια του Eloquent ORM (Object-Relational Mapping) του Laravel, τα μοντέλα συνδέθηκαν με τους αντίστοιχους πίνακες της βάσης δεδομένων, επιτρέποντας την εύκολη εκτέλεση ερωτημάτων SQL μέσω αντικειμενοστραφών μεθόδων. Παρακάτω παρατίθενται τα μοντέλα που δημιουργηθηκαν.

- Activities: Το μοντέλο αυτό αντιπροσωπεύει τις διάφορες δραστηριότητες που προσφέρονται στα χιονοδρομικά κέντρα, όπως σκι, snowboarding.
- Bookings: Το μοντέλο Bookings χρησιμοποιείται για τη διαχείριση των κρατήσεων των χρηστών. Περιλαμβάνει πληροφορίες όπως ο χρήστης που έκανε την κράτηση.
- Costs: Το μοντέλο αυτό περιέχει δεδομένα σχετικά με το κόστος των διαφόρων υπηρεσιών και πακέτων στα χιονοδρομικά κέντρα.
- images: Το μοντέλο Images αποθηκεύει και διαχειρίζεται τις εικόνες που σχετίζονται με τα χιονοδρομικά κέντρα

- LiftAvailability: Το μοντέλο αυτό αντιπροσωπεύει τη διαθεσιμότητα των αναβατήρων (lifts) στα χιονοδρομικά κέντρα. Καταγράφει πληροφορίες για το αν οι αναβατήρες είναι σε λειτουργία.
- Slopes: Το μοντέλο Slopes διαχειρίζεται τα δεδομένα των πιστών σκι, περιλαμβάνοντας πληροφορίες όπως το μήκος, η δυσκολία.
- SnowResorts: Το μοντέλο SnowResorts είναι το κεντρικό μοντέλο που αντιπροσωπεύει τα χιονοδρομικά κέντρα. Περιλαμβάνει γενικές πληροφορίες όπως το όνομα, η τοποθεσία, και άλλες σχετικές λεπτομέρειες που βοηθούν τους χρήστες να επιλέξουν το κέντρο που ταιριάζει στις ανάγκες τους.
- Users: Το μοντέλο Users διαχειρίζεται τα δεδομένα των χρηστών του API. Περιλαμβάνει πληροφορίες όπως το όνομα χρήστη, ο κωδικός πρόσβασης (αποθηκευμένος με κρυπτογράφηση), το email, και τα προσωπικά στοιχεία.

2.3.3 Ανάπτυξη Controllers

Οι controllers του API δημιουργήθηκαν για να χειρίζονται τα αιτήματα (requests) που λαμβάνει το API και να επιστρέφουν τις κατάλληλες απαντήσεις (responses). Κάθε controller σχεδιάστηκε για να διαχειρίζεται μια συγκεκριμένη λειτουργικότητα, όπως η ανάκτηση δεδομένων για τα χιονοδρομικά κέντρα. Χρησιμοποιήθηκαν RESTful πρακτικές, ορίζοντας διαδρομές (routes) για κάθε λειτουργία, όπως GET, POST,PUT, και DELETE, για την αλληλεπίδραση με τα δεδομένα.Οι controllers:

1. **ActivitiesController**

   Ο ActivitiesController οποίος είναι υπεύθυνος για τη διαχείριση των δραστηριοτήτων στα χιονοδρομικά κέντρα. Ο controller περιλαμβάνει τις ακόλουθες βασικές λειτουργίες:

   - **index()**

     Η μέθοδος index() αναλαμβάνει την επιστροφή όλων των δραστηριοτήτων (activities) που υπάρχουν στη βάση δεδομένων. Χρησιμοποιεί τη μέθοδο all() του μοντέλου Activities για να ανακτήσει όλα τα αρχεία και να τα επιστρέψει σε μορφή JSON.

```
public function index()
{
$slopes = Activities::all();
return response()->json($slopes);
}
```

   - **show(id)**

     Η μέθοδος show() είναι υπεύθυνη για την επιστροφή των λεπτομερειών μιας συγκεκριμένης δραστηριότητας μέσω του αναγνωριστικού της (ID). Αν η δραστηριότητα δεν βρεθεί, η μέθοδος χρησιμοποιεί το findOrFail(), που επιστρέφει σφάλμα αν δεν εντοπιστεί το αντίστοιχο αρχείο.

```
public function show($id)
{
$slope = Activities::findOrFail($id);
return response()->json($slope);
}
```

- **store(Request $request)**

  Η μέθοδος store() χειρίζεται την αποθήκευση μιας νέας δραστηριότητας. Αρχικά, γίνεται έλεγχος των δεδομένων μέσω της μεθόδου validate(), εξασφαλίζοντας πως η εισαγωγή περιλαμβάνει το snow_resort_id, την ονομασία της δραστηριότητας, και τη γλώσσα (language). Στη συνέχεια, δημιουργείται το νέο αρχείο και επιστρέφεται ως απάντηση με status 201.

```
public function store(Request $request)
{
$validatedData = $request->validate([
'snow_resort_id' => 'required|integer',
'activity' => 'required|string|max:255',
'language' => 'required|string|max:50',
]);

$slope = Activities::create($validatedData);
return response()->json($slope, 201);
}
```

Χρησιμοποιείται όταν ένας διαχειριστής ή εξουσιοδοτημένος χρήστης προσθέτει μια νέα δραστηριότητα για ένα χιονοδρομικό κέντρο.

- **destroy($id)**

  Η μέθοδος destroy() διαγράφει μια δραστηριότητα με βάση το αναγνωριστικό της. Εφόσον εντοπιστεί η δραστηριότητα μέσω της findOrFail(), διαγράφεται από τη βάση δεδομένων και επιστρέφεται κενή απάντηση με status 204.Αυτή η λειτουργία χρησιμοποιείται για τη διαγραφή μίας συγκεκριμένης δραστηριότητας, π.χ. εάν η δραστηριότητα δεν είναι πλέον διαθέσιμη.

```
public function destroy($id)
{
$slope = Activities::findOrFail($id);
$slope->delete();

return response()->json(null, 204);
}
```

2. BookingController

Σε αυτή την ενότητα, παρουσιάζεται ο BookingController, ο οποίος είναι υπεύθυνος για τη διαχείριση των κρατήσεων από τους χρήστες στο σύστημα. Ο controller περιλαμβάνει βασικές λειτουργίες, όπως την εμφάνιση και δημιουργία κρατήσεων, καθώς και τον έλεγχο του κόστους για κάθε χιονοδρομικό κέντρο. Η χρήση του CostsController ως εξαρτώμενη κλάση (dependency) μέσω του constructor ενισχύει τη λειτουργικότητα της εφαρμογής.Ο BookingController προσφέρει λειτουργίες για τη δημιουργία κρατήσεων από τους χρήστες και την επιβεβαίωση αυτών, ενσωματώνοντας ελέγχους κόστους και αποστολή email επιβεβαίωσης. Ο CostsController ενισχύει τη λειτουργικότητα αυτή μέσω ελέγχων κόστους.

- **__construct(CostsController $costsController)**

  Ο constructor της κλάσης BookingController δέχεται τον CostsController ως παράμετρο, κάτι που επιτρέπει την πρόσβαση και τον έλεγχο των δεδομένων κόστους για κάθε χιονοδρομικό κέντρο κατά τη διαδικασία της κράτησης, αυτή η μέθοδος ονομάζεται dependency injection.

```
public function __construct(CostsController $costsController)
{
$this->costsController = $costsController;
}
```

- **index()**

  Η μέθοδος index() είναι υπεύθυνη για την ανάκτηση και επιστροφή όλων των κρατήσεων που έχει πραγματοποιήσει ο συνδεδεμένος χρήστης. Χρησιμοποιείται η ταυτοποίηση του χρήστη μέσω της μεθόδου Auth::user() για να ληφθούν μόνο οι κρατήσεις που σχετίζονται με τον συγκεκριμένο χρήστη.Χρησιμοποιείται για την προβολή των κρατήσεων από τον συνδεδεμένο χρήστη.

```
public function index()
{
$user = Auth::user();
$bookings = Bookings::where('user_id', $user->id)->get();
return response()->json($bookings);
}
```

- **store(Request $request)**

  Η μέθοδος store() επιτρέπει στον χρήστη να πραγματοποιήσει νέες κρατήσεις σε ένα χιονοδρομικό κέντρο. Τα δεδομένα που λαμβάνονται από το αίτημα υποβάλλονται σε έλεγχο με χρήση της μεθόδου validate(). Στη συνέχεια, για κάθε κράτηση, ελέγχεται το κόστος με τη βοήθεια της μεθόδου validateCost() και αποστέλλεται ένα επιβεβαιωτικό email.

```
      public function store(Request $request)
{
$request->validate([
```

```
'snow_resort_id' => 'required|integer',
'cost' => 'required|integer',
'number_pass' => 'required|integer|min:1'
]);

$user = Auth::user();

if (!$this->validateCost($request->input('snow_resort_id'),
$request->input('cost'))) {
return response()->json(['message' => 'Something went wrong'], 400);
}

$number_pass = $request->input('number_pass');
$createdBookings = [];

for ($i = 1; $i <= $number_pass; $i++) {
$booking = new Bookings();
$booking->snow_resort_id = $request->input('snow_resort_id');
$booking->user_id = $user->id;
$booking->order_time = now();
$booking->save();
$createdBookings[] = $booking;

try {
Mail::to($user->email)->send(new BookingConfirmation($booking));
} catch (\Exception $e) {
Log::error('Failed to send booking confirmation email: ' .
$e->getMessage());
}
}

return response()->json([
'message' => 'Bookings created successfully',
'bookings' => $createdBookings
], 201);
}
```

- **validateCost($snow_resort_id, $requestCost)**

  Η μέθοδος validateCost() ελέγχει αν το κόστος που έχει εισαχθεί από τον χρήστη είναι έγκυρο για το συγκεκριμένο χιονοδρομικό κέντρο. Χρησιμοποιείται ο CostsController για την ανάκτηση των σχετικών δεδομένων κόστους για τον έλεγχο της τιμής σε σχέση με το πραγματικό κόστος.

```
public function validateCost($snow_resort_id, $requestCost): bool
{
$costData =
```

```
$this->costsController->getBySnowResortId($snow_resort_id)->getData();
foreach ($costData as $cost) {
if ($cost->type == 'kanoniko' && $cost->cost == $requestCost) {
return true;
}
}
return false;
}
```

3. **Cost Controller**

O CostsController είναι υπεύθυνος για τη διαχείριση των δεδομένων κόστους σε σχέση με τα χιονοδρομικά κέντρα. Ο controller παρέχει λειτουργίες για την προβολή, δημιουργία, ενημέρωση και διαγραφή τιμών κόστους, καθώς και για την ανάκτηση των δεδομένων κόστους για συγκεκριμένο χιονοδρομικό κέντρο. Χρησιμοποιείται για την επιστροφή των τιμών κόστους για συγκεκριμένο χιονοδρομικό κέντρο. Ο CostsController προσφέρει πλήρη διαχείριση των δεδομένων κόστους, συμπεριλαμβανομένης της δημιουργίας, προβολής, ενημέρωσης και διαγραφής κόστους. Επίσης, παρέχει τη δυνατότητα να ανακτήσει κανείς τα κόστη για ένα συγκεκριμένο χιονοδρομικό κέντρο.

- **index()**

  Η μέθοδος index() επιστρέφει όλα τα κόστη που έχουν καταχωρηθεί στη βάση δεδομένων. Χρησιμοποιείται το μοντέλο Costs για την ανάκτηση αυτών των δεδομένων και την επιστροφή τους σε μορφή JSON.

```
public function index()
{
    $costs = Costs::all();
    return response()->json($costs);
}
```

- **show($id)**

  Η μέθοδος show($id) επιστρέφει τα δεδομένα κόστους για ένα συγκεκριμένο κόστος, βασισμένο στο ID του. Χρησιμοποιείται η μέθοδος findOrFail() του μοντέλου Costs για να εξασφαλιστεί ότι το κόστος υπάρχει.

```
public function show($id)
{
    $costs = Costs::findOrFail($id);
    return response()->json($costs);
}
```

- **store(Request $request)**

Η μέθοδος store() επιτρέπει την προσθήκη νέων τιμών κόστους στο σύστημα. Τα δεδομένα του αιτήματος υποβάλλονται σε έλεγχο με τη χρήση της μεθόδου validate(). Αν το αίτημα είναι έγκυρο, το νέο κόστος αποθηκεύεται στη βάση δεδομένων.

```
public function store(Request $request)
{
    $validatedData = $request->validate([
        'snow_resort_id' => 'required|integer',
        'type' => 'required|string|max:255',
        'costs' => 'required|integer',
    ]);

    $costs = Costs::create($validatedData);
    return response()->json($costs, 201);
}
```

- **update(Request $request, $id)**

  Η μέθοδος update() επιτρέπει την ενημέρωση δεδομένων κόστους για ένα συγκεκριμένο κόστος, βάσει του ID. Τα δεδομένα εισάγονται και ελέγχονται πριν από την ενημέρωση των τιμών κόστους.

```
public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'snow_resort_id' => 'required|integer',
        'type' => 'required|string|max:255',
        'costs' => 'required|integer',
    ]);

    $costs = Costs::findOrFail($id);
    $costs->update($validatedData);

    return response()->json($costs);
}
```

- **destroy($id)**

  Η μέθοδος destroy() επιτρέπει τη διαγραφή ενός συγκεκριμένου κόστους από το σύστημα. Χρησιμοποιείται η μέθοδος findOrFail() για να βρεθεί το κόστος και στη συνέχεια να διαγραφεί.

```
      public function destroy($id)
{
    $costs = Costs::findOrFail($id);
    $costs->delete();
```

```
    return response()->json(null, 204);
}
```

- **getBySnowResortId($id)**

  Η μέθοδος getBySnowResortId($id) επιτρέπει την ανάκτηση όλων των τιμών κόστους για ένα συγκεκριμένο χιονοδρομικό κέντρο βάσει του snow_resort_id. Επιστρέφει τα αποτελέσματα σε μορφή JSON.

```
      public function getBySnowResortId($id)
{
    $costs = Costs::where('snow_resort_id', $id)->get();
    return response()->json($costs);
}
```

4. **ImagesController**

   Ο ImagesController είναι υπεύθυνος για τη διαχείριση των εικόνων που σχετίζονται με τα χιονοδρομικά κέντρα. Περιλαμβάνει λειτουργίες για την προβολή όλων των εικόνων, την εμφάνιση συγκεκριμένης εικόνας βάσει ID, την προσθήκη νέας εικόνας και τη διαγραφή υφιστάμενης εικόνας.Ο ImagesController παρέχει λειτουργικότητα για τη διαχείριση εικόνων, συμπεριλαμβανομένης της καταχώρησης, προβολής, και διαγραφής εικόνων.Ο ImagesController παρέχει λειτουργικότητα για τη διαχείριση εικόνων, συμπεριλαμβανομένης της καταχώρησης, προβολής, και διαγραφής εικόνων.

   - **index()**

     Η μέθοδος index() επιστρέφει όλες τις εικόνες που έχουν καταχωρηθεί στη βάση δεδομένων. Χρησιμοποιείται το μοντέλο Images για την ανάκτηση αυτών των δεδομένων και την επιστροφή τους σε μορφή JSON.

```
public function index()
{
    $images = Images::all();
    return response()->json($images);
}
```

   - **show($id)**

     Η μέθοδος show($id) επιστρέφει μια συγκεκριμένη εικόνα βάσει του ID της. Αν δεν βρεθεί η εικόνα, επιστρέφεται μήνυμα λάθους με κωδικό 404. Χρησιμοποιείται η μέθοδος findOrFail() για τον εντοπισμό της εικόνας.

```
      {
    try {
        $image = Images::findOrFail($id);
```

```
        return response()->json($image);
    } catch (ModelNotFoundException $e) {
        return response()->json(['error' => 'Image not found'], 404);
    }
}
```

- **store(Request $request)**

  Η μέθοδος store() επιτρέπει την καταχώρηση νέας εικόνας. Τα δεδομένα εισάγονται και ελέγχονται με τη μέθοδο validate(). Αν το αίτημα είναι έγκυρο, η νέα εικόνα αποθηκεύεται στη βάση δεδομένων.Χρησιμοποιείται για την προσθήκη νέας εικόνας, με έλεγχο εγκυρότητας στα δεδομένα.

```
public function store(Request $request)
{
    $validatedData = $request->validate([
        'snow_resort_id' => 'required|integer',
        'image_url' => 'required|string',
        'caption' => 'required|string',
    ]);

    $image = Images::create($validatedData);
    return response()->json($image, 201);
}
```

- destroy($id)

  Η μέθοδος destroy($id) επιτρέπει τη διαγραφή μιας εικόνας βάσει του ID της. Αν η εικόνα δεν βρεθεί, επιστρέφεται μήνυμα λάθους με κωδικό 404. Αν βρεθεί, η εικόνα διαγράφεται.

```
public function destroy($id)
{
    try {
        $image = Images::findOrFail($id);
        $image->delete();
        return response()->json(null, 204);
    } catch (ModelNotFoundException $e) {
        return response()->json(['error' => 'Image not found'], 404);
    }
}
```

5. **LiftAvailabilityController**

   Ο LiftAvailabilityController είναι υπεύθυνος για τη διαχείριση της διαθεσιμότητας των αναβατήρων (lifts) σε χιονοδρομικά κέντρα. Περιλαμβάνει λειτουργίες για την εμφάνιση της διαθεσιμότητας των αναβατήρων, καθώς και τη δημιουργία εγγραφών για τη διαθεσιμότητά τους μέσω scraping από εξωτερικές πηγές. Ο LiftAvailabilityController παρέχει λειτουργικότητα για την παρακολούθηση της διαθεσιμότητας των αναβατήρων σε χιονοδρομικά κέντρα, με δυνατότητα άντλησης πληροφοριών μέσω scraping σε περίπτωση έλλειψης δεδομένων.

   - **index($snowResortId)**

     Η μέθοδος index() είναι υπεύθυνη για την εμφάνιση της διαθεσιμότητας των αναβατήρων για ένα συγκεκριμένο χιονοδρομικό κέντρο που καθορίζεται από το snowResortId. Πρώτα, καλεί τη μέθοδο show() του SnowResortController για να πάρει τις λεπτομέρειες του χιονοδρομικού κέντρου. Αν το χιονοδρομικό δεν βρεθεί, επιστρέφεται μήνυμα σφάλματος με κωδικό 404. Στη συνέχεια, γίνεται έλεγχος αν υπάρχουν διαθέσιμες πληροφορίες για τους αναβατήρες στη βάση δεδομένων. Αν δεν υπάρχουν, πραγματοποιείται scraping μέσω της κλάσης Scraping για την απόκτηση των δεδομένων. Τέλος, επιστρέφονται τα δεδομένα διαθεσιμότητας των αναβατήρων σε μορφή JSON.

```
public function index($snowResortId)
{
    $resort = $this->snowResortController->show($snowResortId);
    if ($resort->status() == 404) {
        return response()->json(['message' => 'snow resort not found'],
404);
    }

    $resortData = $resort->getData();
    $liftAvailability = LiftAvailability::where('snow_resort_id',
$snowResortId)->get();

    if ($liftAvailability->isEmpty()) {
        $scraping = new Scraping();
        $lifts = $scraping->getSnowReportPage($resortData->name_en);

        $this->store($lifts['today'], $snowResortId);
    }

    $liftAvailability = LiftAvailability::where('snow_resort_id',
$snowResortId)->get();
    return response()->json($liftAvailability);
}
```

- **store($lifts, $snowResortId)**

  Η μέθοδος store() αποθηκεύει τα δεδομένα διαθεσιμότητας των αναβατήρων που λαμβάνονται μέσω scraping. Η μέθοδος λαμβάνει έναν πίνακα με τα δεδομένα των αναβατήρων ($lifts) και το snowResortId. Δημιουργεί εγγραφές για κάθε αναβατήρα, αποθηκεύοντας τη διαθεσιμότητα στη βάση δεδομένων. Η ημερομηνία και ώρα καταγραφής αποθηκεύεται στη βάση μέσω της τρέχουσας ημερομηνίας. Αποθηκεύει κάθε αναβατήρα με το αν είναι ανοιχτός (is_open) ή κλειστός, μαζί με το όνομα και την ημερομηνία. Επιστρέφει επιτυχώς τα δεδομένα που δημιουργήθηκαν με κωδικό 201.

```
public function store($lifts, $snowResortId)
{
    $data = [];
    foreach ($lifts as $key => $value) {
        $data = [
            'snow_resort_id' => $snowResortId,
            'is_open' => $value,
            'name' => $key,
            'date' => date('Y-m-d H:i:s'),
        ];
        $availability = LiftAvailability::create($data);
        $data = [];
    }
    return response()->json($availability, 201);}
```

6. **SlopesController**

   Ο SlopesController είναι υπεύθυνος για τη διαχείριση των δεδομένων που αφορούν τις πίστες χιονοδρομικών κέντρων. Παρέχει λειτουργίες για εμφάνιση όλων των πιστών, εμφάνιση συγκεκριμένης πίστας, δημιουργία, ενημέρωση και διαγραφή.

   - **index()**

     Η μέθοδος index() επιστρέφει μια λίστα με όλες τις πίστες χιονοδρομικών κέντρων.

```
public function index()
{
    $slopes = Slopes::all();
    return response()->json($slopes);
}
```

   - **show($id)**

     Η μέθοδος show() επιστρέφει τα δεδομένα για μια συγκεκριμένη πίστα με βάση το id της πίστας.

```
public function show($id)
{
    $slope = Slopes::findOrFail($id);
    return response()->json($slope);
}
```

- **store(Request $request)**

  Η μέθοδος store() είναι υπεύθυνη για τη δημιουργία μιας νέας πίστας στη βάση δεδομένων.Ελέγχει τα εισερχόμενα δεδομένα με τη μέθοδο validate(). Δημιουργεί μια νέα πίστα με τα επικυρωμένα δεδομένα και επιστρέφει την πίστα που δημιουργήθηκε με κωδικό 201 (δημιουργήθηκε επιτυχώς).

```
public function store(Request $request)
{
    $validatedData = $request->validate([
        'snow_resort_id' => 'required|integer',
        'name' => 'required|string|max:255',
        'difficulty' => 'nullable|string|max:50',
        'length_m' => 'required|integer',
        'altitude_m' => 'nullable|integer',
        'average_slope_percent' => 'nullable|numeric',
        'details' => 'nullable|string',
    ]);

    $slope = Slopes::create($validatedData);
    return response()->json($slope, 201);
}
```

- **update(Request $request, $id)**

  Η μέθοδος update() ενημερώνει τα δεδομένα μιας υπάρχουσας πίστας. Επικυρώνει τα δεδομένα και ενημερώνει την πίστα που αντιστοιχεί στο id. Επιστρέφει τα ενημερωμένα δεδομένα της πίστας.

```
public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'resort_id' => 'required|integer',
        'name' => 'required|string|max:255',
        'difficulty' => 'nullable|string|max:50',
        'length_m' => 'required|integer',
        'altitude_m' => 'nullable|integer',
        'average_slope_percent' => 'nullable|numeric',
        'details' => 'nullable|string',
    ]);
```

```
    $slope = Slopes::findOrFail($id);
    $slope->update($validatedData);

    return response()->json($slope);
}
```

- **destroy($id)**

  Η μέθοδος destroy() διαγράφει μια πίστα με βάση το id. Διαγράφει την πίστα και επιστρέφει κωδικό 204 (χωρίς περιεχόμενο) για επιτυχία.

```
public function destroy($id)
{
    $slope = Slopes::findOrFail($id);
    $slope->delete();

    return response()->json(null, 204);
}
```

7. **SnowResortController**

   Ο SnowResortController είναι υπεύθυνος για τη διαχείριση των χιονοδρομικών κέντρων, συνδυάζοντας δεδομένα από πίστες, δραστηριότητες και εικόνες για κάθε χιονοδρομικό κέντρο. Παρέχει λειτουργίες για την εμφάνιση, δημιουργία και διαγραφή χιονοδρομικών κέντρων .Ο SnowResortController διαχειρίζεται όλες τις απαραίτητες λειτουργίες για τα χιονοδρομικά κέντρα και συνδυάζει δεδομένα από διάφορους άλλους controllers για ολοκληρωμένες πληροφορίες.

   - **index()**

     Η μέθοδος index επιστρέφει όλα τα χιονοδρομικά κέντρα με τις σχετικές πληροφορίες όπως πίστες, δραστηριότητες και εικόνες.Συλλέγει δεδομένα από τους SlopesController, ActivitiesController και ImagesController. Ενσωματώνει δραστηριότητες (ανά γλώσσα), πίστες και εικόνες για κάθε χιονοδρομικό κέντρο. Καλεί τη μέθοδο transformResortsArray() για να μορφοποιήσει τα δεδομένα.

```
public function index()
{
    $slopes = $this->slopesController->index()->getData();
    $activities = $this->activitiesController->index()->getData();
    $images = $this->imagesController->index()->getData();
    $snowResorts = SnowResorts::all();
```

```
    foreach ($snowResorts as $resort) {
        $resortImages = [];
        $resortSlopes = [];
        $resortActivitiesEn = [];
        $resortActivitiesEl = [];
        $resortActivities = [];

        foreach ($activities as $activity) {
            if ($resort->id == $activity->snow_resort_id) {
                if ($activity->language == 'en') {
                    $resortActivitiesEn[] = $activity;
                } else {
                    $resortActivitiesEl[] = $activity;
                }
            }
        }

        $resortActivities['en'] = $resortActivitiesEn;
        $resortActivities['el'] = $resortActivitiesEl;
        $resort['activities'] = $resortActivities;

        foreach ($slopes as $slope) {
            if ($resort->id == $slope->snow_resort_id) {
                $resortSlopes[] = $slope;
            }
        }

        $resort['slopes'] = $resortSlopes;

        foreach ($images as $image) {
            if ($resort->id == $image->snow_resort_id) {
                $resortImages[] = $image;
            }
        }

        $resort['images'] = $resortImages;
    }

    $snowResorts = $this->transformResortsArray($snowResorts);

    return response()->json($snowResorts);
}
```

- **store(Request $request)**

Η μέθοδος store() είναι υπεύθυνη για τη δημιουργία ενός νέου χιονοδρομικού κέντρου. Επικυρώνει τα εισερχόμενα δεδομένα και δημιουργεί ένα νέο χιονοδρομικό κέντρο. Επιστρέφει το χιονοδρομικό κέντρο που δημιουργήθηκε με κωδικό 201 (δημιουργήθηκε επιτυχώς).

```
public function store(Request $request)
{
    $data = $request->validate([
        'name' => 'required',
        'location' => 'required',
    ]);

    $snowResort = SnowResorts::create($data);

    return response()->json($snowResort, 201);
}
```

- **show($id)**

  Η μέθοδος show() επιστρέφει τα δεδομένα ενός συγκεκριμένου χιονοδρομικού κέντρου με βάση το id. Αναζητά το χιονοδρομικό κέντρο με το δεδομένο id. Αν δεν βρεθεί, επιστρέφει σφάλμα 404.

```
public function show($id)
{
    $snowResort = SnowResorts::find($id);

    if (!$snowResort) {
        return response()->json(['message' => 'Snow resort not found'],
404);
    }

    return response()->json($snowResort);
}
```

- **destroy($id)**

  Η μέθοδος destroy() διαγράφει ένα χιονοδρομικό κέντρο με βάση το id.

```
public function destroy($id)
{
    $snowResort = SnowResorts::find($id);

    if (!$snowResort) {
        return response()->json(['message' => 'Snow resort not found'],
404);
```

```
    }

    try {
        $snowResort->delete();
        return response()->json(['message' => 'Snow resort deleted
successfully'], 200);
    } catch (\Exception $e) {
        return response()->json(['message' => 'Error deleting record'],
500);
    }
}
```

- **transformResortsArray($snowResorts)**

  Η μέθοδος transformResortsArray() μορφοποιεί τα δεδομένα των χιονοδρομικών κέντρων για να περιλαμβάνουν τις γλώσσες, το υψόμετρο και άλλες πληροφορίες με διαφορετική δομή.

```
protected function transformResortsArray($snowResorts)
{
    return $snowResorts->map(function ($resort) {
        $resort->name = [
            'el' => $resort->name_el,
            'en' => $resort->name_en,
        ];
        $resort->elevation = [
            'base' => $resort->elevation_base,
            'peak' => $resort->elevation_peak,
        ];
        unset($resort->elevation_base);
        unset($resort->elevation_peak);
        unset($resort->name_el);
        unset($resort->name_en);
        return $resort;
    });
}
```

8. **UserController**

   Ο UserController διαχειρίζεται τις βασικές λειτουργίες εγγραφής, σύνδεσης, αποσύνδεσης και επιστροφής των στοιχείων του αυθεντικοποιημένου χρήστη. Αυτός ο controller καλύπτει τις βασικές λειτουργίες ενός συστήματος χρήστη: εγγραφή, σύνδεση, αποσύνδεση και επιστροφή των στοιχείων του αυθεντικοποιημένου χρήστη, χρησιμοποιώντας tokens για την αυθεντικοποίηση.

   - **register(Request $request)**

Η μέθοδος αυτή χρησιμοποιείται για την εγγραφή ενός νέου χρήστη. Ελέγχει αν όλα τα πεδία είναι σωστά (όνομα, email, κωδικός).Ο κωδικός χρήστη αποθηκεύεται κρυπτογραφημένος χρησιμοποιώντας το Hash::make(). Αν η εγγραφή ολοκληρωθεί επιτυχώς, επιστρέφεται μήνυμα επιτυχίας με κωδικό 201.

```
public function register(Request $request)
{
    $request->validate([
        'name' => 'required|string',
        'email' => 'required|string|email|unique:users',
        'password' => 'required|string|min:8',
    ]);

    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
    ]);

    return response()->json(['message' => 'User registered
successfully'], 201);
}
```

- **login(Request $request)**

  Αυτή η μέθοδος χρησιμοποιείται για τη σύνδεση του χρήστη και την παροχή ενός token αυθεντικοποίησης. Ελέγχει αν τα πεδία email και κωδικός είναι έγκυρα. Αν τα στοιχεία είναι σωστά, δημιουργείται ένα token για τον χρήστη μέσω της μεθόδου createToken().

```
public function login(Request $request): \Illuminate\Http\JsonResponse
{
    $request->validate([
        'email' => 'required|string|email',
        'password' => 'required|string',
    ]);

    if (!Auth::attempt($request->only('email', 'password'))) {
        throw new AuthenticationException('The provided credentials are
incorrect.');
    }

    $user = Auth::user();
    $token = $user->createToken('auth_token')->plainTextToken;

    return response()->json(['token' => $token], 200);
```

```
}
```

- **logout(Request $request)**

  Η μέθοδος αυτή χρησιμοποιείται για την αποσύνδεση του χρήστη, ανακαλώντας το token του. Όλα τα tokens του χρήστη διαγράφονται, πράγμα που σημαίνει ότι ο χρήστης αποσυνδέεται.

```
public function logout(Request $request)
{
    $request->user()->tokens()->delete();

    return response()->json(['message' => 'Successfully logged out'],
200);
}
```

- **user(Request $request)**

  Αυτή η μέθοδος χρησιμοποιείται για να επιστρέψει τα στοιχεία του αυθεντικοποιημένου χρήστη.

```
public function user(Request $request)
{
    return response()->json($request->user());
}
```

2.3.4 Δημιουργία και Διαχείριση Routes

Το Laravel Route system χρησιμοποιήθηκε για την καθοδήγηση των αιτημάτων στις κατάλληλες μεθόδους των controllers. Οι routes καθορίστηκαν στο αρχείο api.php για τις διαδρομές του API. Οι routes αυτές επιτρέπουν στους χρήστες να επικοινωνούν με το API μέσω των κατάλληλων endpoints, όπως /api/SnowResorts για την ανάκτηση δεδομένων για όλα τα χιονοδρομικά κέντρα /api/SnowResort/{snowResortId} για την ανάκτηση δεδομένων συγκεκριμμένου κέντρου.

```
Route::post( uri: '/register', [\App\Http\Controllers\UserController::class, 'register']);
Route::post( uri: '/login', [\App\Http\Controllers\UserController::class, 'login']);

Route::middleware( middleware: 'auth:sanctum')->group(function () {
    Route::post( uri: '/logout', [\App\Http\Controllers\UserController::class, 'logout']);
    Route::get( uri: '/user', [\App\Http\Controllers\UserController::class, 'user']);
    Route::post( uri: 'booking', [\App\Http\Controllers\BookingController::class, 'store']);
    Route::get( uri: 'mybooking', [\App\Http\Controllers\BookingController::class, 'index']);
```

Σχήμα 2.2 routes

2.3.5 Authentication και Authorization

Για την υλοποίηση του μηχανισμού αυθεντικοποίησης και εξουσιοδότησης (authentication και authorization) στο API, χρησιμοποιήθηκε το Laravel Sanctum. Το Sanctum προσφέρει ευέλικτη λύση για την ασφαλή διαχείριση της πρόσβασης των χρηστών σε προστατευμένα endpoints του API. Το Laravel Sanctum επιλέχθηκε για την υλοποίηση της αυθεντικοποίησης λόγω της απλότητας και της ευελιξίας του. Σε αντίθεση με άλλα συστήματα όπως το Laravel Passport, το Sanctum επιτρέπει τόσο την αυθεντικοποίηση session-based για παραδοσιακές εφαρμογές web, όσο και την token-based αυθεντικοποίηση για single-page applications (SPAs) ή mobile εφαρμογές. Με το Sanctum, κάθε χρήστης μπορεί να δημιουργήσει ένα ή περισσότερα API tokens, τα οποία χρησιμοποιούνται για την αυθεντικοποίηση του στις διάφορες λειτουργίες του API.

2.3.6 Τεστ και Επικύρωση Λειτουργιών

Κατά τη διάρκεια της ανάπτυξης, κάθε λειτουργία του API δοκιμάστηκε σχολαστικά χρησιμοποιώντας εργαλεία όπως το Postman για API testing. Οι δοκιμές αυτές περιλάμβαναν την επικύρωση των αιτημάτων και των απαντήσεων, την αξιολόγηση της απόδοσης των endpoints και τη διασφάλιση της ορθής διαχείρισης σφαλμάτων.

2.4 Φιλοξενία σε VPS

Για την παραγωγική φιλοξενία του API που αναπτύχθηκε, επιλέχθηκε η χρήση ενός Virtual Private Server (VPS) στη DigitalOcean. Η επιλογή αυτή έγινε για να εξασφαλιστεί η απαραίτητη ευελιξία, η απόδοση και η ασφάλεια της εφαρμογής, παρέχοντας ταυτόχρονα τον έλεγχο και τις δυνατότητες προσαρμογής που δεν προσφέρουν οι παραδοσιακές υπηρεσίες φιλοξενίας. Η αρχική ρύθμιση του VPS περιλάμβανε την επιλογή του κατάλληλου πακέτου που θα μπορούσε να υποστηρίξει το API, λαμβάνοντας υπόψη παράγοντες όπως η αναμενόμενη κίνηση, οι ανάγκες αποθήκευσης και οι απαιτήσεις επεξεργαστικής ισχύος επιλέχθηκε ένα πακέτο με 1 gb ram με δυνατότητα transfer 1000Gib και ssd 25 Gib. Ο επιλεγμένος server ρυθμίστηκε με τις εξής προδιαγραφές:

- **Λειτουργικό Σύστημα**: Debian 12, επιλεγμένο για τη σταθερότητα και την υποστήριξη που παρέχει.
- **Εγκατάσταση Web Server**: Επιλέχθηκε ο Nginx για τη διαχείριση των αιτημάτων HTTP προς το API.
- **Ρύθμιση PHP και MySQL**: Εγκαταστάθηκε η τελευταία έκδοση της PHP (8.3) μαζί με τον MySQL server για την υποστήριξη της βάσης δεδομένων.

2.5 Nginx: Χρήση και Ρύθμιση

Το Nginx είναι ένας εξαιρετικά αποδοτικός web server που χρησιμοποιείται ευρέως για την εξυπηρέτηση στατικών αρχείων, την προώθηση αιτημάτων σε backend εφαρμογές, και την εφαρμογή ασφαλείας σε ένα web project. Στο πλαίσιο του SnowHub API, ο Nginx λειτουργεί τόσο για την

εξυπηρέτηση του frontend μέρους της εφαρμογής (την Angular SPA) όσο και για την προώθηση των API αιτημάτων στην backend εφαρμογή Laravel.

1. Server Name: Ορίστηκαν τα ονόματα τομέα snowhub.gr και www.snowhub.gr για την αναγνώριση των αιτημάτων προς τον server.
2. Root για το Angular App: Ορίζεται ο φάκελος όπου βρίσκεται η Angular εφαρμογή, στην τοποθεσία /var/www/html/snowhub, έτσι ώστε να εξυπηρετούνται τα στατικά αρχεία και το index.html.
3. Security Headers: Προστίθενται βασικές κεφαλίδες ασφαλείας όπως:
    - X-Frame-Options: Αποτρέπει την ενσωμάτωση της σελίδας σε iframe για προστασία από clickjacking επιθέσεις.
    - X-Content-Type-Options: Αποτρέπει το browser από το να μαντεύει τον τύπο αρχείων.
    - X-XSS-Protection: Προστασία από cross-site scripting (XSS) επιθέσεις.
    - Strict-Transport-Security: Υποχρεώνει τη χρήση HTTPS για τα επόμενα 31536000 δευτερόλεπτα (1 έτος) και περιλαμβάνει τα subdomains.
4. Προώθηση API αιτημάτων: Για αιτήματα προς το /api/, ορίζεται η προώθηση στο backend Laravel API. Τα υπόλοιπα αιτήματα που δεν αφορούν το API, προωθούνται στον Angular frontend
5. SSL/HTTPS Διαμόρφωση: Η χρήση του Nginx για την ασφαλή εξυπηρέτηση αιτημάτων γίνεται μέσω του πρωτοκόλλου HTTPS. με SSL.

```
    root /var/www/html/snowhub;

    # Security Headers
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";
    add_header X-XSS-Protection "1; mode=block";
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

   index index.html;
    charset utf-8;

    # Handle API requests with Laravel
    location /api/ {
        alias /srv/app/GreekSnowResortsApi/public/;
        try_files $uri $uri/ /index.php?$query_string;
    }

    # Serve Angular frontend
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Handle PHP files
    location ~ \.php$ {
        root /srv/app/GreekSnowResortsApi/public;  # Ensure Laravel's root is used here
        fastcgi_pass unix:/var/run/php/php8.3-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }


    # Handle static files
    location ~* \.(jpg|jpeg|gif|css|png|js|ico)$ {
        try_files $uri =404;
        access_log off;
        expires max;
    }

    # Security: deny access to hidden files
    location ~ /\.(?!well-known).* {
        deny all;
    }
```

Σχήμα 2.3 nginx configuration

Η διαμόρφωση του Nginx εξυπηρετεί τόσο το frontend όσο και το backend κομμάτι της εφαρμογής SnowHub, ενώ ταυτόχρονα φροντίζει για την ασφάλεια μέσω HTTPS και των κατάλληλων κεφαλίδων ασφαλείας. Επιπλέον, με την ορθή διαχείριση των αιτημάτων και των σφαλμάτων, επιτυγχάνεται βελτιστοποίηση στην απόδοση και εξασφαλίζεται μια ασφαλής και σταθερή εμπειρία για τους χρήστες της πλατφόρμας.

2.6 Επίλογος

Στο κεφάλαιο αυτό, παρουσιάστηκαν οι κύριες διαδικασίες και τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη και υλοποίηση του API, από τον σχεδιασμό της βάσης δεδομένων μέχρι την φιλοξενία της εφαρμογής σε VPS. Αναλύθηκε ο τρόπος με τον οποίο κάθε στοιχείο συμβάλλει στη

συνολική λειτουργικότητα και απόδοση του API, ενώ παράλληλα δόθηκε έμφαση στις μεθόδους διαχείρισης και ασφάλειας των δεδομένων.

Η ανάπτυξη του API με το Laravel πρόσφερε μια ευέλικτη πλατφόρμα για τη διαχείριση πληροφοριών σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας. Η επιλογή ενός VPS για τη φιλοξενία του συστήματος επέτρεψε την πλήρη προσαρμογή του περιβάλλοντος φιλοξενίας στις ανάγκες της εφαρμογής, διασφαλίζοντας την απόδοση, την ασφάλεια και τη δυνατότητα μελλοντικής επέκτασης.

Με την ολοκλήρωση αυτού του κεφαλαίου, καθίσταται σαφές πως οι επιλογές που έγιναν κατά την ανάπτυξη και υλοποίηση του API, όχι μόνο ικανοποίησαν τους αρχικούς στόχους του έργου αλλά και δημιούργησαν τις βάσεις για μελλοντικές βελτιώσεις και επεκτάσεις της εφαρμογής.

Κεφάλαιο 3ο: Αποτελέσματα και Αξιολόγηση

3.1 Εισαγωγή

Το παρόν κεφάλαιο εστιάζει στην παρουσίαση των αποτελεσμάτων της ανάπτυξης και υλοποίησης του SnowHub API, καθώς και στην αξιολόγηση της αποτελεσματικότητας και της ποιότητάς του. Η αξιολόγηση περιλαμβάνει τόσο ποσοτικά όσο και ποιοτικά κριτήρια, αναλύοντας τη λειτουργικότητα του API, τη χρήση του από τους τελικούς χρήστες, καθώς και τις επιδόσεις του συστήματος.

3.2 Endpoints

Η παρακάτω ενότητα περιγράφει τα endpoints που παρέχει το SnowHub API, το οποίο διαχειρίζεται πληροφορίες σχετικά με τα χιονοδρομικά κέντρα της Ελλάδας. Κάθε endpoint επιτρέπει την πρόσβαση σε διαφορετικές πτυχές των δεδομένων που σχετίζονται με τα χιονοδρομικά κέντρα, προσφέροντας έτσι πλήρη και ευέλικτη διαχείριση πληροφοριών.

ENDPOINTS	DESCRIPTION	Headers
GET /SnowResorts	Ανακτά τα στοιχεία όλων των χιονοδρομικών κέντρων.	None required
GET /lifts/{snowResortId}	Ανακτά τη λίστα των αναβατήρων για ένα συγκεκριμένο χιονοδρομικό κέντρο.	None required

POST /register	Δημιουργία νέου λογαριασμού χρήστη.	• Content-Type: application/json • Authorization: Bearer your_generated_token
POST /login	Αυθεντικοποίηση χρήστη και απόκτηση Bearer token.	• Content-Type: application/json
POST /logout	Αποσύνδεση του αυθεντικοποιημένου χρήστη και ακύρωση του Bearer token.	• Authorization: Bearer your_generated_token
POST /booking	Δημιουργία κράτησης από τον χρήστη.	• Content-Type: application/json • Authorization: Bearer your_generated_token
GET /mybooking	Ανακτά τη λίστα των κρατήσεων που έχει κάνει ο αυθεντικοποιημένος χρήστης.	• Authorization: Bearer your_generated_token
GET /user	Ανακτά τα στοιχεία του αυθεντικοποιημένου χρήστη.	• Authorization: Bearer your_generated_token
GET /slopes	Ανακτά τη λίστα όλων των πίστων που είναι διαθέσιμες στα χιονοδρομικά κέντρα.	None required
GET /images	Ανακτά τη λίστα όλων των εικόνων που είναι διαθέσιμες στα χιονοδρομικά κέντρα.	None required
GET /SnowResort/{snowResortId}	Ανακτά πληροφορίες σχετικά με ένα χιονοδρομικό.	None required

Πίνακας 3.1: endpoints

1. **Get Snow Resorts :** GET /SnowResorts

   Το endpoint GET /SnowResorts χρησιμοποιείται για την ανάκτηση των λεπτομερειών όλων των χιονοδρομικών κέντρων. Το συγκεκριμένο endpoint επιστρέφει μια λίστα με πληροφορίες που αφορούν τα χιονοδρομικά κέντρα, όπως το όνομα, την

περιγραφή, την τοποθεσία, τη διαθεσιμότητα των δραστηριοτήτων και των πιστών, καθώς και άλλα σχετικά δεδομένα.Η απάντηση περιλαμβάνει μια λίστα από αντικείμενα, με κάθε αντικείμενο να αντιπροσωπεύει ένα χιονοδρομικό κέντρο. Για κάθε χιονοδρομικό κέντρο περιλαμβάνονται οι εξής πληροφορίες:

Response:

- **id**: Ο μοναδικός αναγνωριστικός αριθμός του χιονοδρομικού κέντρου.
- **name**: Το όνομα του χιονοδρομικού σε διάφορες γλώσσες, όπως ελληνικά (el) και αγγλικά (en).
- **description**: Μια σύντομη περιγραφή του χιονοδρομικού κέντρου.
- **location**: Η γεωγραφική τοποθεσία του κέντρου.
- **slopes_map**: Συντεταγμένες χάρτη που δείχνουν τη θέση των πιστών του χιονοδρομικού κέντρου.
- **status**: Η κατάσταση λειτουργίας του κέντρου (π.χ., "Closed").
- **activities**: Λίστα με τις διαθέσιμες δραστηριότητες (π.χ., σκι) σε διάφορες γλώσσες.
- **slopes**: Πληροφορίες για τις πίστες, όπως το όνομα, το επίπεδο δυσκολίας, το μήκος, και το υψόμετρο.
- **images**: Εικόνες που σχετίζονται με το χιονοδρομικό κέντρο.
- **elevation**: Πληροφορίες για το υψόμετρο, με τη βάση και την κορυφή του χιονοδρομικού.


```
[
    {
        "id": 2,
        "name": {
            "el": "Χ.Κ. Πηλίου",
            "en": "pilio"
        },
        "description": "Το Χιονοδρομικό Κέντρο Πηλίου Αγριόλευκες...",
        "created_at": null,
        "updated_at": null,
        "location": "Πήλιο",
        "slopes_map": "39.38661281127292, 23.083438074693607",
        "site": "-",
        "status": "Closed",
        "activities": {
            "en": [
                {
                    "id": 12,
                    "created_at": null,
                    "updated_at": null,
                    "snow_resort_id": 2,
                    "activity": "ski",
                    "language": "en"
                },
                ...
            ],
```

```
            "el": [
                {
                    "id": 11,
                    "created_at": null,
                    "updated_at": null,
                    "snow_resort_id": 2,
                    "activity": "σκι",
                    "language": "el"
                },
                ...
            ]
        },
        "slopes": [
            {
                "id": 22,
                "created_at": null,
                "updated_at": null,
                "snow_resort_id": 2,
                "name": "Πανόραμα1 Πίστα",
                "difficulty": "red",
                "length_m": 1050,
                "altitude_m": 0,
                "average_slope_percent": "0.00",
                "details": "-"
            },
            ...
        ],
        "images": [
            {
                "id": 2,
                "created_at": null,
                "updated_at": null,
                "snow_resort_id": 2,
                "caption": "view",
                "image_url":
"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT5FN6VMD98vZ3p7okawayeRigOs4l_NXJ79w&s"
            }
        ],
        "elevation": {
            "base": 1300,
            "peak": 1471
        }
    },
    ...
]
```

2. **Get Liftss :** GET /lifts/{snowResortId}

To endpoint GET /lifts/{snowResortId} χρησιμοποιείται για την ανάκτηση της λίστας των ανελκυστήρων (lifts) ενός συγκεκριμένου χιονοδρομικού κέντρου. Οι πληροφορίες που παρέχονται περιλαμβάνουν το όνομα κάθε ανελκυστήρα, την κατάσταση λειτουργίας του και την ημερομηνία καταγραφής των δεδομένων.Το snowResortId (integer) ειναι ο μοναδικός αναγνωριστικός αριθμός του χιονοδρομικού κέντρου για το οποίο ζητούνται τα δεδομένα των αναβατήρων.Το response περιέχει μια λίστα από αντικείμενα, το καθένα από τα οποία αναπαριστά έναν αναβατήρα του χιονοδρομικού κέντρου. Για κάθε αναβατήρα περιλαμβάνονται οι παρακάτω πληροφορίες:

Response:

- **id**: Ο μοναδικός αναγνωριστικός αριθμός του ανελκυστήρα.
- **snow_resort_id**: Ο αναγνωριστικός αριθμός του χιονοδρομικού κέντρου στο οποίο ανήκει ο ανελκυστήρας.
- **name**: Το όνομα του ανελκυστήρα.
- **is_open**: Κατάσταση λειτουργίας του ανελκυστήρα, με τιμές 0 για κλειστό και 1 για ανοιχτό.
- **date**: Η ημερομηνία κατά την οποία καταγράφηκαν τα δεδομένα για τον ανελκυστήρα.
- **created_at**: Η χρονική σήμανση δημιουργίας της εγγραφής του ανελκυστήρα στη βάση δεδομένων.
- **updated_at**: Η χρονική σήμανση της τελευταίας ενημέρωσης της εγγραφής του ανελκυστήρα.:


```
    [
    {
        "id": 1990,
        "snow_resort_id": 2,
        "name": "Πήλιο 1(1-θέσιος Εναέριος)",
        "is_open": 0,
        "date": "2024-09-02",
        "created_at": "2024-09-02T20:21:00.000000Z",
        "updated_at": "2024-09-02T20:21:00.000000Z"
    },
    {
        "id": 1991,
        "snow_resort_id": 2,
        "name": "Πήλιο 2(1-θέσιος Εναέριος)",
        "is_open": 0,
        "date": "2024-09-02",
        "created_at": "2024-09-02T20:21:00.000000Z",
        "updated_at": "2024-09-02T20:21:00.000000Z"
    },
    ...
]
```

3. **POST Register :**POST /register

   Το endpoint POST /register χρησιμοποιείται για τη δημιουργία ενός νέου λογαριασμού χρήστη στο σύστημα. Οι πληροφορίες του χρήστη, όπως το όνομα, το email και ο κωδικός πρόσβασης, αποστέλλονται στο σώμα της αίτησης, και εφόσον η διαδικασία ολοκληρωθεί με επιτυχία, ο χρήστης καταχωρείται στο σύστημα.

   Headers:

   - Content-Type: application/json

Request Body:

```
      {
    "name":  "john",
    "email":  "john12@gmail.com",
    "password":  "password"
}
```

- **name:** Το όνομα του νέου χρήστη.
- **email:** Η ηλεκτρονική διεύθυνση του χρήστη, η οποία θα χρησιμοποιηθεί ως μοναδικό στοιχείο ταυτοποίησης**.**
- **password:** Ο κωδικός πρόσβασης του χρήστη.

Response:

```
{ "message": "User registered successfully" }
```

4. **POST Login :**POST /login

   Το endpoint POST /login χρησιμοποιείται για την αυθεντικοποίηση ενός χρήστη και την απόκτηση ενός Bearer token, το οποίο είναι απαραίτητο για την πρόσβαση σε εξουσιοδοτημένες λειτουργίες της εφαρμογής.To Bearer token που παράγεται μπορεί να χρησιμοποιηθεί σε εξουσιοδοτημένα endpoints, τοποθετώντας το στην κεφαλίδα Authorization: Bearer your_generated_token των μελλοντικών αιτημάτων.

   Headers:

   - Content-Type: application/json

Request Body:

```
{ "email": "user@example.com", "password": "yourpassword" }
```

- **email**: Η ηλεκτρονική διεύθυνση που χρησιμοποιήθηκε κατά την εγγραφή του χρήστη.
- **password**: Ο κωδικός πρόσβασης του χρήστη.

Response:

```
{ "token": "your_generated_token" }
```

5. **POST Logout :**POST /logout

   Το endpoint POST /logout χρησιμοποιείται για την αποσύνδεση του αυθεντικοποιημένου χρήστη και την ακύρωση του Bearer token που έχει εκδοθεί. Με την αποσύνδεση, ο χρήστης χάνει την εξουσιοδότηση για να πραγματοποιήσει μελλοντικά αιτήματα που απαιτούν έγκυρο token.

   Headers:

   - Content-Type: Authorization: your_token: Το Bearer token του χρήστη που πρόκειται να αποσυνδεθεί.

Response:

```
{ "message": "User logged out successfully." }
```

6. **POST Booking:**POST /booking

   Το endpoint POST /booking χρησιμοποιείται για να δημιουργήσει μία ή περισσότερες κρατήσεις εισιτηρίων για χιονοδρομικό κέντρο από έναν αυθεντικοποιημένο χρήστη.Το endpoint αυτό επιτρέπει στον χρήστη να αγοράσει πάσο για ένα συγκεκριμένο χιονοδρομικό κέντρο, καταχωρώντας τα στοιχεία της παραγγελίας και αποθηκεύοντας τις αντίστοιχες κρατήσεις.

   Headers:

   - Content-Type: application/json
   - Authorization: Bearer your_generated_token: Το Bearer token που χρησιμοποιείται για την αυθεντικοποίηση του χρήστη.

   Request:

   - **snow_resort_id:**Το ID του χιονοδρομικού κέντρου στο οποίο θα γίνει η κράτηση.
   - **Cost**: Το κώστος του χιονοδρομικού το οποίο θα γίνει η κράτηση.
   - **number_pass**: Ο αριθμός των εισιτηρίων των οποίων θα γίνει η κράτηση.

Request Body:

```
{
    "snow_resort_id": 1,
    "cost": 13,
    "number_pass": 2
}
```

Response:

- **message**: Επιστρέφει ένα μήνυμα επιβεβαίωσης ότι οι κρατήσεις δημιουργήθηκαν με επιτυχία.
- **bookings**: Μία λίστα με τα στοιχεία των κρατήσεων που δημιουργήθηκαν. Κάθε κράτηση περιλαμβάνει τα παρακάτω πεδία:
- **snow_resort_id**: Το ID του χιονοδρομικού κέντρου για το οποίο έγινε η κράτηση.
- **user_id:** Το ID του χρήστη που έκανε την κράτηση.
- **order_time**: Η χρονική στιγμή κατά την οποία έγινε η κράτηση.
- **created_at**: Η ημερομηνία και ώρα δημιουργίας της κράτησης.
- **updated_at**: Η ημερομηνία και ώρα τελευταίας ενημέρωσης της κράτησης.
- **id**: Το μοναδικό ID της κράτησης.


```
{
    "message": "Bookings created successfully",
"bookings": [
    {
        "snow_resort_id": 1,
        "user_id": 4,
        "order_time": "2024-09-03T09:19:23.218906Z",
        "updated_at": "2024-09-03T09:19:23.000000Z",
        "created_at": "2024-09-03T09:19:23.000000Z",
        "id": 28
    },
    {
        "snow_resort_id": 1,
        "user_id": 4,
        "order_time": "2024-09-03T09:19:23.998001Z",
        "updated_at": "2024-09-03T09:19:23.000000Z",
        "created_at": "2024-09-03T09:19:23.000000Z",
        "id": 29
    }
]
}
```


7. **Get MyBookings:**GET /mybooking

   To endpoint Get My Bookings επιτρέπει την ανάκτηση της λίστας των κρατήσεων που έχει κάνει ο αυθεντικοποιημένος χρήστης.Η απάντηση περιέχει μια λίστα από κρατήσεις που έχουν γίνει από τον χρήστη.

Headers:

- Authorization: Bearer your_generated_token: Το Bearer token που χρησιμοποιείται για την αυθεντικοποίηση του χρήστη.

Response:

- **id**: Το μοναδικό αναγνωριστικό της κράτησης
- **created_at**: Η ημερομηνία και ώρα δημιουργίας της κράτησης σε μορφή ISO.
- **updated_at**: Η ημερομηνία και ώρα τελευταίας ενημέρωσης της κράτησης.
- **snow_resort_id**: Το ID του χιονοδρομικού κέντρου στο οποίο έγινε η κράτηση.
- **user_id**: Το ID του χρήστη που έκανε την κράτηση.
- **order_time**: Η ώρα κατά την οποία έγινε η κράτηση.


```
[
{
    "id": 1,
    "created_at": "2024-08-16T13:58:05.000000Z",
    "updated_at": "2024-08-16T13:58:05.000000Z",
    "snow_resort_id": 1,
    "user_id": 4,
    "order_time": "13:58:05"
},
{
    "id": 2,
    "created_at": "2024-08-16T14:30:16.000000Z",
    "updated_at": "2024-08-16T14:30:16.000000Z",
    "snow_resort_id": 1,
    "user_id": 4,
    "order_time": "14:30:16"
},
{
    "id": 20,
    "created_at": "2024-08-30T09:48:24.000000Z",
    "updated_at": "2024-08-30T09:48:24.000000Z",
    "snow_resort_id": 1,
    "user_id": 4,
    "order_time": "09:48:24"
}
]
```


8. **Get Authenticated User:**GET /user

   Το endpoint get user επιτρέπει την ανάκτηση των πληροφοριών του αυθεντικοποιημένου χρήστη.Η απάντηση περιέχει τα στοιχεία του αυθεντικοποιημένου χρήστη, όπως το αναγνωριστικό του (ID), το όνομα, το email και την ημερομηνία δημιουργίας του λογαριασμού.

Headers:

- Authorization: Bearer your_generated_token: Το Bearer token που χρησιμοποιείται για την αυθεντικοποίηση του χρήστη.

Response:

- **id**: Το μοναδικό αναγνωριστικό της κράτησης
- **name**: Το όνομα του χρήστη.
- **email**: Το email του χρήστη.
- **email_verified_at**: Η ημερομηνία επαλήθευσης του email (αν υπάρχει).
- **created_at**: Η ημερομηνία και ώρα δημιουργίας του λογαριασμού.
- **updated_at**: Η ημερομηνία και ώρα τελευταίας ενημέρωσης του λογαριασμού.


```
{

"id": 4,
"name": "john",
"email": "john@gmail.com",
"email_verified_at": null,
"created_at": "2024-08-14T16:19:48.000000Z",
"updated_at": "2024-08-14T16:19:48.000000Z"
}
```


9. **Get all slopes:**GET /slopes

To get all slopes endpoint επιτρέπει την ανάκτηση της λίστας με όλες τις διαθέσιμες πίστες χιονοδρομικών κέντρων. Η απάντηση περιέχει λίστα με τα χαρακτηριστικά κάθε πίστας, συμπεριλαμβανομένων της δυσκολίας, του μήκους και του υψομέτρου της.

Response:

- **id**: Το μοναδικό αναγνωριστικό της πίστας.
- **created_at**: Η ημερομηνία δημιουργίας της εγγραφής.
- **updated_at**: Η ημερομηνία τελευταίας ενημέρωσης της εγγραφής.
- **snow_resort_id**: Το ID του χιονοδρομικού κέντρου στο οποίο ανήκει η πίστα.
- **name**: Το όνομα της πίστας.
- **difficulty**: Το επίπεδο δυσκολίας της πίστας (π.χ. κόκκινη, μπλε).
- **length_m**: Το μήκος της πίστας σε μέτρα.
- **altitude_m**: Το υψόμετρο της πίστας σε μέτρα.
- **average_slope_percent**: Η μέση κλίση της πίστας σε ποσοστό.
- **details**: Επιπλέον λεπτομέρειες για την πίστα (αν υπάρχουν).


```
[
    {
    "id": 1,
    "created_at": null,
    "updated_at": null,
    "snow_resort_id": 1,
    "name": "Αφροδίτη Α (No 1) Πίστα",
    "difficulty": "red",
    "length_m": 800,
    "altitude_m": 1950,
    "average_slope_percent": "25.00",
```

```
    "details": "-"
},
{
    "id": 2,
    "created_at": null,
    "updated_at": null,
    "snow_resort_id": 1,
    "name": "Αίολος (No 4) Πίστα",
    "difficulty": "blue",
    "length_m": 800,
    "altitude_m": 2100,
    "average_slope_percent": "19.00",
    "details": "-"
}
    // Additional slope objects...
]
```


10. **Get all images:**GET /images

    To Get All Images endpoint επιτρέπει την ανάκτηση όλων των εικόνων που σχετίζονται με τα χιονοδρομικά κέντρα. Η απάντηση περιέχει μία λίστα με τα χαρακτηριστικά των εικόνων, συμπεριλαμβανομένων των πληροφοριών για το χιονοδρομικό κέντρο, τη λεζάντα της εικόνας και το URL της.

    Response:

    - **id**: Το μοναδικό αναγνωριστικό της εικόνας.
    - **created_at**: Η ημερομηνία δημιουργίας της εγγραφής.
    - **updated_at**: Η ημερομηνία τελευταίας ενημέρωσης της εγγραφής.
    - **snow_resort_id**: Το ID του χιονοδρομικού κέντρου στο οποίο ανήκει η εικόνα.
    - **caption**: Η λεζάντα της εικόνας που περιγράφει το περιεχόμενο.


```
image_url: Το URL της εικόνας.
[

    "id": 1,
    "created_at": null,
    "updated_at": null,
    "snow_resort_id": 1,
    "caption": "map",
    "image_url":
"https://parnassos-ski.gr/wp-content/uploads/2022/01/Map_Parnassos-2022-scaled.jpg"
},
{
    "id": 2,
    "created_at": null,
```

```
    "updated_at": null,
    "snow_resort_id": 2,
    "caption": "view",
    "image_url":
"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT5FN6VMD98vZ3p7okawayeRigOs4l_NXJ79w&s"
}
    // Additional image objects...
]
```


11. **Get snow Resort:**GET /SnowResort/{snowResortId}

    Το Get Snow Resort by ID endpoint επιτρέπει την ανάκτηση των λεπτομερειών ενός συγκεκριμένου χιονοδρομικού κέντρου με βάση το μοναδικό του αναγνωριστικό.Η απάντηση περιέχει λεπτομερείς πληροφορίες για το χιονοδρομικό κέντρο, όπως το όνομά του, την περιγραφή, την τοποθεσία, το υψόμετρο, και τον χάρτη των πιστών.

    Response:

    - **id**: Το μοναδικό αναγνωριστικό του χιονοδρομικού κέντρου.
    - **name**: Το όνομα του χιονοδρομικού κέντρου στα αγγλικά.
    - **name_el**: Το όνομα του χιονοδρομικού κέντρου στα ελληνικά..
    - **created_at**: Ημερομηνία δημιουργίας της εγγραφής
    - **cupdated_at:** Ημερομηνία τελευταίας ενημέρωσης της εγγραφής.
    - **location**: Η τοποθεσία του χιονοδρομικού κέντρου.
    - **elevation_base**: Το υψόμετρο της βάσης του χιονοδρομικού κέντρου.
    - **elevation_peak**: Το υψόμετρο της κορυφής του χιονοδρομικού κέντρου.
    - **slopes_map**: Συντεταγμένες του χάρτη πιστών.
    - **site**: Ιστοσελίδα του χιονοδρομικού κέντρου.

- **status**: Η κατάσταση λειτουργίας του χιονοδρομικού κέντρου.

Το παραπάνω κεφάλαιο παρέχει μερική για τα διαθέσιμα endpoints του SnowHub API, διευκολύνοντας τους χρήστες και τους προγραμματιστές να ενσωματώσουν ή να αλληλεπιδράσουν με τα δεδομένα του API με τον πιο αποδοτικό τρόπο.Παρατίθεται ένας σύνδεσμος με το πλήρες documantation του api
https://github.com/xenofon23/GreekSnowResorts

3.3 Αξιοπιστία και Ενημερώσεις Δεδομένων

Η αξιοπιστία και η συχνότητα ενημερώσεων των δεδομένων του SnowHub API είναι βασικά στοιχεία για την ομαλή λειτουργία του συστήματος και την ικανοποίηση των χρηστών. Σε αυτό το τμήμα, αναλύονται οι παράγοντες που επηρεάζουν την αξιοπιστία της υπηρεσίας, καθώς και οι διαδικασίες που ακολουθούνται για την ενημέρωση και την ακρίβεια των δεδομένων.

Η αξιοπιστία του συστήματος μετράται με βάση τη διαθεσιμότητα των υπηρεσιών, την ακρίβεια των δεδομένων και την αντιμετώπιση προβλημάτων σε πραγματικό χρόνο. Οι βασικοί δείκτες αξιοπιστίας περιλαμβάνουν:

- **Χρόνος Διαθεσιμότητας (Uptime)**: Το SnowHub API παρουσίασε υψηλό ποσοστό διαθεσιμότητας, φτάνοντας το 100**% uptime** κατά την περίοδο δοκιμών.Δεν παρατηρήθηκαν διακοπές παρά μόνο προγραμματισμένες συντηρήσεις του συστήματος όπως ένα καινούριο deploy.
- **Αντιμετώπιση Σφαλμάτων**: Το σύστημα παρέχει αποτελεσματικούς μηχανισμούς εντοπισμού και αντιμετώπισης σφαλμάτων. Κάθε σφάλμα καταγράφεται αυτόματα στο log file μέσα στο project.
- **Ανθεκτικότητα**: Το SnowHub API έχει σχεδιαστεί με τρόπο που να αντέχει σε φορτία υψηλής επισκεψιμότητα που οφειλεται μέσα από διαδικασίες του framework της laravel, εξασφαλίζοντας σταθερή απόδοση.

Η ενημέρωση των δεδομένων είναι κρίσιμη για την ακρίβεια των πληροφοριών που παρέχονται στους χρήστες, ειδικά σε περιπτώσεις που οι συνθήκες στις χιονοδρομικές εγκαταστάσεις μεταβάλλονται δυναμικά. Ο τρόπος με τον οποίο το SnowHub API διαχειρίζεται τις ενημερώσεις περιλαμβάνει:

- **Αυτόματες Ενημερώσεις**: Τα δεδομένα για τις χιονοδρομικές εγκαταστάσεις, όπως η κατάσταση των αναβατήρων, ενημερώνονται αυτόματα μέσω cronjob κάθε 10 λεπτά που εκτελείται scraping σε αξιόπιστη ιστοσελίδα (snowreport.gr). Οι ενημερώσεις πραγματοποιούνται σε πραγματικό χρόνο, διασφαλίζοντας ότι οι πληροφορίες είναι πάντα επικαιροποιημένες.
- **Πηγή Δεδομένων**: Οι πληροφορίες αντλούνται από αξιόπιστες πηγές, όπως η ιστοσελίδα (snowreport.gr) και τα επίσημες ιστοσελίδες των χιονοδρομικών κέντρων, εξασφαλίζοντας την ακρίβεια και την επικαιρότητα των δεδομένων.

3.4 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν τα αποτελέσματα και η αξιολόγηση της υλοποίησης του SnowHub API. Αναλύθηκαν τα βασικά endpoints του συστήματος, η αξιοπιστία των δεδομένων και η

διαδικασία ενημέρωσής τους. Η δημιουργία ενός αξιόπιστου και λειτουργικού API, ικανού να διαχειρίζεται και να παρέχει ακριβείς πληροφορίες σε πραγματικό χρόνο, όπου αποτέλεσε βασικό στόχο της ανάπτυξης. Παρά τις προκλήσεις, η υλοποίηση πέτυχε τους στόχους της, εξασφαλίζοντας μια σταθερή βάση για μελλοντική εξέλιξη και περαιτέρω βελτιώσεις.

Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Στο παρόν κεφάλαιο παρουσιάζονται τα βασικά συμπεράσματα που προέκυψαν από την ανάπτυξη του SnowHub API, καθώς και προτάσεις για μελλοντικές βελτιώσεις. Το SnowHub API αποτελεί ένα σύγχρονο εργαλείο για την παροχή ενημερώσεων σχετικά με τις συνθήκες στα χιονοδρομικά κέντρα. Τα συμπεράσματα επικεντρώνονται στη λειτουργικότητα του συστήματος, ενώ οι προτάσεις αποσκοπούν στη βελτίωση της απόδοσης, της ασφάλειας και της εμπειρίας των χρηστών

1. **Επιτυχία στη Δημιουργία Ενοποιημένου Συστήματος Δεδομένων**: Το SnowHub API κατάφερε να συγκεντρώσει δεδομένα από διάφορες πηγές και να τα παρουσιάσει με συνεπή τρόπο στους χρήστες.
2. **Βελτίωση της Εμπειρίας Χρηστών**: Το SnowHub API προσέφερε στους χρήστες έγκαιρη και αξιόπιστη πληροφόρηση, επιτρέποντάς τους να σχεδιάσουν τις εξορμήσεις τους με μεγαλύτερη ακρίβεια και ασφάλεια. Επίσης αποτελεί ένα εργαλείο που θα δίνει την δυνατότητα να κατασκευαστούν android , ios και web εφαρμογές μέσω αυτού.

4.1 Επεκτασημότητα

Η επιτυχία του SnowHub API ανοίγει τον δρόμο για μελλοντικές επεκτάσεις που θα ενισχύσουν τόσο τη λειτουργικότητά του όσο και την αλληλεπίδραση με τα χιονοδρομικά κέντρα. Στην παρούσα μορφή, το API συγκεντρώνει δεδομένα από διάφορες πηγές, αλλά με την εξέλιξη της τεχνολογίας και την αυξανόμενη ζήτηση για άμεση και ακριβή πληροφόρηση, υπάρχουν σημαντικές δυνατότητες για περαιτέρω επέκταση.

1. **Δημιουργία Διαχειριστικού Συστήματος για τα Χιονοδρομικά Κέντρα**

Με τη συνεχιζόμενη επιτυχία του API, μια φυσική επέκταση θα μπορούσε να είναι η ανάπτυξη ενός ολοκληρωμένου διαχειριστικού συστήματος, μέσω του οποίου τα ίδια τα χιονοδρομικά κέντρα θα έχουν τη δυνατότητα να ενημερώνουν τις πληροφορίες σε πραγματικό χρόνο. Αυτό το σύστημα θα μπορούσε να λειτουργεί ως μια διαδικτυακή πλατφόρμα ή εφαρμογή, όπου οι διαχειριστές των κέντρων θα εισάγουν τις απαραίτητες πληροφορίες σχετικά με τις καιρικές συνθήκες, την κατάσταση των πιστών και των αναβατήρων, την προσβασιμότητα, καθώς και άλλες σημαντικές πληροφορίες όπως ωράρια λειτουργίας και διαθέσιμες υπηρεσίες.

Η αυτοδιαχείριση των δεδομένων από τα χιονοδρομικά κέντρα θα είχε πολλαπλά οφέλη:

- **Ακρίβεια και συνέπεια**: Οι πληροφορίες θα είναι άμεσα διαθέσιμες στους χρήστες και θα είναι πιο ακριβείς, καθώς θα προέρχονται απευθείας από την πηγή.

- **Ευελιξία**: Τα χιονοδρομικά κέντρα θα μπορούν να προσαρμόζουν άμεσα τα δεδομένα, ανταποκρινόμενα σε αλλαγές των συνθηκών ή σε έκτακτες ανάγκες.

2. **Αυτοματοποίηση μέσω Αισθητήρων**

Μια ακόμα πιο προχωρημένη προσέγγιση στην επεκτασιμότητα του SnowHub API θα μπορούσε να είναι η πλήρης αυτοματοποίηση της συλλογής δεδομένων μέσω της εγκατάστασης αισθητήρων στα χιονοδρομικά κέντρα. Οι αισθητήρες αυτοί θα μπορούσαν να συλλέγουν δεδομένα για:

- **Θερμοκρασία** και άλλες καιρικές συνθήκες.
- **Πάχος και ποιότητα χιονιού**.
- **Ορατότητα** και άλλες κρίσιμες πληροφορίες για την ασφάλεια των σκιέρ.
- **Κατάσταση πιστών**: Πόσες πίστες είναι ανοιχτές, ποιες είναι οι συνθήκες σε αυτές, καθώς και η κατάσταση των λιφτ.

Αυτοί οι αισθητήρες, συνδεδεμένοι σε ένα κεντρικό σύστημα, θα μπορούσαν να παρέχουν ενημερώσεις σε πραγματικό χρόνο, χωρίς την ανάγκη ανθρώπινης παρέμβασης. Αυτό το σύστημα θα μπορούσε να λειτουργεί πλήρως αυτόνομα, μειώνοντας την εξάρτηση από την ανθρώπινη συμμετοχή και αυξάνοντας τη συνέπεια και την αξιοπιστία των δεδομένων.

4.2 Επίλογος

Στο τελευταίο κεφάλαιο της πτυχιακής εργασίας αναλύθηκαν τα συμπεράσματα και οι προτάσεις για τη βελτίωση του SnowHub API. Η επιτυχής υλοποίηση του συστήματος αποδεικνύει την αξία και τη χρησιμότητά του για τη διαχείριση πληροφοριών χιονοδρομικών κέντρων. Με βάση τα ευρήματα, προκύπτουν ευκαιρίες για επεκτασιμότητα, όπως η δημιουργία ενός διαχειριστικού εργαλείου για τα χιονοδρομικά ή η αυτοματοποίηση των δεδομένων με τη χρήση αισθητήρων. Οι δυνατότητες περαιτέρω ανάπτυξης του έργου είναι σημαντικές, με στόχο την παροχή πιο ακριβών και ενημερωμένων πληροφοριών σε πραγματικό χρόνο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] PHP Documentation. (n.d.). [Online]. Available: https://www.php.net/docs.php

[2] laravel Documentation. (n.d.). [Online]. Available: https://laravel.com/docs/11.x

[3] Mysql Documentation. (n.d.). [Online]. Available:https://dev.mysql.com/doc/

[4] postman Documentation. (n.d.). [Online]. Available:https://learning.postman.com/docs

[5] php informations. (n.d.). [Online]. Available:https://en.wikipedia.org/wiki/PHP

[6] GitHub. (n.d.). *GitHub Documentation*. [Online]. Available: https://docs.github.com/en

[7] Debian 12. (n.d.). Debian *Documentation*. [Online]. Available:https://www.debian.org/doc/

[8] PhpStorm. (n.d.). PhpStorm *Documentation*. [Online]. Available:https://www.jetbrains.com/phpstorm/resources/

[9] Nginx. (n.d.). Nginx *Documentation*. [Online]. Available:https://nginx.org/en/docs/