ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Ανάπτυξη Ολοκληρωμένου Συστήματος Εξυπηρέτησης Διατροφολόγου και Πελατών – Διαδικτυακή Πλατφόρμα και Εφαρμογή για Κινητά Development of an Integrated Service System for a Nutritionist and Clients – Web Platform and Mobile Application
Ονοματεπώνυμο Φοιτητή	Δημήτρης Στυλιανού
Πατρώνυμο	Σάββας
Αριθμός Μητρώου	Π/20004
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης Σεπτέμβριος 2024

Copyright ©

Ευχαριστίες

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες στον επιβλέποντα καθηγητή μου, κ. Ευθύμιο Αλέπη, για την πολύτιμη καθοδήγηση και την συνεχή υποστήριξη που μου προσέφερε καθ' όλη τη διάρκεια της εκπόνησης της πτυχιακής μου εργασίας, αλλά και σε όλους τους καθηγητές του Τμήματος Πληροφορικής για τις γνώσεις που μου παρείχαν καθ' όλη τη διάρκεια των προπτυχιακών σπουδών.

Επίσης, νιώθω την ανάγκη να ευχαριστήσω την οικογένειά μου για την αμέριστη ηθική στήριξη και την κατανόηση που μου έδειξαν καθ' όλη τη διάρκεια αυτής της απαιτητικής διαδικασίας. Χωρίς τη βοήθειά τους, η ολοκλήρωση αυτής της εργασίας θα ήταν σαφώς πιο δύσκολη.

Τέλος, θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου στην κοπέλα μου, που ήταν πάντα δίπλα μου σε κάθε βήμα της διαδικασίας, προσφέροντας υποστήριξη και ενθάρρυνση, καθώς και στους φίλους και συμφοιτητές μου, οι οποίοι ήταν πάντα πρόθυμοι να συζητήσουν και να μου προσφέρουν χρήσιμες συμβουλές. Επιπλέον, θα ήθελα να ευχαριστήσω τον φίλο μου Λούκα Τζάκσικ για τη δημιουργία του λογότυπου της κινητής εφαρμογής, το οποίο προσέδωσε μοναδικότητα και χαρακτήρα στο έργο, κάνοντάς το να φαίνεται πιο ολοκληρωμένο και επαγγελματικό.

Περίληψη

Η παρούσα εργασία στοχεύει στην υλοποίηση ενός συστήματος εξυπηρέτησης διατροφολόγου που διευκολύνει την επικοινωνία με τους πελάτες του. Το σύστημα υποστηρίζει τη χρήση διαδικτυακής πλατφόρμας για τον διατροφολόγο και εφαρμογής για κινητά τηλέφωνα για τους πελάτες. Η αρχιτεκτονική περιλαμβάνει έναν διακομιστή υποστήριξης (backend server) με Spring Boot Framework, ο οποίος εξυπηρετεί τόσο την διαδικτυακή πλατφόρμα όσο και την εφαρμογή για κινητά, η οποία αναπτύχθηκε χρησιμοποιώντας το Jetpack Compose toolkit.

Για την επιτυχή υλοποίηση, εφαρμόστηκαν οι βέλτιστες δυνατές πρακτικές και μεθοδολογίες που ισχύουν στη σύγχρονη αγορά εργασίας. Η εργασία ανήκει στον τομέα ανάπτυξης λογισμικού. Μέσα από αυτή τη διαδικασία εκπόνησης της εργασίας, αποκτήθηκε μια πιο σφαιρική εικόνα για το εύρος δυνατοτήτων που υπάρχουν στον σύγχρονο τεχνολογικό κόσμο σχετικά με την ανάπτυξη και υποστήριξη τέτοιων συστημάτων και εφαρμογών.

Λέξεις Κλειδιά: λογισμικό, διαδικτυακή πλατφόρμα, εφαρμογή για κινητά, διακομιστής υποστήριξης, Spring Boot, Jetpack Compose toolkit.

Abstract

This thesis aims to implement a service system for nutritionists that facilitates communication with their clients. The system supports the use of a web platform for the nutritionist and a mobile application for clients. The architecture includes a backend server using Spring Boot Framework, which serves both the web platform, and the mobile application developed with the Jetpack Compose toolkit.

To achieve successful implementation, the best possible practices and methodologies applicable in the modern job market were applied. This work falls within the field of software development. Through the development process of this thesis, a broader insight was gained into the range of possibilities offered by the modern technological world regarding the development and support of such systems and applications.

Keywords: software, web platform, mobile application, backend server, Spring Boot, Jetpack Compose toolkit.

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Εισαγωγή

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη ενός ολοκληρωμένου συστήματος για τη διευκόλυνση της επικοινωνίας και της διαχείρισης των πελατών από διατροφολόγους. Το σύστημα αποτελείται από μια διαδικτυακή πλατφόρμα για τη χρήση από έναν διατροφολόγο (με δυνατότητα επέκτασης για περισσότερους), καθώς και μια εφαρμογή για κινητά προσαρμοσμένη στις ανάγκες των πελατών. Στόχος είναι να επιτευχθεί μια πιο άμεση και αποτελεσματική επικοινωνία μεταξύ των διατροφολόγων και των πελατών τους, καθώς και να διευκολυνθεί η διαχείριση διατροφικών προγραμμάτων και άλλων πληροφοριών σε πραγματικό χρόνο.

Στην εποχή μας, η τεχνολογία παίζει καθοριστικό ρόλο στη βελτίωση των υπηρεσιών υγείας, και οι διατροφολόγοι, ως επαγγελματίες υγείας, επωφελούνται ιδιαίτερα από ψηφιακά εργαλεία που επιτρέπουν την άμεση παρακολούθηση και επικοινωνία με τους πελάτες τους. Τα συστήματα αυτά προσφέρουν σημαντικές ευκολίες στους πελάτες, παρέχοντας τους πρόσβαση σε χρήσιμες πληροφορίες όπως το πρόγραμμα διατροφής τους, τις συναντήσεις τους, αλλά και δεδομένα, σε πραγματικό χρόνο.

Η διεθνής βιβλιογραφία δείχνει αυξανόμενο ενδιαφέρον για την ανάπτυξη ψηφιακών συστημάτων που υποστηρίζουν την αλληλεπίδραση μεταξύ επαγγελματιών υγείας και πελατών, ειδικά μετά την πανδημία του κορονοϊού. Στον τομέα της διατροφής, έχουν αναπτυχθεί πολλές εφαρμογές για τη διαχείριση διατροφικών δεδομένων και την παρακολούθηση της προόδου των πελατών. Ωστόσο, η παρούσα εργασία διαφοροποιείται με τον συνδυασμό διαδικτυακής πλατφόρμας και εφαρμογής για κινητά, προσφέροντας μια ολοκληρωμένη λύση τόσο για τους διατροφολόγους όσο και για τους πελάτες τους.

Οι επιμέρους στόχοι της εργασίας περιλαμβάνουν την ανάπτυξη ενός ασφαλούς και αξιόπιστου διακομιστή υποστήριξης με τη χρήση του Spring Boot, που θα υποστηρίζει τόσο την πλατφόρμα όσο και την εφαρμογή. Παράλληλα, επιδιώκεται η δημιουργία μιας φιλικής προς τον χρήστη εφαρμογής για κινητά με το Jetpack Compose. Η εργασία συνεισφέρει στο επιστημονικό πεδίο της ανάπτυξης λογισμικού, επιδεικνύοντας τη χρήση σύγχρονων τεχνολογιών και πρακτικών για την ανάπτυξη ολοκληρωμένων συστημάτων εξυπηρέτησης.

Η παρούσα πτυχιακή εργασία διαρθρώνεται σε οκτώ βασικά κεφάλαια:

1. **Αναλυτική Περιγραφή Αντικειμένου**: Αναλύει τις λειτουργίες και τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος, με σκοπό την κατανόηση των βασικών χαρακτηριστικών και δυνατοτήτων του.
2. **Μεθοδολογική Προσέγγιση**: Αναλύει την μεθοδολογική προσέγγιση που υιοθετήθηκε για την υλοποίηση του συστήματος, πως έγινε διαχείριση του έργου και ποιες αρχιτεκτονικές σχεδίασης λογισμικού χρησιμοποιήθηκαν.
3. **Αποτελέσματα**: Γίνεται αναφορά στα αποτελέσματα και στις τελικές λειτουργίες που υλοποιήθηκαν.
4. **Συμπεράσματα**: Γίνεται αναφορά στους στόχους, τυχόν προβλήματα που αντιμετωπίστηκαν και ανάδειξη κατευθύνσεων για μελλοντική έρευνα.
5. **Πίνακας Ορολογίας**: Περιέχει όρους που χρησιμοποιήθηκαν στα ελληνικά και στα αγγλικά.
6. **Πίνακας Συντμήσεων-Αρκτικόλεξων-Ακρωνύμιων**: Περιέχει όρους που αναγράφονται συντομογραφικά.
7. **Βιβλιογραφία**: Περιλαμβάνει το σύνολο των πηγών στις οποίες υπάρχει παραπομπή στο κείμενο (με αλφαβητική σειρά).
8. **Παραρτήματα**: Περιλαμβάνει αναφορές σε κώδικα της εργασίας που δεν εισάγεται στο κυρίως κείμενο.

1. Αναλυτική Περιγραφή Αντικειμένου

Η παρούσα εργασία αναπτύσσει ένα ολοκληρωμένο σύστημα εξυπηρέτησης διατροφολόγων και πελατών, το οποίο περιλαμβάνει διαδικτυακή πλατφόρμα και εφαρμογή για κινητά. Στα υποκεφάλαια που ακολουθούν, θα παρουσιαστεί εκτενώς η περιγραφή των λειτουργιών και των εργαλείων που χρησιμοποιούνται για την υλοποίηση του συστήματος.

1.1 Διακομιστής υποστήριξης

Για την υλοποίηση του διακομιστή υποστήριξης επιλέχθηκε το Spring Boot Framework (VMware Tanzu, n.d.-c), το οποίο παρέχει μια γρήγορη, ευέλικτη και ασφαλή λύση για την ανάπτυξη διαδικτυακών εφαρμογών και συστημάτων υποστήριξης. Με τις “έτοιμες προς χρήση” λειτουργίες του, μειώνει την ανάγκη για πολύπλοκες ρυθμίσεις και επιταχύνει τον κύκλο ανάπτυξης των εφαρμογών.

Το Spring Boot βασίζεται στο ευρέως διαδεδομένο Spring Framework (VMware Tanzu, n.d.-f), το οποίο προσφέρει έναν ευέλικτο και σύγχρονο τρόπο ανάπτυξης εφαρμογών για επιχειρήσεις (enterprise applications) βασισμένες στην γλώσσα προγραμματισμού Java. Το Spring Framework εστιάζει κυρίως στην επιχειρηματική λογική σε επίπεδο εφαρμογής (application-level business logic), ενώ παράλληλα απλοποιεί την διαχείριση των περιβαλλόντων εγκατάστασης και εκτέλεσης (deployment environments).

Η επιλογή του Spring Boot έγινε κυρίως λόγω της υποστήριξής του για το μοντέλο αρχιτεκτονικής λογισμικού MVC, το οποίο χρησιμοποιείται στη διαδικτυακή πλατφόρμα, καθώς και της εξαιρετικής του συμβατότητας με RESTful APIs, τα οποία είναι απαραίτητα για την επικοινωνία με την εφαρμογή για κινητά.

Ακολουθεί μια αναλυτική περιγραφή της δομής του διακομιστή, η οποία περιλαμβάνει τα παρακάτω επιμέρους συστατικά:

- **Σύστημα Διαχείρισης Βάσης Δεδομένων (Database Management System)**: Περιλαμβάνει την περιγραφή της βάσης δεδομένων, των πινάκων και των σχέσεων μεταξύ τους. Η βάση δεδομένων φιλοξενείται σε περιβάλλον Docker, το οποίο παρέχει απομόνωση και ευκολία στη διαχείριση.
- **Hibernate ORM και Spring Data JPA**: Χρησιμοποιούνται για τη διαχείριση της βάσης δεδομένων και την υποστήριξη των λειτουργιών CRUD (Create, Read, Update, Delete). Το Hibernate ORM λειτουργεί ως ένα εργαλείο αντικειμενοστραφούς/σχεσιακής αντιστοίχισης (Object/Relational Mapping – ORM), επιτρέποντας την αντιστοίχισης αντικειμένων Java σε πίνακες της βάσης δεδομένων.
- **Hibernate Validator**: Εξασφαλίζει την εγκυρότητα των δεδομένων που διαχειρίζεται το σύστημα, πριν αυτά αποθηκευτούν στη βάση δεδομένων.
- **Spring AOP**: Εφαρμόζεται για τη διαχείριση cross-cutting concerns της επιχειρηματικής λογικής, όπως καταγραφή συμβάντων (logging), μέσω του θεματοστρεφή προγραμματισμού (Aspect-Oriented Programming – AOP).
- **Spring Security**: Εστιάζει στην ασφάλεια του συστήματος και στους μηχανισμούς αυθεντικοποίησης και εξουσιοδότησης (authentication and authorization) των χρηστών.
- **Spring Email**: Χρησιμοποιείται για την αποστολή email μέσω του συστήματος.
- **Project Lombok**: Απλοποιεί τον κώδικα μέσω σχολιασμών (annotations), μειώνοντας τον επαναλαμβανόμενο κώδικα (boilerplate code) και καθιστώντας τον πιο ευανάγνωστο.
- **Thymeleaf**: Χρησιμοποιείται για την ανάπτυξη δυναμικών διαδικτυακών σελίδων στη πλευρά του διακομιστή (server-side), επιτρέποντας την εύκολη ενσωμάτωση δεδομένων και λογικής μέσα σε HTML αρχεία.

- **Java JWT library**: Εφαρμόζεται για την δημιουργία JSON Web Tokens (JWT), διασφαλίζοντας την ασφαλή επικοινωνία μεταξύ πελάτη και διακομιστή (client and server).

1.1.1 Σύστημα Διαχείρισης Βάσης Δεδομένων

Για την αποθήκευση και διαχείριση των δεδομένων του συστήματος, επιλέχθηκε το σύστημα διαχείρισης βάσης δεδομένων PostgreSQL (PostgreSQL Global Development Group, n.d.), το οποίο αποτελεί ένα από τα πιο διαδεδομένα σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων (Relational Database Management Systems – RDBMS). Το PostgreSQL παρέχει υψηλή αξιοπιστία (reliability), δυνατότητες επεκτασιμότητας (scalability) και συμμόρφωσης με το πρότυπο SQL (SQL standard).

Το σύστημα αυτό φιλοξενείται σε περιβάλλον Docker (Docker Inc, n.d.), το οποίο επιτρέπει τη δημιουργία ενός απομονωμένου και εύκολα διαχειρίσιμου περιβάλλοντος. Η χρήση του Docker διευκολύνει την ανάπτυξη και δοκιμή του συστήματος διαχείρισης βάσης δεδομένων, χωρίς την ανάγκη τοπικής εγκατάστασης. Αυτό καθιστά τη διαδικασία ανάπτυξης πιο αποδοτική και αποτρέπει τα προβλήματα συμβατότητας μεταξύ διαφορετικών μηχανημάτων ή λειτουργικών συστημάτων.

Η βάση δεδομένων έχει σχεδιαστεί σύμφωνα με τις ανάγκες του συστήματος και περιλαμβάνει πίνακες που αντιπροσωπεύουν τις διάφορες οντότητες (entities) του. Οι πίνακες αυτοί περιλαμβάνουν περιορισμούς ακεραιότητας (integrity constraints), όπως πρωτεύοντα κλειδιά (primary keys), ξένα κλειδιά (foreign keys) και περιορισμούς μοναδικότητας (unique constraints) και μη κενών τιμών (not null constraints). Έτσι, διασφαλίζεται η συνέπεια και η ακεραιότητα των δεδομένων (data consistency and integrity).

Ακολουθεί λεπτομερής ανάλυση των πινάκων της βάσης δεδομένων, με περιγραφή των πεδίων τους και των περιορισμών που επιβάλλονται σε αυτά, εξασφαλίζοντας την ορθή αποθήκευση και ανάκτηση των δεδομένων.

Πίνακας *user*

Ο πίνακας *user* περιέχει τα πεδία που είναι απαραίτητα για την αυθεντικοποίηση των χρηστών του συστήματος. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό για κάθε χρήστη. Είναι τύπου *serial*, το οποίο σημαίνει ότι η τιμή αυξάνεται αυτόματα με κάθε νέα καταχώρηση. Το πεδίο αυτό ορίζεται ως το πρωτεύον κλειδί του πίνακα, το οποίο συνοδεύεται από περιορισμούς μοναδικότητας και μη κενής τιμής. Η προσθήκη των περιορισμών αυτών εξασφαλίζει τη μοναδικότητα κάθε εγγραφής στον πίνακα.
- ***username***: Το όνομα χρήστη που χρησιμοποιείται για την είσοδο στο σύστημα. Είναι τύπου *varchar(102)*, με μέγιστο μήκος 102 χαρακτήρες. Η συγκεκριμένη επιλογή έγινε βάσει του αλγορίθμου δημιουργίας προσωρινού ονόματος χρήστη κατά την εγγραφή στο σύστημα (βλ. Παράρτημα Α). Το πεδίο έχει περιορισμούς μοναδικότητας και μη κενής τιμής, για να εξασφαλιστεί ότι κάθε όνομα χρήστη είναι μοναδικό και υποχρεωτικό.
- ***password***: Το κρυπτογραφημένο συνθηματικό του χρήστη. Είναι τύπου *varchar(60)* και υποχρεωτικό. Το μέγιστο μήκος των 60 χαρακτήρων προκύπτει από τη χρήση του αλγορίθμου κρυπτογράφησης BCrypt, ο οποίος παράγει κωδικούς συγκεκριμένου μήκους.
- ***enabled***: Δείκτης τύπου *boolean* που υποδεικνύει αν ο λογαριασμός είναι ενεργός ή όχι. Η προεπιλεγμένη τιμή είναι *false* για νέους χρήστες, κάτι που σημαίνει ότι ο λογαριασμός πρέπει να ενεργοποιηθεί πριν χρησιμοποιηθεί. Το πεδίο είναι υποχρεωτικό.
- ***role***: Ο ρόλος του χρήστη στο σύστημα, ο οποίος καθορίζεται από τον τύπο *role_enum*. Το πεδίο αυτό είναι υποχρεωτικό και εξασφαλίζει ότι κάθε χρήστης έχει συγκεκριμένα δικαιώματα και δυνατότητες ανάλογα με τον ρόλο του. Οι αποδεκτές τιμές είναι “CLIENT” και “DIETITIAN”.

Ο συγκεκριμένος τύπος περιορίζει τις τιμές που μπορεί να πάρει το πεδίο, παρέχοντας ακρίβεια και ασφάλεια στη διαχείριση των ρόλων.

```
create type role_enum as enum ('CLIENT', 'DIETITIAN');

create table "user"
(
    id       serial       not null unique primary key,
    username varchar(102) not null unique,
    password varchar(60)  not null,
    enabled  boolean      not null default false,
    role     role_enum    not null
);
```

Εικόνα 1. Πίνακας user

Πίνακας *user-info*

Ο πίνακας *user_info* περιέχει πεδία που διατηρούν επιπλέον χρήσιμες πληροφορίες για τους χρήστες, όπως τα προσωπικά τους στοιχεία και δεδομένα επικοινωνίας. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό για κάθε εγγραφή στον πίνακα. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***user_id***: Το μοναδικό αναγνωριστικό του χρήστη που σχετίζεται με την εγγραφή, και αντιστοιχεί στο πεδίο *id* του πίνακα *user*. Ορίζεται ως ξένο κλειδί, εξασφαλίζοντας τη συσχέτιση μεταξύ των δύο πινάκων. Το πεδίο μοναδικό και υποχρεωτικό.
- **first_name**: Το μικρό όνομα του χρήστη. Είναι τύπου *varchar(50)* με μέγιστο μήκος 50 χαρακτήρες και είναι υποχρεωτικό.
- **last_name**: Το επώνυμο του χρήστη. Είναι τύπου *varchar(50)* με μέγιστο μήκος 50 χαρακτήρες και είναι υποχρεωτικό.
- ***gender***: Το φύλο του χρήστη. Είναι τύπου *gender_enum* με τις αποδεκτές τιμές “MALE” και “FEMALE”. Το πεδίο είναι υποχρεωτικό.
- ***date_of_birth***: Η ημερομηνία γέννησης του χρήστη. Είναι τύπου *date* και είναι υποχρεωτικό.
- ***created_at***: Η ημερομηνία και ώρα κατά την οποία δημιουργήθηκε η εγγραφή. Είναι τύπου *timestamp* και έχει προκαθορισμένη τιμή την τρέχουσα ημερομηνία και ώρα κατά τη στιγμή της καταχώρησης (*now()*). Το πεδίο είναι υποχρεωτικό.
- ***email***: Η διεύθυνση ηλεκτρονικού ταχυδρομείου του χρήστη. Είναι τύπου *varchar(50)* με μέγιστο μήκος 50 χαρακτήρες και είναι υποχρεωτικό.
- ***phone***: Ο αριθμός τηλεφώνου του χρήστη. Είναι τύπου *varchar(20)* με μέγιστο μήκος 20 χαρακτήρες και είναι υποχρεωτικό.

```
create type gender_enum as enum ('MALE', 'FEMALE');

create table user_info
(
    id            serial      not null unique primary key,
    user_id       int         not null unique references "user" (id),
    first_name    varchar(50) not null,
    last_name     varchar(50) not null,
    gender        gender_enum not null,
    date_of_birth date        not null,
    created_at    timestamp   not null default now(),
    email         varchar(50) not null,
    phone         varchar(20) not null
);
```

Εικόνα 2. Πίνακας user_info

Πίνακας *tag_category*

Ο πίνακας *tag_category* διατηρεί τις κατηγορίες στις οποίες ανήκουν οι ετικέτες που χρησιμοποιούνται στο σύστημα. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε κατηγορίας. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***name***: Το όνομα της κατηγορίας. Είναι τύπου *varchar(100)* με μέγιστο μήκος 100 χαρακτήρες και είναι μοναδικό και υποχρεωτικό, εξασφαλίζοντας ότι δεν υπάρχουν διπλές εγγραφές με το ίδιο όνομα κατηγορίας.

```
create table tag_category
(
    id   serial       not null unique primary key,
    name varchar(100) not null unique
);
```

Εικόνα 3. Πίνακας tag_category

Πίνακας *tag*

Ο πίνακας *tag* περιέχει τις ετικέτες που χρησιμοποιούνται από τον διατροφολόγο για την εξατομικευμένη ανάθεση ετικετών σε πελάτες και άρθρα. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε ετικέτας. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***tag_category_id***: Το μοναδικό αναγνωριστικό της κατηγορίας στην οποία ανήκει η ετικέτα. Αντιστοιχεί στο πεδίο *id* του πίνακα *tag_category* και ορίζεται ως ξένο κλειδί, εξασφαλίζοντας τη συσχέτιση μεταξύ των δύο πινάκων. Το πεδίο είναι υποχρεωτικό.
- ***name***: Το όνομα της ετικέτας. Είναι τύπου *varchar(100)* με μέγιστο μήκος 100 χαρακτήρες. Το πεδίο είναι μοναδικό και υποχρεωτικό, αποτρέποντας την καταχώρηση ετικετών με το ίδιο όνομα.

```
create table tag
(
    id              serial       not null unique primary key,
    tag_category_id int          not null references tag_category (id),
    name            varchar(100) not null unique
);
```

Εικόνα 4. Πίνακας tag

Πίνακας *client_tag_association*

Ο πίνακας *client_tag_association* καταγράφει τις ετικέτες που αντιστοιχούν σε κάθε πελάτη, δημιουργώντας μια συσχέτιση μεταξύ των χρηστών και των ετικετών. Αναλυτικά:

- ***user_info_id***: Το μοναδικό αναγνωριστικό του πελάτη. Ορίζεται ως ξένο κλειδί που συσχετίζεται με το πεδίο *id* του πίνακα *user_info*, εξασφαλίζοντας ότι κάθε ετικέτα αντιστοιχεί σε έναν συγκεκριμένο πελάτη. Το πεδίο είναι υποχρεωτικό.
- ***tag_id***: Το μοναδικό αναγνωριστικό της ετικέτας. Ορίζεται ως ξένο κλειδί που συνδέεται με το πεδίο *id* του πίνακα *tag*, εξασφαλίζοντας ότι κάθε ετικέτα που αποδίδεται σε έναν πελάτη είναι έγκυρη και υπάρχει στον πίνακα ετικετών. Το πεδίο είναι υποχρεωτικό.

  Το πρωτεύον κλειδί του πίνακα αποτελείται από το συνδυασμό των πεδίων *user_info_id* και *tag_id*, εξασφαλίζοντας ότι η κάθε συσχέτιση μεταξύ ενός πελάτη και μιας ετικέτας είναι μοναδική.

```
create table client_tag_association
(
    user_info_id int not null references user_info (id),
    tag_id       int not null references tag (id),
    primary key (user_info_id, tag_id)
);
```

Εικόνα 5. Πίνακας client_tag_association

Πίνακας *diet_plan*

Ο πίνακας *diet_plan* περιέχει τα απαραίτητα στοιχεία για τη διαχείριση των προγραμμάτων διατροφής που δημιουργούνται για τους πελάτες. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε προγράμματος διατροφής. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***user_info_id***: Το μοναδικό αναγνωριστικό του χρήστη για τον οποίο δημιουργείται το πρόγραμμα διατροφής. Ορίζεται ως ξένο κλειδί που αναφέρεται στο πεδίο *id* του πίνακα *user_info*, εξασφαλίζοντας τη σύνδεση του προγράμματος διατροφής με έναν συγκεκριμένο χρήστη. Το πεδίο αυτό είναι υποχρεωτικό.
- ***name***: Το όνομα που δίνεται στο πρόγραμμα διατροφής. Είναι τύπου *varchar(50)*, με μέγιστο όριο 50 χαρακτήρες. Κάθε όνομα πρέπει να είναι μοναδικό, καθώς αυτό χρησιμοποιείται για την ανάκτηση και τη διαχείριση του στο χώρο αποθήκευσης του διακομιστή.
- ***created_on***: Η ημερομηνία δημιουργίας του προγράμματος διατροφής. Είναι τύπου *date* και έχει προκαθορισμένη τιμή την τρέχουσα ημερομηνία (*current_date*), πράγμα που σημαίνει ότι με κάθε νέα καταχώρηση αποθηκεύεται αυτόματα η ημερομηνία κατά την οποία δημιουργήθηκε το πρόγραμμα.
- ***unique (user_info_id, created_on)***: Αυτός ο περιορισμός συνδυάζει τα πεδία *user_info_id* και *created_on*, εξασφαλίζοντας τη μοναδικότητα του συνδυασμού τους, δηλαδή ότι δεν

μπορεί να καταχωρηθεί πάνω από ένα πρόγραμμα διατροφής για έναν χρήστη την ίδια ημέρα.

```
create table diet_plan
(
    id           serial      not null unique primary key,
    user_info_id int         not null references user_info (id),
    name         varchar(50) not null unique,
    created_on   date        not null default current_date,
    unique (user_info_id, created_on)
);
```

Εικόνα 6. Πίνακας diet_plan

Πίνακας *article*

Ο πίνακας *article* περιέχει τα στοιχεία που αφορούν τα άρθρα που δημοσιεύονται στο σύστημα. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε άρθρου. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***title***: Ο τίτλος του άρθρου. Είναι τύπου *varchar(100)*, επιτρέποντας μέγιστο μήκος 100 χαρακτήρες. Το πεδίο είναι μοναδικό και υποχρεωτικό, εξασφαλίζοντας ότι δεν θα υπάρχουν δύο άρθρα με τον ίδιο τίτλο.
- ***content***: Το περιεχόμενο του άρθρου. Είναι τύπου *text* και είναι υποχρεωτικό. Σε αυτό το πεδίο αποθηκεύεται το κυρίως κείμενο του άρθρου.
- ***created_at***: Η χρονική σήμανση που υποδεικνύει πότε δημιουργήθηκε το άρθρο. Είναι τύπου *timestamp* και έχει προκαθορισμένη τιμή την τρέχουσα ημερομηνία και ώρα τη στιγμή της καταχώρησης (*now()*), εξασφαλίζοντας ότι κάθε άρθρο θα έχει καταχωρημένη τη χρονική στιγμή της δημιουργίας του.

```
create table article
(
    id         serial       not null unique primary key,
    title      varchar(100) not null unique,
    content    text         not null,
    created_at timestamp    not null default now()
);
```

Εικόνα 7. Πίνακας article

Πίνακας *article_tag_association*

Ο πίνακας *article_tag_association* καταγράφει τις συσχετίσεις μεταξύ άρθρων και ετικετών, επιτρέποντας την ανάθεση πολλαπλών ετικετών σε κάθε άρθρο. Αυτό παρέχει τη δυνατότητα κατηγοριοποίησης και ομαδοποίησης των άρθρων, διευκολύνοντας την εξατομίκευση τους. Αναλυτικά:

- ***article_id***: Το μοναδικό αναγνωριστικό του άρθρου. Ορίζεται ως ξένο κλειδί που συσχετίζεται με το πεδίο *id* του πίνακα *article*, εξασφαλίζοντας ότι κάθε ετικέτα αντιστοιχεί σε έναν συγκεκριμένο άρθρο. Το πεδίο είναι υποχρεωτικό.
- ***tag_id***: Το μοναδικό αναγνωριστικό της ετικέτας. Ορίζεται ως ξένο κλειδί που συνδέεται με το πεδίο *id* του πίνακα *tag*, εξασφαλίζοντας ότι κάθε ετικέτα που αντιστοιχίζεται σε ένα άρθρο είναι έγκυρη και υπάρχει στον πίνακα ετικετών. Το πεδίο είναι υποχρεωτικό.

Το πρωτεύον κλειδί του πίνακα αποτελείται από το συνδυασμό των πεδίων *article*_id και *tag_id*, εξασφαλίζοντας ότι η κάθε συσχέτιση μεταξύ ενός άρθρου και μιας ετικέτας είναι μοναδική και δεν μπορεί να υπάρξει διπλή αντιστοίχιση της ίδιας ετικέτας στο ίδιο άρθρο.

```
create table article_tag_association
(
    article_id int not null references article (id),
    tag_id     int not null references tag (id),
    primary key (article_id, tag_id)
);
```

Εικόνα 8. Πίνακας article_tag_association

Πίνακας *announcement*

Ο πίνακας *announcement* περιέχει τα στοιχεία που αφορούν τις ανακοινώσεις που αναρτιόνται στο σύστημα. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε ανακοίνωσης. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***title***: Ο τίτλος της ανακοίνωσης. Είναι τύπου *varchar(100)*, επιτρέποντας μέγιστο μήκος 100 χαρακτήρες. Το πεδίο είναι μοναδικό και υποχρεωτικό, εξασφαλίζοντας ότι δεν θα υπάρχουν δύο ανακοινώσεις με τον ίδιο τίτλο.
- ***content***: Το περιεχόμενο της ανακοίνωσης. Είναι τύπου *text* και είναι υποχρεωτικό. Σε αυτό το πεδίο αποθηκεύεται το κυρίως κείμενο της ανακοίνωσης.
- ***created_at***: Η χρονική σήμανση που υποδεικνύει πότε δημιουργήθηκε η ανακοίνωση. Είναι τύπου *timestamp* και έχει προκαθορισμένη τιμή την τρέχουσα ημερομηνία και ώρα τη στιγμή της καταχώρησης (*now()*), εξασφαλίζοντας ότι κάθε ανακοίνωση θα έχει καταχωρημένη τη χρονική στιγμή της δημιουργίας της.

```
create table announcement
(
    id         serial       not null unique primary key,
    title      varchar(100) not null unique,
    content    text         not null,
    created_at timestamp    not null default now()
);
```

Εικόνα 9. Πίνακας announcement

Πίνακας *appointment*

Ο πίνακας *appointment* καταγράφει τα στοιχεία που αφορούν τις συναντήσεις διατροφολόγου και πελατών. Περιλαμβάνει πληροφορίες για τη συνάντηση, όπως ο τίτλος, η ημερομηνία, και η κατάσταση της συνάντησης, διασφαλίζοντας την ορθή διαχείριση τους. Αναλυτικά:

- ***id***: Το μοναδικό αναγνωριστικό κάθε συνάντησης. Είναι τύπου *serial* και αποτελεί το πρωτεύον κλειδί του πίνακα, και όπως όλα τα πρωτεύοντα κλειδιά, φέρει περιορισμούς μοναδικότητας και μη κενής τιμής.
- ***client_user_info_id***: Το μοναδικό αναγνωριστικό του πελάτη που σχετίζεται με τη συνάντηση, και συνδέεται με τον πίνακα *user_info* μέσω ενός ξένου κλειδιού που αναφέρεται στο πεδίο *id* του πίνακα *user_info*. Το πεδίο είναι υποχρεωτικό και διασφαλίζει τη συσχέτιση κάθε ραντεβού με έναν συγκεκριμένο πελάτη.

- ***title***: Ο τίτλος της συνάντησης, ο οποίος παρέχει μια σύντομη περιγραφή του σκοπού ή της φύσης του. Είναι τύπου *varchar(100)* με μέγιστο μήκος 100 χαρακτήρες και είναι υποχρεωτικό.
- ***description***: Περιέχει μια πιο λεπτομερή περιγραφή για τη συνάντηση, παρέχοντας επιπλέον πληροφορίες. Είναι τύπου *text* και μπορεί να παραμείνει κενό.
- ***appointment_date_time***: Η ημερομηνία και η ώρα της συνάντησης. Είναι τύπου *timestamp* και είναι υποχρεωτικό, καθορίζοντας πότε έχει προγραμματιστεί η συνάντηση.
- ***status****:* Η κατάσταση της συνάντησης, που καθορίζεται από τον τύπο *appointment_status_enum*. Οι αποδεκτές τιμές είναι “PENDING”, “SCHEDULED”, “COMPLETED”, “CANCELLED”, και “DECLINED”. Το πεδίο είναι υποχρεωτικό και αποθηκεύει την τρέχουσα κατάσταση του ραντεβού.

```
create type appointment_status_enum as enum (
    'PENDING', 'SCHEDULED', 'COMPLETED', 'CANCELLED', 'DECLINED'
    );

create table appointment
(
    id                   serial                not null unique primary key,
    client_user_info_id  int                   not null references user_info (id),
    title                varchar(100)          not null,
    description          text,
    appointment_date_time timestamp            not null,
    status               appointment_status_enum not null
);
```

Εικόνα 10. Πίνακας appointment

1.1.2 Hibernate ORM και Spring Data JPA

Hibernate ORM

Το Hibernate ORM (Hibernate, n.d.-b) είναι ένα ORM framework για τη γλώσσα προγραμματισμού Java. Το κύριο καθήκον του είναι να απλοποιεί την διαδικασία αλληλεπίδρασης μεταξύ μιας Java εφαρμογής και μιας σχεσιακής βάσης δεδομένων, όπως είναι η PostgreSQL. Το Hibernate ORM επιτρέπει τη μετατροπή αντικειμένων Java σε εγγραφές πινάκων μιας βάσης δεδομένων και το αντίστροφο, διευκολύνοντας έτσι το έργο του προγραμματιστή. Ακολουθεί ένα παράδειγμα που δείχνει την αντιστοίχιση ενός πίνακα της βάσης δεδομένων με την αντίστοιχη κλάση Java:

```
create table tag_category
(
    id   serial       not null unique primary key,
    name varchar(100) not null unique
);
```

Εικόνα 11. Παράδειγμα Hibernate ORM - Πίνακας βάσης δεδομένων

```
@Entity
@Table(name = "tag_category", schema = "public")
public class TagCategory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", unique = true, nullable = false, updatable = false)
    private Integer id;

    @Column(name = "name", unique = true, nullable = false, length = 100)
    private String name;
}
```

Εικόνα 12. Παράδειγμα Hibernate ORM - Αντίστοιχη κλάση σε Java

Στην εικόνα 11 βλέπουμε τον SQL κώδικα για τη δημιουργία του πίνακα *tag_category*, ενώ στην εικόνα 12 απεικονίζεται ο Java κώδικας για την αντιστοίχιση της κλάσης *TagCategory* με τον πίνακα της βάσης δεδομένων. Γίνεται εύκολα αντιληπτό ότι η αντιστοίχιση της κλάσης με τον πίνακα επιτυγχάνεται μέσω της χρήσης annotations, τα οποία παρέχουν πληροφορίες σχετικά με τη δομή και τη συμπεριφορά των κλάσεων και των πεδίων τους στο Hibernate ORM.

Spring Data JPA

Το Spring Data JPA (VMware Tanzu, n.d.-e), μέρος της μεγαλύτερης οικογένειας Spring Data (VMware Tanzu, n.d.-d), είναι μια επέκταση του JPA (Java Persistence API) που απλοποιεί την υλοποίηση αποθετηρίων δεδομένων (repositories) μέσω της χρήσης διεπαφών (interfaces). Το ιδιαίτερο αυτής της προσέγγισης είναι ότι οι διεπαφές δεν χρειάζονται χειροκίνητη υλοποίηση από τον προγραμματιστή. Το Spring Data JPA αναλαμβάνει αυτόματα να παράγει τη λειτουργικότητα βασισμένη σε μερικούς απλούς κανόνες ονομασίας μεθόδων.

Με αυτόν τον τρόπο, το Spring Data JPA παρέχει έτοιμες μεθόδους για τις πιο κοινές λειτουργίες βάσης δεδομένων, όπως οι λειτουργίες CRUD, μειώνοντας σημαντικά τον κώδικα που απαιτείται για τη διαχείριση δεδομένων. Ένα χαρακτηριστικό παράδειγμα είναι το εξής:

```
public interface TagCategoryRepository extends JpaRepository<TagCategory, Integer>
 {

    TagCategory findByName(String name);

}
```

Εικόνα 13. Παράδειγμα JPA Repository

Σε αυτό το παράδειγμα, το *TagCategoryRepository* είναι μια διεπαφή που επεκτείνει το *JpaRepository* και μας παρέχει αυτόματα βασικές λειτουργίες (όπως save, findAll, delete, κ.λπ.) χωρίς να χρειαστεί να υλοποιηθούν αυτές οι μέθοδοι από τον προγραμματιστή. Επιπλέον, με τη χρήση κανόνων ονομασίας, όπως η μέθοδος *findByName*, το Spring Data JPA παράγει αυτόματα το σχετικό SQL ερώτημα για να επιστρέψει την κατηγορία με το συγκεκριμένο όνομα.

Το Hibernate ORM και το Spring Data JPA συνεργάζονται άψογα για την αλληλεπίδραση με τη βάση δεδομένων. Το Hibernate ORM λειτουργεί ως το υποκείμενο ORM, ενώ το Spring Data JPA παρέχει μια ανώτερη διεπαφή που αυτοματοποιεί πολλές διαδικασίες και επιτρέπει τη γρήγορη και εύκολη πρόσβαση στα δεδομένα. Παράλληλα, το Spring Data JPA προσφέρει επιπλέον λειτουργίες που κάνουν πιο εύκολη τη διαχείριση και την εκτέλεση ερωτημάτων σε επίπεδο repository.

Έτσι, η συνεργασία αυτών των δύο τεχνολογιών επιτρέπει την ανάπτυξη σύγχρονων και ευέλικτων εφαρμογών, μειώνοντας την πολυπλοκότητα του κώδικα και επιτρέποντας στους προγραμματιστές να επικεντρωθούν περισσότερο στη λογική της εφαρμογής παρά στις λεπτομέρειες της βάσης δεδομένων.

1.1.3 Hibernate Validator

Το Hibernate Validator (Hibernate-a, n.d.) είναι ένα ισχυρό εργαλείο στην οικογένεια του Hibernate, το οποίο παρέχει δυνατότητες επικύρωσης δεδομένων για Java εφαρμογές. Αποτελεί την αναφορά υλοποίησης του προτύπου JSR-380 framework και της προδιαγραφής του Jakarta Bean Validation 3.0, η οποία βασίζεται στις δυνατότητες που εισήγαγε το Bean Validation API στην Java EE 7. Το Hibernate Validator επιτρέπει την επικύρωση (validation) των δεδομένων αντικειμένων με έναν τυποποιημένο τρόπο, εξασφαλίζοντας ότι τα δεδομένα πληρούν συγκεκριμένους κανόνες πριν την αποθήκευση ή την επεξεργασία τους στη βάση δεδομένων.

Μέσω του Hibernate Validator, οι προγραμματιστές μπορούν να προσθέτουν κανόνες επικύρωσης στα πεδία, στις κλάσεις, ή στις μεθόδους χρησιμοποιώντας απλά annotations. Τα annotations αυτά βοηθούν στην αυτόματη επικύρωση των δεδομένων πριν από τη μεταφορά τους σε άλλες μεθόδους ή πριν αποθηκευτούν στη βάση δεδομένων, αποτρέποντας σφάλματα και διασφαλίζοντας την ακεραιότητα των δεδομένων.

Ακολουθεί ένα παράδειγμα χρήσης του Hibernate Validator για την επικύρωση των δεδομένων της κλάσης User:

```java
public class User {

    @NotBlank(message = "First Name cannot be empty")
    @Size(min = 2, max = 50, message = "First Name must be between 2 and 50 characters")
    @Pattern(regexp = "^[a-zA-Z]{2,50}$", message = "First Name must contain only letters")
    private String firstName;

    @NotNull(message = "Date of birth is required")
    @Past(message = "Date of birth must be in the past")
    private LocalDate dateOfBirth;

    @Email(message = "Invalid email address")
    private String email;

}
```

Εικόνα 14. Παράδειγμα Hibernate Validator

Σύντομη περιγραφή των annotations

- ***@NotBlank***: Εφαρμόζεται στο πεδίο *firstName* και διασφαλίζει ότι το όνομα του χρήστη δεν θα είναι κενό ή μόνο κενοί χαρακτήρες. Η παράμετρος *message* παρέχει ένα εξατομικευμένο μήνυμα σφάλματος σε περίπτωση μη έγκυρης εισαγωγής.
- ***@Size(min = 2, max = 50)***: Ορίζει το επιτρεπτό εύρος χαρακτήρων για το πεδίο *firstName*. Το όνομα του χρήστη πρέπει να έχει τουλάχιστον 2 και το πολύ 50 χαρακτήρες. Σε περίπτωση μη συμμόρφωσης, εμφανίζεται το μήνυμα που περιγράφει το πρόβλημα.

- ***@Pattern(regexp = "^[a-zA-Z]{2,50}$")***: Χρησιμοποιείται για να εξασφαλίσει ότι το *firstName* αποτελείται μόνο από λατινικούς χαρακτήρες (κεφαλαίους ή μικρούς) και ότι το μήκος του είναι μεταξύ 2 και 50 χαρακτήρων.
- ***@NotNull***: Χρησιμοποιείται στο πεδίο *dateOfBirth* για να βεβαιώσει ότι η ημερομηνία γέννησης δεν είναι null, δηλαδή η ημερομηνία γέννησης είναι υποχρεωτική.
- ***@Past***: Διασφαλίζει ότι η τιμή του πεδίου *dateOfBirth* αντιστοιχεί σε ημερομηνία που βρίσκεται στο παρελθόν, αποτρέποντας τη μελλοντική ημερομηνία ως έγκυρη είσοδο.
- ***@Email***: Εφαρμόζεται στο πεδίο *email* και επικυρώνει ότι η διεύθυνση ηλεκτρονικού ταχυδρομείου που εισάγεται έχει σωστή μορφή σύμφωνα με τα πρότυπα για email.

Προσαρμοσμένη επικύρωση

Εκτός από τα προκαθορισμένα annotations, οι προγραμματιστές μπορούν να δημιουργήσουν τις δικές τους επικυρώσεις, ορίζοντας δικά τους annotations, προσαρμόζοντας έτσι την επικύρωση στις ειδικές ανάγκες της εφαρμογής τους.

Το Hibernate Validator συνεργάζεται άψογα με το Hibernate ORM και χρησιμοποιείται συχνά σε συνδυασμό με το Spring Framework για να εξασφαλίσει ότι τα δεδομένα που διαχειρίζεται το σύστημα είναι έγκυρα πριν την αποθήκευσή τους. Μέσω του Spring Boot, η επικύρωση γίνεται ακόμα πιο εύκολη και αυτόματη σε όλο το σύστημα.

1.1.4 Spring AOP

Το Spring AOP (VMware Tanzu, n.d.-a) είναι ένα σημαντικό κομμάτι του Spring Framework, το οποίο υποστηρίζει τον θεματοστρεφή προγραμματισμό. Ο θεματοστρεφής προγραμματισμός επιτρέπει την απομόνωση και την οργάνωση των λεγόμενων “cross-cutting concerns” από την επιχειρηματική λογική της εφαρμογής. Με τον όρο “cross-cutting concerns” αναφερόμαστε σε διάφορες λειτουργίες που εμφανίζονται σε διάφορα σημεία του κώδικα της εφαρμογής, όπως η ασφάλεια, η καταγραφή συμβάντων, η διαχείριση εξαιρέσεων, και άλλα.

Χωρίς την χρήση του θεματοστρεφή προγραμματισμού, τέτοιες λειτουργίες θα έπρεπε να επαναλαμβάνονται σε διάφορα σημεία του κώδικα, οδηγώντας σε περιττό και επαναλαμβανόμενο κώδικα (γνωστό ως “code scattering”). Αυτό καθιστά τη συντήρηση και τη διαχείριση του κώδικα δύσκολη. Ο θεματοστρεφής προγραμματισμός επιτρέπει την απομόνωση αυτών των concerns σε ξεχωριστές μονάδες, μειώνοντας τον κατακερματισμό του κώδικα και επιτυγχάνοντας κεντρικοποίηση της λογικής.

Κύρια στοιχεία του Spring AOP

- **Aspect:** Μια μονάδα που περιλαμβάνει τη λογική ενός cross-cutting concern, όπως για παράδειγμα η καταγραφή συμβάντων.
- **Join Point:** Το συγκεκριμένο σημείο στο πρόγραμμα που μπορεί να εφαρμοστεί ένα aspect. Στο Spring, τα join points είναι οι μέθοδοι.
- **Advice:** Η λογική που θέλουμε να εκτελείται σε ένα συγκεκριμένο join point. Τα advice μπορεί να εκτελούνται πριν ή μετά την εκτέλεση μιας μεθόδου ή όταν προκύπτει εξαίρεση.
- **Pointcut:** Ένα κατηγόρημα (predicate) που ελέγχει και επιλέγει ποια join points ταιριάζουν με αυτό. Ένα advice συσχετίζεται με ένα pointcut και εκτελείται στα σημεία που αντιστοιχούν στο pointcut.

Στο παρακάτω παράδειγμα, χρησιμοποιούμε το Spring AOP για την αυτόματη καταγραφή συμβάντων όταν προκύπτει εξαίρεση σε κλήσεις του service layer.

```java
@Aspect
@Slf4j
public class LoggingAspect {

    @Pointcut("execution(public * gr.unipi.thesis.dimstyl.services.impl.*.*(..))")
    public void serviceLayerMethods() {
    }

    @AfterThrowing(pointcut = "serviceLayerMethods()", throwing = "ex")
    public void logServiceLayerException(JoinPoint joinPoint, Throwable ex) throws Throwable {
        String methodName = joinPoint.getSignature().toLongString();

        // Log the exception message
        log.error("Exception thrown in method: {}, exception message: {}", methodName, ex.getMessage());

        // Check if the exception has a cause and log it
        Throwable cause = ex.getCause();
        if (cause != null) log.error("Cause of the exception: {}", cause.getMessage());

        // Re-throw the exception to maintain the original behavior
        throw ex;
    }

}
```

Εικόνα 15. Παράδειγμα χρήσης Spring AOP

Σύντομη επεξήγηση κώδικα

- ***@Aspect***: Δηλώνει ότι αυτή η κλάση είναι ένα aspect, δηλαδή περιέχει τη λογική που θα εκτελεστεί σε συγκεκριμένα σημεία του κώδικα (join points).
- ***@Slf4j***: Παρέχεται από τη βιβλιοθήκη Lombok και δημιουργεί αυτόματα έναν logger για την καταγραφή συμβάντων, χωρίς να χρειάζεται η χειροκίνητη δήλωση του.
- ***@Pointcut***: Ορίζει το σημείο εφαρμογής του. Η έκφραση *execution(public * gr.unipi.thesis.dimstyl.services.impl.*.*(..)))* δηλώνει ότι το pointcut θα εφαρμοστεί σε όλες τις δημόσιες μεθόδους του πακέτου *gr.unipi.thesis.dimstyl.services.impl*.
- ***@AfterThrowing***: Το advice που εκτελείται όταν προκύπτει εξαίρεση σε κάποια από τις μεθόδους που αντιστοιχούν στο pointcut. Η παράμετρος *throwing = "ex"* επιτρέπει την πρόσβαση στην εξαίρεση που προέκυψε.
- ***logServiceLayerException***: Αυτή η μέθοδος εκτελείται κάθε φορά που προκύπτει εξαίρεση στις μεθόδους του service layer. Η κλήση *joinPoint.getSignature().toLongString()* επιστρέφει το πλήρες όνομα της μεθόδου όπου προέκυψε η εξαίρεση και το *log.error* καταγράφει την εξαίρεση. Τέλος, η εξαίρεση επανεκδίδεται ώστε συνεχιστεί η κανονική ροή της εφαρμογής.

Με τον παραπάνω τρόπο επιτυγχάνεται αυτόματη καταγραφή των εξαιρέσεων χωρίς ο προγραμματιστής να πρέπει να προσθέσει χειροκίνητα σε κάθε μέθοδο κώδικα για την καταγραφή πιθανών εξαιρέσεων.

1.1.5 Spring Security

Το Spring Security (VMware Tanzu, n.d.-g) είναι ένα από τα πιο δημοφιλή και ισχυρά frameworks για την ασφάλεια Java εφαρμογών που αναπτύσσονται με Spring. Παρέχει ολοκληρωμένες λύσεις για αυθεντικοποίηση και εξουσιοδότηση, επιτρέποντας στους προγραμματιστές να προστατεύσουν τις εφαρμογές τους με σχετική λίγη προσαρμογή. Οι λύσεις αυτές περιλαμβάνουν κωδικοποίηση κωδικών πρόσβασης (password encoding), έλεγχο ταυτότητας μέσω πολλαπλών μηχανισμών (π.χ.

Basic Authentication, JWT, OAuth2), και λεπτομερή έλεγχο προσβάσεων σε συγκεκριμένες λειτουργίες ή πόρους εφαρμογής.

Η ευκολία παραμετροποίησης και επεκτασιμότητας του Spring Security είναι η κύρια δύναμη του. Η ενσωμάτωση του με το Spring Boot γίνεται σχεδόν αυτόματα, προσθέτοντας ένα προκαθορισμένο επίπεδο ασφάλειας από την αρχή, ακόμα και χωρίς επιπλέον ρυθμίσεις. Αυτό επιτρέπει τη γρήγορη προσθήκη βασικών μηχανισμών ασφαλείας στην εφαρμογή.

Κύρια χαρακτηριστικά Spring Security

- **Αυθεντικοποίηση:** Αυτή είναι η διαδικασία επαλήθευσης της ταυτότητας ενός χρήστη, η οποία υποστηρίζεται από διάφορους μηχανισμούς όπως ο βασικός έλεγχος ταυτότητας (Basic Authentication), η αυθεντικοποίηση μέσω φόρμας (Form-based Login), το OAuth2, και τα JSON Web Tokens. Το Spring Security μπορεί να προσαρμοστεί ώστε να υποστηρίζει σύνδεση μέσω εξωτερικών παρόχων (π.χ. Google, Facebook, GitHub) χρησιμοποιώντας το OAuth2.
- **Εξουσιοδότηση:** Μόλις γίνει αυθεντικοποίηση του χρήστη, το Spring Security ελέγχει τα δικαιώματα του χρήστη για να διασφαλίσει ότι έχει πρόσβαση μόνο στους πόρους ή τις λειτουργίες που του επιτρέπονται. Οι κανόνες εξουσιοδότησης καθορίζουν ποιος έχει πρόσβαση σε συγκεκριμένα URLs ή ενέργειες ανάλογα με τους ρόλους χρηστών (π.χ. DIETITIAN, CLIENT).
- **Κωδικοποίηση κωδικών:** Το Spring Security παρέχει μηχανισμούς κωδικοποίησης και επαλήθευσης κωδικών πρόσβασης για την αποθήκευσή τους σε ασφαλή μορφή στη βάση δεδομένων.
- **Φίλτρα ασφάλειας (Security filters):** Το Spring Security βασίζεται σε φίλτρα για να επεξεργάζεται και να προστατεύει τα αιτήματα HTTP. Αυτά τα φίλτρα μπορούν να τροποποιηθούν και να προσαρμοστούν ανάλογα με τις απαιτήσεις της εφαρμογής, επιτρέποντας την εφαρμογή ειδικών κανόνων ασφαλείας.

Java JWT Library

Στην παρούσα εργασία για την ασφάλεια του συστήματος χρησιμοποιήθηκε και η βιβλιοθήκη Java JWT Library (Java JWT, n.d.), η οποία συνεργάζεται με το Spring Security για την προστασία των αιτημάτων που προέρχονται από την κινητή εφαρμογή των πελατών. Το JWT προσφέρει έναν stateless τρόπο αυθεντικοποίησης, καθώς οι πληροφορίες του χρήστη είναι ενσωματωμένες στο ίδιο το token. Έτσι, δεν απαιτείται η αποθήκευση συνεδριών (sessions) ή η χρήση μπισκότων (cookies) και άλλων παραδοσιακών μηχανισμών αποθήκευσης συνεδριών, επιτρέποντας πιο αποτελεσματική διαχείριση της ασφάλειας στα APIs της εφαρμογής.

1.1.6 Spring Email

Το Spring Email (VMware Tanzu, n.d.-b) επιτρέπει την αποστολή email από Java εφαρμογές με απλό και ευέλικτο τρόπο. Χρησιμοποιώντας το Spring Mail API, μπορούμε να αποστέλλουμε emails σε διάφορες μορφές, όπως απλό κείμενο, HTML κείμενο και emails με συνημμένα αρχεία, μέσω της ενσωμάτωσης με SMTP servers.

1.1.7 Project Lombok

Το Project Lombok (https://projectlombok.org) είναι ένα εργαλείο που απλοποιεί την ανάπτυξη Java κώδικα μέσω της χρήσης annotations, μειώνοντας τον επαναλαμβανόμενο κώδικα και καθιστώντας τον πιο ευανάγνωστο. Με τη χρήση του Project Lombok, μπορούμε να αποφύγουμε τη χειροκίνητη

δημιουργία μεθόδων όπως getters, setters, constructors και άλλων κοινών στοιχείων, επιτρέποντας έτσι στους προγραμματιστές να εστιάσουν στην επιχειρηματική λογική της εφαρμογής.

Ακολουθεί ένα σύντομο παράδειγμα χρήσης του Project Lombok:

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {

    private String firstName;
    private String lastName;

}
```

Εικόνα 16. Παράδειγμα χρήσης Project Lombok

Σύντομη επεξήγηση κώδικα

- ***@Getter***: Δημιουργεί αυτόματα τις getter μεθόδους για όλα τα πεδία της κλάσης.
- ***@Setter***: Δημιουργεί αυτόματα τις setter μεθόδους για όλα τα πεδία της κλάσης.
- ***@NoArgsConstructor***: Δημιουργεί έναν constructor χωρίς παραμέτρους.
- ***@AllArgsConstructors***: Δημιουργεί έναν constructor με όλα τα πεδία της κλάσης ως παραμέτρους.

Με αυτό τον τρόπο, η κλάση παραμένει απλή και ευανάγνωστη, χωρίς την ανάγκη να γράψουμε τον επαναλαμβανόμενο κώδικα που απαιτείται συνήθως για τη διαχείριση των πεδίων.

1.1.8 Thymeleaf

Το Thymeleaf (https://www.thymeleaf.org) είναι ένα ισχυρό και ευέλικτο server-side templating engine για εφαρμογές Java, που χρησιμοποιείται για τη δημιουργία δυναμικών ιστοσελίδων. Ο βασικός του στόχος είναι να επιτρέπει στους προγραμματιστές να ενσωματώνουν δεδομένα και λογική μέσα σε αρχεία HTML με τρόπο που καθιστά τον κώδικα εύκολα αναγνώσιμο και συντηρήσιμο.

Ένα από τα κύρια χαρακτηριστικά του Thymeleaf είναι η φυσική προσέγγιση των προτύπων (natural templates), δηλαδή τα HTML αρχεία μπορούν να είναι πλήρως λειτουργικά ως στατικές σελίδες, ακόμη και χωρίς τη διαδικασία του rendering από τον server. Αυτό διευκολύνει τους web designers και τους προγραμματιστές να δουλεύουν ταυτόχρονα στα ίδια αρχεία.

Το Thymeleaf ενσωματώνεται εύκολα με το Spring Boot, παρέχοντας αυτοματοποιημένη διαχείριση για την εμφάνιση δεδομένων στις ιστοσελίδες με ελάχιστη ρύθμιση. Στηρίζεται σε expressions για την παρουσίαση δεδομένων, την εμφάνιση συνθηκών, τη δημιουργία συνδέσμων, και άλλα.

1.2 Εφαρμογή για κινητά

Για την υλοποίηση της εφαρμογής για κινητά χρησιμοποιήθηκε το Jetpack Compose (Google for Developers, n.d.), το οποίο είναι το σύγχρονο toolkit της Google για τη δημιουργία διεπαφής χρήστη (UI) σε Android. Το Jetpack Compose, σε αντίθεση με την παραδοσιακή προσέγγιση που χρησιμοποιεί XML για την δημιουργία του περιβάλλοντος χρήστη, επιτρέπει στους προγραμματιστές να γράφουν το UI απευθείας με Kotlin, παρέχοντας ένα πιο συνεκτικό και αποτελεσματικό εργαλείο για την ανάπτυξη Android εφαρμογών.

Ένα από τα κύρια πλεονεκτήματα του Jetpack Compose είναι η απλότητα και η παραγωγικότητα που προσφέρει. Η ανάπτυξη UI γίνεται με λιγότερο κώδικα σε σχέση με τις παλαιότερες μεθόδους, ενώ η χρήση της Kotlin και στο UI και στη λογική της εφαρμογής επιτρέπει καλύτερη συνέπεια στον κώδικα. Επιπλέον, το Jetpack Compose προσφέρει δυναμική και επαναχρησιμοποιήσιμη λογική UI μέσω των λεγόμενων “composables”.

Η επιλογή της Kotlin ως γλώσσα προγραμματισμού ήταν προφανής, καθώς η Kotlin είναι η προτεινόμενη γλώσσα από την Google για την ανάπτυξη Android εφαρμογών, με σαφή πλεονεκτήματα σε σχέση με την Java. Εκτός από το γεγονός ότι η Kotlin είναι πιο λιτή και εκφραστική, παρέχει καλύτερη διαχείριση σφαλμάτων και ασφάλεια τύπων (type safety), γεγονός που μειώνει τον κίνδυνο σφαλμάτων (bugs) και βελτιώνει την ποιότητα του κώδικα. Αυτό το ενιαίο περιβάλλον για τη δημιουργία του UI και της λογικής της εφαρμογής διευκολύνει τον προγραμματιστή, καθώς δεν απαιτείται πλέον ο διαχωρισμός του UI (σε XML) από την λογική της εφαρμογής (σε Kotlin/Java), κάτι που ήταν μια περίπλοκη διαδικασία στο παρελθόν.

Η επικοινωνία και η ανταλλαγή δεδομένων της κινητής εφαρμογής με τον διακομιστή υποστήριξης γίνεται μέσω RESTful API κλήσεων, χρησιμοποιώντας HTTP αιτήματα για την αποστολή και λήψη δεδομένων. Για την ασφάλεια αυτών των αιτημάτων, κάθε HTTP αίτημα περιλαμβάνει μια επικεφαλίδα εξουσιοδότησης (authorization header) που φέρει το JWT, το οποίο εξασφαλίζει την αυθεντικοποίηση και εξουσιοδότηση του χρήστη. Αυτό διασφαλίζει ότι τα δεδομένα και οι υπηρεσίες της εφαρμογής είναι προσβάσιμα μόνο από εξουσιοδοτημένους χρήστες.

Στη συνέχεια θα παρουσιαστεί η δομή της εφαρμογής καθώς και οι τεχνικές υλοποίησης που ακολουθήθηκαν για την επίτευξη των λειτουργιών της.

1.2.1 Διαχείριση κατάστασης UI (UI State Management)

Η διαχείριση της κατάστασης του UI αποτελεί κρίσιμο παράγοντα όχι μόνο για την αποδοτική λειτουργία αλλά και για την εμπειρία του χρήστη σε εφαρμογές Android και γενικότερα σε κινητές εφαρμογές. Το Jetpack Compose διευκολύνει την διαχείριση της κατάστασης του UI μέσω των εννοιών “state” και “state hoisting” (Android Developers, 2024e), προσφέροντας έναν πιο συνεπή και απλοποιημένο τρόπο για τον έλεγχο της κατάστασης των composables. Αυτή προσέγγιση επιτρέπει στους προγραμματιστές να διαχειρίζονται την κατάσταση με ευκολία, μειώνοντας τον επαναλαμβανόμενο κώδικα και βελτιώνοντας την συντηρησιμότητα της εφαρμογής.

Επιπρόσθετα, η αποτελεσματική διαχείριση του κύκλου ζωής - lifecycle (Android Developers, 2024d) της εφαρμογής διασφαλίζει ότι το UI παραμένει συγχρονισμένο με την κατάσταση του composable, ανάλογα με τη φάση κύκλου ζωής στην οποία βρίσκεται. Αυτό μειώνει τον κίνδυνο διαρροών μνήμης (memory leaks) και προβλημάτων που σχετίζονται με την ασύγχρονη ενημέρωση της κατάστασης μνήμης του UI.

1.2.2 Πλοήγηση (Navigation)

Για την υλοποίηση της πλοήγησης μεταξύ των διάφορων οθονών της εφαρμογής, κάθε μία από τις οποίες αποτελεί ένα composable, χρησιμοποιήθηκε το Navigation component (Android Developers, 2024c) του Jetpack Compose. Αυτό το εργαλείο προσφέρει έναν απλό και αποτελεσματικό τρόπο πλοήγησης, διευκολύνοντας τη μετάβαση μεταξύ των οθονών χωρίς την ανάγκη σύνταξης περίπλοκου κώδικα, όπως συνέβαινε με τις παραδοσιακές προσεγγίσεις στο Android.

Μέσω του Navigation component, διευκολύνεται η διαχείριση της στοίβας πλοήγησης (back stack), η οποία διατηρεί τις ήδη επισκεπτόμενες οθόνες του χρήστη. Επίσης, υποστηρίζεται η πλοήγηση με παραμέτρους και η υλοποίηση πιο περίπλοκων πλοηγήσεων εντός και εκτός της εφαρμογής. Η χρήση αυτής της βιβλιοθήκης καθιστά τη συντήρηση της εφαρμογής πιο εύκολη και πιο διαχειρίσιμη.

1.2.3 Διαχείριση δεδομένων και αλληλεπίδραση με τον διακομιστή

Η αλληλεπίδραση με τον διακομιστή πραγματοποιείται μέσω RESTful API κλήσεων. Κάθε αίτημα αποστέλλεται μέσω HTTP και συνοδεύεται από την κατάλληλη επικεφαλίδα εξουσιοδότησης (JWT), η οποία εξασφαλίζει την αυθεντικοποίηση και εξουσιοδότηση του χρήστη. Για την υλοποίηση αυτών των κλήσεων χρησιμοποιούνται οι βιβλιοθήκες Retrofit (Square, Inc., n.d.-a), OkHttp (Block, Inc., n.d.) και Gson Converter (Square, Inc., n.d.-b).

Η βιβλιοθήκη OkHttp είναι διαχειρίζεται τα αιτήματα και τις επικεφαλίδες εξουσιοδότησης, διασφαλίζοντας την ασφαλή αποστολή και παραλαβή των δεδομένων. Η Gson Converter είναι υπεύθυνη για τη σειριοποίηση και αποσειριοποίηση (serialization and deserialization) των δεδομένων σε μορφή JSON, ενώ η Retrofit αναλαμβάνει την αποστολή των αιτημάτων και την λήψη των απαντήσεων, συνεργαζόμενη με τις υπόλοιπες βιβλιοθήκες για την πραγμάτωση των RESTful κλήσεων.

Για τη λήψη του προγράμματος διατροφής του χρήστη, χρησιμοποιείται η υπηρεσία συστήματος του Android, Download Manager (Android Developers, 2024b), η οποία χειρίζεται μακροχρόνιες HTTP λήψεις. Ομοίως, όπως και στις υπόλοιπες κλήσεις, προστίθεται η επικεφαλίδα εξουσιοδότησης στο HTTP αίτημα.

Για την προβολή περιεχομένου, όπως άρθρα ή ανακοινώσεις, αξιοποιείται το composable AndroidView (Android Developers, 2024f), το οποίο χρησιμοποιεί ένα WebView (Android Developers, 2024g) για την εμφάνιση του περιεχομένου. Και εδώ επίσης προστίθεται η επικεφαλίδα εξουσιοδότησης για την εξασφάλιση της σωστής πρόσβασης.

1.2.4 Αποθήκευση, ανάκτηση, και διαγραφή JSON Web Token

Η αποθήκευση, ανάκτηση και διαγραφή του JWT πραγματοποιείται μέσω της βιβλιοθήκης DataStore (Android Developers, 2024a). Το DataStore επιλέχθηκε αντί του παραδοσιακού SharedPreferences, καθώς το Android πλέον συνιστά τη χρήση του ως την πιο σύγχρονη και αποδοτική λύση για τη διαχείριση δεδομένων. Σε αντίθεση με το SharedPreferences, το DataStore είναι χτισμένο πάνω σε Kotlin coroutines και Flow, προσφέροντας καλύτερη διαχείριση ασύγχρονων λειτουργιών και αποφυγή προβλημάτων που προκύπτουν από τον ταυτόχρονο χειρισμό δεδομένων.

Αυτά τα χαρακτηριστικά επιτρέπουν στο DataStore να διαχειρίζεται με ασφάλεια την αποθήκευση δεδομένων όπως το JWT, εξασφαλίζοντας παράλληλα μεγαλύτερη ανθεκτικότητα και σταθερότητα στην απόδοση της εφαρμογής.

2. Μεθοδολογική Προσέγγιση

2.1 Διαχείριση έργου

Για τη διαχείριση του έργου, χρησιμοποιήθηκαν τρία κύρια εργαλεία: το Git (https://git-scm.com), ένα σύστημα ελέγχου εκδόσεων (Version Control System - VCS), το GitHub (https://github.com), μια διαδικτυακή πλατφόρμα που υποστηρίζει το Git, και το Jira Software (Atlassian, n.d.), που προσφέρει τη δυνατότητα αλληλεπίδρασης με το GitHub. Το Jira χρησιμοποιήθηκε για την οργάνωση και την παρακολούθηση της προόδου του έργου, μέσω της μεθοδολογίας Kanban.

Η μεθοδολογία Kanban είναι μια προσέγγιση ανάπτυξης λογισμικού που εντάσσεται στις ευέλικτες μεθοδολογίες (Agile methodologies). Το Kanban απαιτεί επικοινωνία σε πραγματικό χρόνο και προωθεί την πλήρη διαφάνεια της εργασίας. Τα διάφορα αντικείμενα εργασίας αναπαρίστανται οπτικά σε έναν πίνακα, γνωστό ως kanban board, δίνοντας στα μέλη της ομάδας τη δυνατότητα να παρακολουθούν την κατάσταση οποιουδήποτε μέρους της δουλειάς ανά πάσα στιγμή.

Ένα βασικό χαρακτηριστικό του kanban board είναι ότι βοηθά στην οπτικοποίηση και βελτιστοποίηση της ροής της εργασίας (workflow) μεταξύ των ομάδων. Είτε χρησιμοποιείται ένας φυσικός πίνακας είτε ένας ψηφιακός, όπως αυτός του Jira, το Kanban επιτρέπει στην ομάδα να απεικονίζει τη δουλειά της, να τυποποιεί τη διαδικασία εργασίας της και να αναγνωρίζει έγκαιρα τα πιθανά εμπόδια ή εξαρτήσεις.

Η κλασική ροή εργασίας σε ένα kanban board αποτελείται από τρία στάδια: Do, In Progress, και Done. Παρόλα αυτά, η ροή εργασίας μπορεί να προσαρμοστεί ανάλογα με το μέγεθος και τις ανάγκες της ομάδας, διασφαλίζοντας ότι η διαδικασία υποστηρίζει τους συγκεκριμένους στόχους της.

Οι θεμελιώδεις αρχές της μεθοδολογίας Kanban (Koceski, 2016, p. 29) είναι οι εξής:

- Ξεκινήστε με αυτό που έχετε τώρα – η τρέχουσα διαδικασία είναι η βάση για βελτιώσεις
- Συμφωνήστε να ακολουθήσετε μια εξελικτική προσέγγιση για την αλλαγή και βελτίωση
- Σεβαστείτε τους τρέχοντες ρόλους και τις ευθύνες της ομάδας ή του οργανισμού

Βάσει αυτών των αρχών, το επόμενο βήμα είναι η υιοθέτηση των πέντε βασικών αρχών του Kanban (Koceski, 2016, p. 29):

- Οπτικοποιήστε την εργασία και τη ροή εργασίας που ακολουθεί
- Περιορίστε την εργασία σε εξέλιξη (Work-In-Progress – WIP) χρησιμοποιώντας ένα εικονικό σύστημα Kanban
- Διαχειριστείτε τη ροή της εργασίας
- Κάντε σαφείς τις πολιτικές διαχείρισης
- Χρησιμοποιήστε μοντέλα και την επιστημονική μέθοδο για να βελτιώνεστε μέσα από τη συνεργασία

2.2 Αρχιτεκτονική σχεδίαση διακομιστή υποστήριξης

Για την υλοποίηση του διακομιστή υποστήριξης, επιλέχθηκε το μοντέλο αρχιτεκτονικής λογισμικού MVC (Model-View-Controller), καθώς ο τρόπος εφαρμογής του διευκολύνει την ανάπτυξη της διαδικτυακής πλατφόρμας. Το MVC διαιρεί την εφαρμογή σε τρία διασυνδεδεμένα μέρη, επιτρέποντας τον διαχωρισμό της παρουσίασης της πληροφορίας που παρέχεται στον χρήστη, από τη μορφή με την οποία είναι αποθηκευμένη (π.χ. στη βάση δεδομένων). Τα τρία μέρη που συνθέτουν το μοντέλο MVC είναι τα εξής:

- **Model**: Αποτελεί τον πυρήνα της εφαρμογής, ο οποίος διαχειρίζεται τα δεδομένα.
- **View**: Είναι υπεύθυνο για την αναπαράσταση της πληροφορίας που προέρχεται από το Model μέσω μιας διεπαφής χρήστη.
- **Controller**: Λειτουργεί ως μεσολαβητής μεταξύ Model και View, μεταδίδοντας την πληροφορία από το Model στο View και αντίστροφα.

Στην τρέχουσα εργασία, το Model είναι περαιτέρω διαχωρισμένο σε επιμέρους επίπεδα, όπως repositories και services, προκειμένου να οργανωθούν καλύτερα οι διαφορετικές λειτουργίες και οι αλληλεπιδράσεις με τη βάση δεδομένων. Επιπλέον, το επίπεδο Controller διαχωρίζεται και αυτό περαιτέρω σε web και API controllers, καθώς μέσω του ιδίου διακομιστή εξυπηρετούνται και οι RESTful API κλήσεις που έρχονται από την κινητή εφαρμογή.

2.3 Αρχιτεκτονική σχεδίαση εφαρμογής για κινητά

Για την ανάπτυξη της κινητής εφαρμογής, υιοθετήθηκε το μοντέλο Clean Architecture, το οποίο αποτελεί επέκταση και βελτίωση του αρχιτεκτονικού μοντέλου MVVM (Model-View-ViewModel). Το Clean Architecture προσφέρει έναν πιο σαφή διαχωρισμό των επιπέδων της εφαρμογής, διασφαλίζοντας ότι η λογική, τα δεδομένα και η διεπαφή χρήστη είναι πλήρως απομονωμένα μεταξύ τους. Τα επίπεδα του Clean Architecture είναι τα εξής:

- **Presentation Layer (View και ViewModel)**: Το Presentation Layer είναι υπεύθυνο για την εμφάνιση των δεδομένων στον χρήστη. Περιλαμβάνει το View και ViewModel, τα οποία διαχειρίζονται την παρουσίαση της πληροφορίας και την αλληλεπίδραση με τον χρήστη, χωρίς να ασχολούνται με την επιχειρηματική λογική ή τα δεδομένα.
- **Domain Layer**: Αποτελεί τον πυρήνα της εφαρμογής και περιέχει την επιχειρηματική λογική. Εδώ βρίσκονται τα Use Cases, που αντιπροσωπεύουν τις επιχειρηματικές ροές της εφαρμογής και διασφαλίζουν την ορθή επεξεργασία των δεδομένων, ανεξάρτητα πως εμφανίζονται στη διεπαφή χρήστη.
- **Data Layer**: Το Data Layer είναι υπεύθυνο για τη διαχείριση των δεδομένων και την αλληλεπίδραση με τοπικές βάσεις δεδομένων ή εξωτερικές πηγές όπως απομακρυσμένους διακομιστές. Σε αυτό το επίπεδο βρίσκονται τα repositories, τα οποία διαχειρίζονται αυτές τις αλληλεπιδράσεις και την διαχείριση των δεδομένων.

Η αρχιτεκτονική Clean Architecture προσφέρει σημαντικά πλεονεκτήματα, όπως:

- **Διαχωρισμός ανησυχιών (Separation of Concerns)**: Κάθε επίπεδο της εφαρμογής είναι πλήρως απομονωμένο από τα υπόλοιπα, διευκολύνοντας την ανεξάρτητη συντήρηση και επέκταση της εφαρμογής.
- **Επαναχρησιμοποίηση και επεκτασιμότητα**: Ο διαχωρισμός των επιπέδων επιτρέπει την εύκολη επαναχρησιμοποίηση του κώδικα, καθώς και την εύκολη αντικατάσταση ή αναβάθμιση επιμέρους τμημάτων, χωρίς να επηρεάζονται τα υπόλοιπα.
- **Διασφάλιση ποιότητας κώδικα:** Η αρχιτεκτονική αυτή διευκολύνει τη συγγραφή μονάδων κώδικα που μπορούν να δοκιμαστούν αυτόνομα, επιτρέποντας πιο αξιόπιστες δοκιμές και διασφάλιση της ποιότητας.

3. Αποτελέσματα

Η ανάγκη για μια πιο αποτελεσματική διαχείριση της επικοινωνίας και των δεδομένων μεταξύ διατροφολόγου και πελατών οδήγησε στην ανάπτυξη ενός συστήματος που απλοποιεί τις διαδικασίες αυτές και προσφέρει εύχρηστα εργαλεία και στους δύο χρήστες. Το σύστημα αποτελείται από δύο εφαρμογές, μια διαδικτυακή πλατφόρμα που χρησιμοποιεί ο διατροφολόγος και μια εφαρμογή για κινητά που χρησιμοποιούν οι πελάτες. Οι βασικές λειτουργίες κάθε εφαρμογής είναι οι εξής:

Βασικές λειτουργίες διαδικτυακής πλατφόρμας

1. Προβολή πελατών
2. Προσθήκη νέου πελάτη στο σύστημα
3. Διαγραφή πελάτη από το σύστημα
4. Επεξεργασία στοιχείων πελάτη
5. Προσθήκη ετικετών ανά πελάτη
6. Μεταφόρτωση, προβολή και διαγραφή προγραμμάτων διατροφής ανά πελάτη
7. Ανάρτηση, επεξεργασία, προβολή και διαγραφή ανακοινώσεων
8. Ανάρτηση, επεξεργασία, προβολή και διαγραφή εξατομικευμένων άρθρων
9. Προβολή συναντήσεων
10. Καταχώρηση νέας συνάντησης με πελάτη
11. Επεξεργασία στοιχείων συνάντησης
12. Ακύρωση, απόρριψη, αποδοχή και ολοκλήρωση συνάντησης

Βασικές λειτουργίες εφαρμογής για κινητά

1. Επεξεργασία προσωπικών στοιχείων
2. Λήψη προγραμμάτων διατροφής
3. Προβολή συναντήσεων
4. Δημιουργία νέου αιτήματος συνάντησης
5. Ακύρωση συνάντησης
6. Προβολή άρθρων
7. Προβολή ανακοινώσεων

Τέλος, η χρήση του Jetpack Compose επέτρεψε τη γρήγορη ανάπτυξη της εφαρμογής για κινητά, διευκολύνοντας και απλοποιώντας την υλοποίηση των γραφικών στοιχείων (UI). Συγκριτικά με την παραδοσιακή ανάπτυξη εφαρμογών Android, η προσέγγιση αυτή επέτρεψε την ταχύτερη επίτευξη λειτουργικότητας και βελτίωσε την εμπειρία ανάπτυξης.

4. Συμπεράσματα

Οι αρχικοί στόχοι της εργασίας περιλάμβαναν την υλοποίηση βασικών λειτουργιών για τη διαχείριση των πελατών, των συναντήσεων, των προγραμμάτων διατροφής, καθώς και την ανάρτηση άρθρων και ανακοινώσεων, με στόχο τη βελτίωση της εμπειρίας χρήστη και την αμεσότερη επικοινωνία και αλληλεπίδραση με τους πελάτες.

Η υλοποίηση των λειτουργιών έγινε με επιτυχία, προσφέροντας ένα πολύ καλό αποτέλεσμα τόσο στην διαδικτυακή πλατφόρμα όσο και στην κινητή εφαρμογή. Ωστόσο, κατά τη διάρκεια της ανάπτυξης συναντήθηκαν διάφορες προκλήσεις, όπως η ασφάλεια του διακομιστή υποστήριξης. Ήταν απαραίτητο να περιορίζεται η πρόσβαση σε πόρους ανάλογα με τον ρόλο του χρήστη και την προέλευση του αιτήματος, είτε από την διαδικτυακή πλατφόρμα είτε από την εφαρμογή για κινητά.

Σε γενικές γραμμές, οι αρχικοί στόχοι επιτεύχθηκαν, με τις περισσότερες λειτουργίες να είναι πλήρως υλοποιημένες και λειτουργικές. Η συνεισφορά της εργασίας έγκειται στη δημιουργία ενός ολοκληρωμένου συστήματος που ενισχύει την αλληλεπίδραση και την επικοινωνία του διατροφολόγου με τους πελάτες του, καθιστώντας τη διαδικασία πιο αποτελεσματική και ευχάριστη.

Η μελλοντική έρευνα θα πρέπει να εστιάσει στην προσθήκη αυτόματων ειδοποιήσεων για τα επερχόμενα ραντεβού, στην ανταλλαγή μηνυμάτων σε πραγματικό χρόνο, καθώς και σε πιο εξατομικευμένες προσεγγίσεις, όπως η ανάλυση δεδομένων για την πρόοδο των πελατών και η υπενθύμιση της τήρησης της διατροφής τους.

5. Πίνακας Ορολογίας

Ξενόγλωσσος όρος	Ελληνικός όρος
Backend server	Διακομιστής υποστήριξης
Enterprise application	Εφαρμογή για επιχειρήσεις
Application-level business logic	Επιχειρηματική λογική σε επίπεδο εφαρμογής
Deployment environment	Περιβάλλον εγκατάστασης και εκτέλεσης
Database Management System	Σύστημα Διαχείρισης Βάσης Δεδομένων
Relational Database Management System	Σχεσιακό Σύστημα Διαχείρισης Βάσης Δεδομένων
Object/Relational Mapping	Αντικειμενοστραφής/Σχεσιακή Αντιστοίχιση
Authentication and authorization	Αυθεντικοποίηση και εξουσιοδότηση
Logging	Καταγραφή συμβάντων
Exception Handling	Διαχείριση εξαιρέσεων
Annotation	Σχολιασμός
Server-side	Πλευρά του διακομιστή
Boilerplate code	Επαναλαμβανόμενος κώδικας
Client and server	Πελάτης και διακομιστής
Reliability	Αξιοπιστία
Scalability	Επεκτασιμότητα
SQL standard	Πρότυπο SQL
Entity	Οντότητα
Relationship	Συσχέτιση
Integrity constraint	Περιορισμός ακεραιότητας
Primary key	Πρωτεύον κλειδί
Foreign key	Ξένο κλειδί
Unique constraint	Περιορισμός μοναδικότητας
Not null constraint	Περιορισμός μη κενής τιμής
Data consistency and integrity	Συνέπεια και ακεραιότητα δεδομένων
Repository	Αποθετήριο
Interface	Διεπαφή
Validation	Επικύρωση
Password encoding	Κωδικοποίηση κωδικού πρόσβασης
Basic authentication	Βασικός έλεγχος ταυτότητας
Form-based login	Αυθεντικοποίηση μέσω φόρμας
Security filters	Φίλτρα ασφάλειας
Cookie	Μπισκότο
Session	Συνεδρία
Predicate	Κατηγόρημα
Type safety	Ασφάλεια τύπων
Bug	Σφάλμα

Authorization header	Επικεφαλίδα εξουσιοδότησης
UI state management	Διαχείριση κατάστασης διεπαφής χρήστη
Memory leak	Διαρροή μνήμης
Navigation	Πλοήγηση
Serialization and deserialization	Σειριοποίηση και αποσειριοποίηση
Version control	Σύστημα ελέγχου εκδόσεων
Agile methodology	Ευέλικτη μεθοδολογία
Workflow	Ροή εργασίας
Separation of concerns	Διαχωρισμός ανησυχιών

6. Πίνακας Συντμήσεων-Αρκτικόλεξων-Ακρωνύμιων

MVC	Model-View-Controller
API	Application Programming Interface
RESTful APIs	Representational State Transfer API
CRUD	Create, Read, Update, Delete
JPA	Java Persistence API
ORM	Object/Relational Mapping
AOP	Aspect-Oriented Programming
HTML	Hypertext Markup Language
JWT	JSON Web Token
SQL	Structured Query Language
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
SMTP	Simple Mail Transfer Protocol
UI	User Interface
XML	Extensible Markup Language
JSON	JavaScript Object Notation
VCS	Version Control System
MVVM	Model-View-ViewModel

7. Βιβλιογραφία

Ψηφιακές πηγές:

- A. Wilson, F. Wedyan and S. Omari. *"An Empirical Evaluation and Comparison of the Impact of MVVM and MVC GUI Driven Application Architectures on Maintainability and Testability,"* 2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA), San Antonio, TX, USA, 2022, pp. 101-108. https://ieeexplore.ieee.org/document/9923083
- Android Developers. (2024a, September 11). *App Architecture: Data Layer – DataStore*. https://developer.android.com/topic/libraries/architecture/datastore
- Android Developers. (2024b, April 4). *DownloadManager*. https://developer.android.com/reference/android/app/DownloadManager
- Android Developers. (n.d.). *Jetpack Compose UI App Development Toolkit*. https://developer.android.com/compose
- Android Developers. (2024c, October 3). *Lifecycle of composables*. https://developer.android.com/develop/ui/compose/lifecycle
- Android Developers. (2024d, October 3). *Navigation with Compose*. https://developer.android.com/develop/ui/compose/navigation
- Android Developers. (2024e, October 3). *State and Jetpack Compose*. https://developer.android.com/develop/ui/compose/state
- Android Developers. (2024f, October 3). *Using Views in Compose*. https://developer.android.com/develop/ui/compose/migrate/interoperability-apis/views-in-compose
- Android Developers. (2024g, July 18). *WebView*. https://developer.android.com/reference/android/webkit/WebView
- Atlassian. (n.d.). *Jira | Issue & Project Tracking Software*. https://www.atlassian.com/software/jira
- Block, Inc. (n.d.). *Overview*. https://square.github.io/okhttp
- Docker Inc. (n.d.). *Why Docker*. https://www.docker.com/why-docker
- Gabriel Menezes, Bruno Cafeo, Andre Hora. *How are framework code samples maintained and used by developers? The case of Android and Spring Boot*. https://www.sciencedirect.com/science/article/abs/pii/S0164121221002417
- Hibernate. (n.d.-a). *The Bean Validation reference implementation.* https://hibernate.org/validator
- Hibernate. (n.d.-b). *Your relational data. Objectively*. https://hibernate.org/orm
- Java JWT. (n.d.). *JSON Web Token for Java and Android*.

https://github.com/jwtk/jjwt

- Koceski, S. (2016, January 25). *Usage of Kanban methodology at software development teams*.
  https://eprints.ugd.edu.mk/14949
- Luca Ardito, Riccardo Coppola, Giovanni Malnati, Marco Torchiano. *Effectiveness of Kotlin vs. Java in android app development tasks.*
  https://www.sciencedirect.com/science/article/abs/pii/S0950584920301439
- M. Martinez and B. Gois Mateus, *"Why Did Developers Migrate Android Applications From Java to Kotlin?,"* in *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4521-4534, 1 Nov. 2022.
  https://ieeexplore.ieee.org/document/9576593
- PostgreSQL Global Development Group. (n.d.). *About*.
  https://www.postgresql.org/about
- Square, Inc. (n.d.-a). *Gson Converter*.
  https://github.com/square/retrofit/blob/trunk/retrofit-converters/gson/README.md
- Square, Inc. (n.d.-b). *Retrofit*.
  https://square.github.io/retrofit
- VMware Tanzu. (n.d.-a). *Aspect Oriented Programming with Spring*.
  https://docs.spring.io/spring-framework/reference/core/aop.html
- VMware Tanzu (n.d.-b). *Sending Email*.
  https://docs.spring.io/spring-framework/reference/integration/email.html
- VMware Tanzu. (n.d.-c). *Spring Boot*.
  https://spring.io/projects/spring-boot
- VMware Tanzu. (n.d.-d). *Spring Data*.
  https://spring.io/projects/spring-data
- VMware Tanzu. (n.d.-e). *Spring Data JPA*.
  https://spring.io/projects/spring-data-jpa
- VMware Tanzu. (n.d.-f). *Spring Framework*.
  https://spring.io/projects/spring-framework
- VMware Tanzu. (n.d.-g). *Spring Security*.
  https://spring.io/projects/spring-security

8. Παραρτήματα

8.1 Παράρτημα Α

```java
public static String generateUsername(String firstName, String lastName, LocalDate dateOfBirth)
{
    /*
    - This code snippet demonstrates the basic principle of username generation using the client's
    first name, last name and year of birth. Additional logic would be required for real-world use,
    including handling username conflicts.
    - username is generated by combining the firstName, LastName and the last 2 digits of the
    client's year of birth.
    */
    String fn = firstName.toLowerCase();
    String ln = lastName.substring(0, 1).toUpperCase() + lastName.substring( beginIndex: 1).toLowerCase();
    String dob = String.valueOf(dateOfBirth.getYear()).substring( beginIndex: 2);
    return fn + ln + dob;
}
```

DietitianHub-Backend/src/main/java/gr/unipi/thesis/dimstyl/utilities/RegistrationUtil.java