ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ

ΣΧΟΛΉ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΈΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση Διαδικτυακής Εφαρμογής για τη δήλωση ΠΟΘΕΝ ΕΣΧΕΣ με χρήση του framework Jhipster, με μηχανισμό αυθεντικοποίησης Oauth2.0 μέσω taxisnet και ενσωμάτωση με έναν AI Agent

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βουτσελάς Βασίλειος

Επιβλέπων : Παναγιώτης Τσανάκας

Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΔΙΑΤΑΞΕΩΝ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΕΩΝ

Υλοποίηση Διαδικτυακής Εφαρμογής για τη δήλωση ΠΟΘΕΝ ΕΣΧΕΣ με χρήση του framework Jhipster, με μηχανισμό αυθεντικοποίησης Oauth2.0 μέσω taxisnet και ενσωμάτωση με έναν AI Agent

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βουτσελάς Βασίλειος

Επιβλέπων : Παναγιώτης Τσανάκας

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Φεβρουαρίου 2025.

.....	.....	.....
Παναγιώτης Τσανάκας	Δημήτριος Σούντρης	Παναγιώτης Φράγκος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025

.....

Βουτσελάς Βασίλειος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η παρούσα διπλωματική εργασία πραγματεύεται τον σχεδιασμό και την υλοποίηση μιας διαδικτυακής εφαρμογής για την υποβολή και διαχείριση δηλώσεων Πόθεν Έσχες, με στόχο την ενίσχυση της διαφάνειας, της λειτουργικότητας και της διαλειτουργικότητας στον δημόσιο τομέα. Η εφαρμογή βασίστηκε σε σύγχρονες τεχνολογίες ανοικτού λογισμικού, όπως Angular για το frontend και JHipster με Spring Boot για το backend, αξιοποιώντας τα πλεονεκτήματα της αρθρωτής αρχιτεκτονικής και της αυτόματης παραγωγής κώδικα. Η αυθεντικοποίηση των χρηστών υποστηρίζεται μέσω δύο εναλλακτικών μηχανισμών: είτε με OAuth2 μέσω του TaxisNet, είτε με OpenID Connect μέσω Keycloak, επιτρέποντας την προσαρμογή της πλατφόρμας σε πολλαπλά περιβάλλοντα χρήσης.

Ιδιαίτερη καινοτομία αποτελεί η ενσωμάτωση ενός AI-powered chatbot, το οποίο αναπτύχθηκε με χρήση του Krikri και του Ollama, για την αυτόματη επεξεργασία δεδομένων καταγραφής τροποποιήσεων (submission audits) και την παροχή φυσικής αλληλεπίδρασης με τον χρήστη. Η πλατφόρμα υποστηρίζει πλήρως την οργάνωση, προβολή και τροποποίηση δηλώσεων από χρήστες με διακριτούς ρόλους (user/admin), προσφέροντας παράλληλα ασφαλή πρόσβαση, audit trail και έξυπνες δυνατότητες ανάλυσης. Μέσω της υιοθέτησης τεχνολογιών όπως RESTful APIs, JWT authentication, και LLM-based διαλόγου, επιτυγχάνεται μια πλήρως επεκτάσιμη και ευέλικτη λύση για τη διαχείριση θεσμικών δηλώσεων στο πλαίσιο της σύγχρονης ψηφιακής διακυβέρνησης.

Λέξεις-Κλειδιά

Πόθεν Έσχες, Τεχνητή Νοημοσύνη, JHipster, OAuth2, Keycloak, Angular, Chatbot, Krikri, Διαλειτουργικότητα, Δημόσια Διοίκηση

Abstract

This thesis focuses on the design and implementation of a modern web-based application for the submission and management of “Pothen Esches” (declarations of assets and financial interests), aiming to enhance transparency, operational efficiency, and interoperability across the public sector. The application is built using state-of-the-art open-source technologies, including Angular for the frontend and JHipster with Spring Boot for the backend, taking advantage of a modular architecture and automated code generation capabilities. User authentication is supported via two secure and configurable mechanisms: OAuth2 integration with TaxisNet and OpenID Connect through Keycloak, allowing the system to adapt to various deployment scenarios.

A core innovation of this project is the integration of an AI-powered chatbot, developed using Krikri and Ollama, which processes audit data (submission logs) and allows users to interact in natural language. The platform provides full support for submission creation, modification, and audit by users with distinct roles (user/admin), while ensuring secure access, role-based authorization, and a comprehensive audit trail. By incorporating technologies such as RESTful APIs, JWT authentication, and LLM-based conversational interfaces, the system achieves a scalable and flexible solution for managing official asset declarations in the context of modern digital governance.

Keywords

Pothen Esches, Artificial Intelligence, JHipster, OAuth2, Keycloak, Angular, Chatbot, Krikri, Interoperability, Public Sector

Πίνακας περιεχομένων

ΚΕΦΑΛΑΙΟ 1.ΕΙΣΑΓΩΓΗ

1.1 Εξάπλωση της τεχνητής νοημοσύνης

Είναι γεγονός πως τα τελευταία χρόνια έχει παρατηρηθεί μια αλματώδης πρόοδος στον τομέα της τεχνητής νοημοσύνης. Από την ακαδημαϊκή έρευνα έχουμε περάσει στην ενσωμάτωση τέτοιων συστημάτων σε καθημερινές εφαρμογές και σε εταιρείες κολοσσούς που εντάσσουν στα συστήματά τους κάποια υπηρεσία σχετική με τεχνητή νοημοσύνη. Είναι σπάνιο πλέον κάποιος πολίτης του δυτικού κόσμου να μην έχει στην κατοχή του κάποια συσκευή που χρησιμοποιεί AI εφαρμογές είτε να μην γνωρίζει ή να έχει διαδράσει ο ίδιος με κάποιο τέτοιο σύστημα.

Η Τεχνητή Νοημοσύνη (Artificial Intelligence - AI) δεν αποτελεί πλέον μελλοντική υπόσχεση αλλά παρούσα πραγματικότητα: σύμφωνα με τον δείκτη AI Index 2024 του Stanford University, οι επενδύσεις σε εφαρμογές AI αυξήθηκαν κατά 41% σε σύγκριση με το προηγούμενο έτος, ενώ ο αριθμός των οργανισμών που δηλώνουν ενεργή χρήση εργαλείων AI στον πυρήνα των πληροφοριακών τους ροών έχει υπερδιπλασιαστεί [1]. Επιπλέον, η έκθεση της McKinsey (2023) σημειώνει ότι το 55% των εταιρειών παγκοσμίως χρησιμοποιούν πλέον τεχνητή νοημοσύνη τουλάχιστον σε μία λειτουργία τους, με κυριότερες εφαρμογές τον αυτοματισμό, την εξυπηρέτηση πελατών, και τη διαχείριση γνώσης [2].

Ιδιαίτερο ενδιαφέρον παρουσιάζει η εισαγωγή της τεχνητής νοημοσύνης στον τομέα της δημόσιας διοίκησης. Σύμφωνα με τον ΟΟΣΑ, κυβερνήσεις σε όλο τον κόσμο επενδύουν πλέον σε "AI for Public Good" στρατηγικές, αξιοποιώντας γλωσσικά μοντέλα για τη βελτίωση της πρόσβασης στη νομοθεσία, την οργάνωση εγγράφων και τη διαδραστική εξυπηρέτηση πολιτών [3]. Σε αυτό το πλαίσιο εντάσσεται και η υλοποίηση που παρουσιάζεται στην παρούσα διπλωματική: η δημιουργία ενός πληροφοριακού συστήματος με δυνατότητα φυσικής γλώσσας, το οποίο ενσωματώνει ένα μεγάλο γλωσσικό μοντέλο (Large Language Model - LLM) για την ανάλυση και εποπτεία των δηλώσεων Πόθεν Έσχες. Ένα τέτοιο παράδειγμα αποδεικνύει στην πράξη τη μετάβαση από τα στατικά, αυστηρά πληροφοριακά συστήματα του παρελθόντος σε έξυπνες, "ενεργές" πλατφόρμες, ικανές να ενισχύουν την αλληλεπίδραση του χρήστη, να μαθαίνουν από τα δεδομένα και να προσφέρουν πολύ πιο εύχρηστο, ευέλικτο και διαφανές περιβάλλον διάδρασης.

Η μηχανική μάθηση (Machine Learning), τα νευρωνικά δίκτυα (Neural Networks) , η επεξεργασία φωνής και φυσικής γλώσσας (Natural Language Processing) και φυσικά τα μεγάλα γλωσσικά μοντέλα (LLMs) είναι οι σημαντικότερες εφαρμογές της τεχνητής νοημοσύνης και έχουν πλέον εγκαθιδριθεί σαν θεμέλιος λίθος της σύγχρονης πληροφορικής. Πλέον, η ενσωμάτωση της τεχνητής νοημοσύνης σε πλατφόρμες χρήσης δεν είναι κάτι μεμονωμένο ή εξειδικευμένο. Είναι μια τάση που καθιερώνεται σε παγκόσμια κλίμακα, δημιουργώντας νέες προσδοκίες τόσο για τους τελικούς χρήστες όσο και για τους ίδιους τους προγραμματιστές. Οι χρήστες αρχίζουν να περιμένουν πιο φυσικές, προσαρμοστικές και “έξυπνες” εμπειρίες από τα συστήματα με τα οποία αλληλεπιδρούν. Η δυνατότητα να διατυπώνουν ερωτήματα σε φυσική γλώσσα, να λαμβάνουν δυναμικά αποτελέσματα ή να επικοινωνούν με την

πλατφόρμα όπως θα μιλούσαν με έναν άνθρωπο, αλλάζει τα δεδομένα στον σχεδιασμό και την ανάπτυξη λογισμικού.

Παράλληλα, από την πλευρά της υλοποίησης, οι προγραμματιστές και οι ομάδες ανάπτυξης βρίσκονται πλέον μπροστά σε νέα εργαλεία που κάνουν την ενσωμάτωση αυτών των τεχνολογιών πιο απλή, γρήγορη και αποδοτική. Η τεχνητή νοημοσύνη, από εργαλείο των "λίγων", μετατρέπεται σε ένα ευρέως διαθέσιμο σύστημα υποστήριξης και ενίσχυσης των δυνατοτήτων των εφαρμογών.

Αυτή η νέα πραγματικότητα, στην οποία η τεχνητή νοημοσύνη είναι παρούσα σχεδόν παντού, δημιουργεί όχι μόνο τεχνολογικές αλλά και αρχιτεκτονικές και σχεδιαστικές προκλήσεις. Η παρούσα διπλωματική εργασία έρχεται να διερευνήσει ακριβώς αυτό το φαινόμενο: πώς δηλαδή οι σύγχρονες πλατφόρμες μπορούν να εκμεταλλευτούν την τεχνητή νοημοσύνη ώστε να μεταβούν από παραδοσιακές, στατικές λύσεις σε έξυπνα, προσαρμοστικά περιβάλλοντα που εξυπηρετούν καλύτερα τόσο τον χρήστη όσο και τον προγραμματιστή.

1.2 Ευκαιρία να γίνουν λειτουργικότερες οι εφαρμογές που ακολουθούσαν το παλαιό μοντέλο χρήσης

Η εξάπλωση της Τεχνητής Νοημοσύνης και των σύγχρονων εργαλείων ανάπτυξης δεν φέρνει μόνο νέες δυνατότητες για τη δημιουργία καινοτόμων εφαρμογών, αλλά ταυτόχρονα προσφέρει μια μοναδική ευκαιρία: να επανασχεδιαστούν και να γίνουν ουσιαστικά λειτουργικότερες οι υπάρχουσες εφαρμογές που βασίζονται σε παρωχημένα μοντέλα χρήσης και υλοποίησης. Πολλές από αυτές τις εφαρμογές – γνωστές και ως legacy systems – έχουν σχεδιαστεί με βάση αρχές που ίσχυαν σε παλαιότερες εποχές, όταν οι ανάγκες ήταν διαφορετικές, η τεχνολογική υποδομή πιο περιορισμένη και η αλληλεπίδραση με τον χρήστη σαφώς πιο στατική.

Τα προβλήματα που σχετίζονται με τα legacy συστήματα είναι πολυδιάστατα. Σε επίπεδο χρήστη, αυτά τα συστήματα είναι συνήθως περιορισμένα για πραγματική αλληλεπίδραση, καθώς περιορίζονται σε αυστηρά προκαθορισμένες φόρμες, κουμπιά και μηχανισμούς αναζήτησης. Η εμπειρία είναι στατική, με την εφαρμογή να «επιτρέπει» συγκεκριμένες ενέργειες αντί να «κατανοεί» τις ανάγκες του χρήστη. Σε τέτοιες υλοποιήσεις, ο χρήστης οφείλει να προσαρμοστεί στη δομή του συστήματος και όχι το αντίστροφο. Δεν είναι ότι απαιτείται τεχνική γνώση, αλλά απουσιάζει η δυνατότητα να εκφράσει ελεύθερα και με ευελιξία αυτό που χρειάζεται, περιορίζοντάς τον σε μια εμπειρία που βασίζεται σε κουμπιά, φίλτρα και σενάρια χρήσης που δεν καλύπτουν απαραίτητα τις ιδιαίτερες περιπτώσεις του. Αντίθετα, με την ενσωμάτωση τεχνητής νοημοσύνης και ιδιαίτερα εργαλείων φυσικής γλώσσας, δίνεται η δυνατότητα στον χρήστη να εκφράζεται όπως μιλάει – να διατυπώνει ερωτήματα, εντολές ή ανάγκες σε απλή, καθημερινή γλώσσα. Το σύστημα μπορεί να μεταφράζει την πρόθεση αυτή σε ενέργεια (query, αναζήτηση, δράση), δημιουργώντας έτσι ένα πιο ανθρώπινο, φυσικό και προσαρμοζόμενο περιβάλλον.

Έτσι, οι χρήστες δεν χρειάζεται να μάθουν πώς να χειρίζονται το σύστημα, αλλά αντίθετα, το σύστημα μαθαίνει να καταλαβαίνει τους χρήστες του.

Από την πλευρά των προγραμματιστών, τα legacy συστήματα παρουσιάζουν διαφορετικής φύσης περιορισμούς. Οι μέθοδοι διασύνδεσης και η δομή είναι συνήθως σχεδιασμένα με αυστηρή λογική, περιορισμένα σε συγκεκριμένα πρότυπα ή workflows, και δύσκολα προσαρμόσιμα σε νέα σενάρια χρήσης. Η προσθήκη νέων λειτουργιών απαιτεί εκτεταμένο refactoring ή προσθήκες που διατρέχουν τον κίνδυνο να προκαλέσουν ανεπιθύμητα αποτελέσματα λόγω της δομικής πολυπλοκότητας. Παράλληλα, η ίδια η αρχιτεκτονική αυτών των συστημάτων – συνήθως μονολιθική – δυσκολεύει την παράλληλη ανάπτυξη, την ευέλικτη δοκιμή ή την ασφαλή απομόνωση νέων modules. Αντίθετα, με τη χρήση εργαλείων Τεχνητής Νοημοσύνης, δίνεται στους προγραμματιστές μια νέα διάσταση υλοποίησης: οι λειτουργίες μπορούν να σχεδιαστούν με πιο ελεύθερο τρόπο, βασισμένες στην πρόθεση του χρήστη και το περιεχόμενο των ερωτημάτων του, αντί για συγκεκριμένα μονοπάτια περιπτώσεων χρήσης.

Η AI έρχεται να ξεκλειδώσει αυτά τα αδιέξοδα. Με την ενσωμάτωσή της, τα ίδια τα εργαλεία μπορούν να γίνουν πιο κατανοητά, πιο ανθρώπινα και πιο αποτελεσματικά, χωρίς απαραίτητα να απαιτείται ριζική ανακατασκευή της πλατφόρμας από το μηδέν. Δίνεται η δυνατότητα για επικοινωνία σε φυσική γλώσσα, για αυτοματοποίηση ενεργειών, για παροχή βοήθειας σε πραγματικό χρόνο και για έξυπνη αναζήτηση δεδομένων – πράγματα που μεταμορφώνουν τον τρόπο με τον οποίο αντιλαμβανόμαστε την εργασία μέσα σε ένα ψηφιακό σύστημα. Ένα chatbot, για παράδειγμα, μπορεί να απαντά σε ευρύ φάσμα ερωτημάτων χωρίς ο προγραμματιστής να έχει προδιαγράψει κάθε πιθανό σενάριο. Εργαλεία όπως τα LLMs επιτρέπουν πιο modular και low-code επεκτάσεις, μειώνοντας σημαντικά τον χρόνο ανάπτυξης και την ανάγκη συνεχούς τροποποίησης του πυρήνα του συστήματος.

Ένα ενδεικτικό παράδειγμα, που προέκυψε και μέσα από την παρούσα διπλωματική εργασία, αφορά την παραδοσιακή διαδικασία ανάκτησης πληροφοριών από βάσεις δεδομένων. Σε ένα υπάρχον σύστημα, ο προγραμματιστής έπρεπε να γνωρίζει τη δομή της βάσης, τα ονόματα των πινάκων και τις κατάλληλες SQL εντολές για να εξάγει τα στοιχεία που τον ενδιέφεραν. Η υλοποίηση ενός LLM-powered chatbot, το οποίο μπορούσε να κατανοήσει ερωτήματα σε φυσική γλώσσα (π.χ. «Δείξε μου όλες τις καταχωρήσεις της τελευταίας εβδομάδας») και να μεταφράσει το ερώτημα αυτό αυτόματα σε σωστή SQL εντολή, άλλαξε δραματικά την εμπειρία. Οι χρήστες απέκτησαν πρόσβαση στη γνώση που κρυβόταν στα δεδομένα χωρίς να περιορίζονται σε συγκεκριμένα use cases, ενώ οι προγραμματιστές απελευθερώθηκαν από μικρές, επαναλαμβανόμενες εργασίες, ώστε να επικεντρωθούν στην περαιτέρω ανάπτυξη της πλατφόρμας.

Η αναβάθμιση legacy συστημάτων με τη χρήση εργαλείων τεχνητής νοημοσύνης δεν είναι απλώς μια επιλογή βελτίωσης. Είναι μια στρατηγική κίνηση για να καταστούν οι πλατφόρμες βιώσιμες, ευέλικτες και φιλικές προς τον τελικό χρήστη, σε έναν κόσμο όπου η τεχνολογία εξελίσσεται ταχύτερα από ποτέ.

1.3 Οργάνωση Τόμου

Η παρούσα διπλωματική εργασία είναι δομημένη σε έξι κύρια κεφάλαια, το καθένα από τα οποία επιτελεί διακριτό ρόλο στην ανάπτυξη και παρουσίαση του θέματος, της υλοποίησης και των συμπερασμάτων που προέκυψαν. Η σειρά και η θεματολογία των κεφαλαίων έχουν σχεδιαστεί ώστε να ακολουθείται μία λογική ροή από τη γενική παρουσίαση της ιδέας προς την ανάλυση της υλοποίησης και την αποτίμηση της προστιθέμενης αξίας του έργου.

Στο **Κεφάλαιο 1**, γίνεται εισαγωγή στο θέμα της εργασίας, εστιάζοντας στην εξάπλωση της τεχνητής νοημοσύνης και τη δυνατότητα αξιοποίησής της για τη δημιουργία λειτουργικότερων πληροφοριακών εφαρμογών. Επιπλέον, αναδεικνύεται το πρόβλημα των legacy συστημάτων και η ανάγκη επαναπροσδιορισμού τους με τη χρήση σύγχρονων τεχνολογιών.

Το **Κεφάλαιο 2** περιγράφει συνοπτικά την ιδέα της εφαρμογής που αναπτύχθηκε στο πλαίσιο της διπλωματικής. Αναλύεται η αρχιτεκτονική της πλατφόρμας, η οποία βασίζεται στον συνδυασμό Angular και JHipster, ενώ περιγράφεται και η πλατφόρμα Πόθεν Έσχες, ώστε να αναδειχθεί η στοχευμένη προσπάθεια για μία πιο ευέλικτη και διαδραστική εναλλακτική λύση.

Στο **Κεφάλαιο 3**, γίνεται εκτενής τεχνική ανάλυση των εργαλείων και τεχνολογιών που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Παρουσιάζονται οι δυνατότητες του JHipster, εναλλακτικές επιλογές, οι τεχνικοί περιορισμοί και οι επεκτάσεις που πραγματοποιήθηκαν, ενώ γίνεται και αναλυτική αναφορά στις τεχνολογίες LLM (Large Language Models) που αξιοποιήθηκαν για τη δημιουργία του chatbot, με παράθεση αποτελεσμάτων benchmarking.

Το **Κεφάλαιο 4** είναι αφιερωμένο στην ανάλυση των διαδικασιών της εφαρμογής μέσω διαγραμμάτων ροής. Παρουσιάζονται κρίσιμες λειτουργίες όπως η σύνδεση χρηστών, η υποβολή και διαχείριση δηλώσεων, η αλληλεπίδραση του admin με το σύστημα και η λειτουργία του AI chatbot. Επιπλέον, αναδεικνύεται η καινοτομία της δυνατότητας ερωτημάτων σε φυσική γλώσσα στη θέση παραδοσιακών SQL queries.

Στο **Κεφάλαιο 5**, παρουσιάζονται αντιπροσωπευτικές περιπτώσεις χρήσης (use cases) για τους δύο βασικούς ρόλους της εφαρμογής, τον χρήστη και τον διαχειριστή. Τα σενάρια αυτά αποτυπώνουν τις λειτουργικές απαιτήσεις και τη συμπεριφορά του συστήματος σε ρεαλιστικές συνθήκες, αναδεικνύοντας την ευχρηστία και την αποτελεσματικότητα της πλατφόρμας.

Τέλος, στο **Κεφάλαιο 6**, διατυπώνονται προτάσεις για μελλοντική εργασία και επεκτάσεις της εφαρμογής. Προτείνονται νέες λειτουργίες όπως σύστημα αξιολόγησης κινδύνου, ψηφιακή υπογραφή δηλώσεων, role-based notifications, εξαγωγή reports και υποστήριξη για mobile ή άλλες εφαρμογές. Γίνεται επίσης αναφορά στην προοπτική επαναχρησιμοποίησης της πλατφόρμας σε άλλα θεσμικά ή επαγγελματικά περιβάλλοντα.

Αυτή η δομή επιτρέπει την σταδιακή εμβάθυνση από το θεωρητικό και τεχνολογικό υπόβαθρο έως την πρακτική αξιολόγηση και τις προοπτικές του έργου, προσφέροντας μία ολοκληρωμένη και συνεκτική παρουσίαση της διπλωματικής εργασίας.

ΚΕΦΑΛΑΙΟ 2. ΠΑΡΟΥΣΙΑΣΗ ΘΕΜΑΤΟΣ

2.1 Συνοπτική Παρουσίαση της ιδέας

Η παρούσα διπλωματική εργασία εστιάζει στον σχεδιασμό και την υλοποίηση μιας ολοκληρωμένης και τεχνολογικά σύγχρονης διαδικτυακής πλατφόρμας για τη διαχείριση και υποβολή δηλώσεων Πόθεν Έσχες, φιλοδοξώντας να επανακαθορίσει τον τρόπο με τον οποίο οργανώνονται και εκτελούνται θεσμικές υποχρεώσεις υψηλής διαφάνειας και λογοδοσίας. Η πλατφόρμα δεν αποσκοπεί απλώς στην ψηφιακή μεταφορά μιας υφιστάμενης γραφειοκρατικής διαδικασίας, αλλά προτείνει ένα νέο μοντέλο ψηφιακής διακυβέρνησης, βασισμένο σε αρχές ευελιξίας, ασφάλειας και – κυρίως – διαλειτουργικότητας.

Στο πλαίσιο αυτό, η εφαρμογή υλοποιήθηκε με τεχνολογικές επιλογές που διευκολύνουν την αλληλεπίδραση μεταξύ συστημάτων και την αποδοτική διασύνδεση με εξωτερικές δημόσιες υπηρεσίες. Μέσω της αξιοποίησης σύγχρονων προτύπων όπως RESTful APIs και open authentication protocols (OAuth2, OpenID Connect), δημιουργείται μια τεχνολογική υποδομή ικανή να υποστηρίξει δυναμική ανταλλαγή πληροφορίας ανάμεσα σε διακριτές υπηρεσίες, συστήματα και πλατφόρμες, χωρίς την ανάγκη για πολύπλοκες χειροκίνητες παρεμβάσεις ή ad-hoc διασυνδέσεις. Η αρχιτεκτονική της πλατφόρμας επιτρέπει, για παράδειγμα, την ταυτόχρονη συνεργασία με παρόχους αυθεντικοποίησης όπως το TaxisNet ή το Keycloak, αποδεικνύοντας έμπρακτα την προσαρμοστικότητά της σε περιβάλλοντα με διαφορετικά πρωτόκολλα ή πολιτικές πρόσβασης.

Σε λειτουργικό επίπεδο, η δυνατότητα αυτή καθιστά το σύστημα ικανό να λειτουργεί ως ψηφιακός κόμβος ενοποίησης θεσμικών ροών, επιτρέποντας στους αρμόδιους φορείς να ανταλλάσσουν δεδομένα σε πραγματικό χρόνο και να αποκτούν εποπτεία επί κρίσιμων διαδικασιών χωρίς την ανάγκη για κατακερματισμένες εφαρμογές ή επαναλαμβανόμενες ενέργειες από τον χρήστη. Με αυτόν τον τρόπο, η διαδικασία της δήλωσης Πόθεν Έσχες δεν αποτελεί πλέον ένα απομονωμένο πληροφοριακό κόμβο, αλλά εντάσσεται οργανικά σε ένα σύγχρονο οικοσύστημα συνεργαζόμενων υπηρεσιών που προάγουν τη διαφάνεια, την προσβασιμότητα και την ενιαία, διασυνδεδεμένη ψηφιακή εμπειρία.

Η τεχνική υλοποίηση βασίστηκε στο συνδυασμό σύγχρονων τεχνολογιών: το frontend αναπτύχθηκε με τη χρήση Angular, προσφέροντας ένα διαδραστικό και responsive περιβάλλον χρήστη, ενώ για το backend επιλέχθηκε το framework JHipster, το οποίο επιτρέπει τη γρήγορη δημιουργία πλήρους λειτουργικού server με ενσωματωμένη υποστήριξη βάσεων δεδομένων, RESTful υπηρεσιών και role-based πρόσβασης. Η ασφάλεια της πλατφόρμας διασφαλίστηκε μέσω ενσωμάτωσης του OAuth 2.0 πρωτοκόλλου αυθεντικοποίησης και ταυτοποίησης, με χρήση του επίσημου μηχανισμού σύνδεσης του TaxisNet (gsis.gr). Έτσι, επιτυγχάνεται ελεγχόμενη πρόσβαση με βάση κρατικά στοιχεία, διασφαλίζοντας την εγκυρότητα και την ακεραιότητα των δεδομένων των χρηστών.

Ιδιαίτερη έμφαση δόθηκε στην υποστήριξη πολλαπλών ρόλων εντός της πλατφόρμας. Πέρα από τον απλό χρήστη που μπορεί να υποβάλει ή να τροποποιεί δηλώσεις,

προβλέφθηκε ρόλος διαχειριστή (admin), ο οποίος έχει τη δυνατότητα επισκόπησης όλων των δηλώσεων, τροποποίησης μεταδεδομένων (π.χ. βαθμός, ιδιότητα, επιτροπή) και παρακολούθησης του ιστορικού καταχωρήσεων από τους χρήστες.

Καινοτομία της πλατφόρμας αποτελεί η ενσωμάτωση ενός AI chatbot, βασισμένου στο ελληνικό LLM Krikri, που παρέχει λειτουργικότητα φυσικής γλώσσας για την αναζήτηση και επεξεργασία πληροφοριών από το ιστορικό δηλώσεων. Το chatbot έχει τη δυνατότητα να δέχεται ερωτήματα σε φυσική γλώσσα, όπως "ποιες δηλώσεις άλλαξαν την τελευταία εβδομάδα;" ή "τι ενέργειες έγιναν από τον χρήστη X", και να επιστρέφει οργανωμένες απαντήσεις βασισμένες σε πραγματικά δεδομένα του συστήματος. Το υποσύστημα αυτό λειτουργεί ως ενδιάμεσος μηχανισμός κατανόησης προθέσεων (intent parsing) και εκτέλεσης δυναμικών queries, προσφέροντας μια εμπειρία διαχείρισης που προσεγγίζει τις αρχές του conversational interface και του no-code interaction. Με αυτόν τον τρόπο, ενισχύεται σημαντικά η λειτουργικότητα της πλατφόρμας, ιδιαίτερα από την πλευρά του διαχειριστή, ενώ παράλληλα μειώνεται η ανάγκη τεχνικής εξοικείωσης από την πλευρά των χρηστών.

Συνολικά, η εργασία αποσκοπεί όχι μόνο στην τεχνική υλοποίηση μιας πλατφόρμας δηλώσεων Πόθεν Έσχες, αλλά και στη δημιουργία ενός σύγχρονου περιβάλλοντος χρήσης που αξιοποιεί την τεχνητή νοημοσύνη για την υποστήριξη της διαφάνειας, της προσβασιμότητας και της εμπιστοσύνης προς τον ψηφιακό δημόσιο χώρο.

2.2 Ανάλυση της Αρχιτεκτονικής

Η αρχιτεκτονική της πλατφόρμας που υλοποιήθηκε στο πλαίσιο αυτής της διπλωματικής εργασίας ακολουθεί μια καθαρή, πολυεπίπεδη και αποκεντρωμένη λογική, βασισμένη σε μοντέρνες τεχνικές διαχωρισμού ρόλων (separation of concerns), επικοινωνίας μέσω REST APIs και αξιοποίησης εξωτερικών υποσυστημάτων για προηγμένες λειτουργίες. Ο σχεδιασμός έγινε με γνώμονα την ευκολία επέκτασης, την ασφάλεια και τη δυνατότητα διασύνδεσης με τρίτα εργαλεία.

Η εφαρμογή διαχωρίζεται σε τρία βασικά επίπεδα: το frontend (Angular), το backend (JHipster + Spring Boot), και το εξωτερικό σύστημα του chatbot, το οποίο λειτουργεί αυτόνομα. Το frontend αποτελείται από μια μοντέρνα Angular εφαρμογή που υλοποιεί πλήρως τη λογική του single-page application (SPA), οργανωμένη σε modules, components και services. Οι χρήστες αλληλεπιδρούν με την εφαρμογή μέσω μιας καθαρής και λειτουργικής διεπαφής, με τις λειτουργίες να είναι πλήρως ασύγχρονες και να πραγματοποιούνται μέσω HTTP requests προς τα RESTful endpoints του backend. Η επικοινωνία επιτυγχάνεται μέσω του Angular HttpClient και περιλαμβάνει πλήρη υποστήριξη για authentication headers, guards και error handling.

Το backend αναπτύχθηκε με τη χρήση του JHipster framework, το οποίο βασίζεται στο Spring Boot και παρέχει out-of-the-box λειτουργίες για user management, role-based access control, διαχείριση οντοτήτων και σύνδεση με τη βάση δεδομένων μέσω Hibernate. Το framework αυτό διασφαλίζει τη δημιουργία καθαρών REST

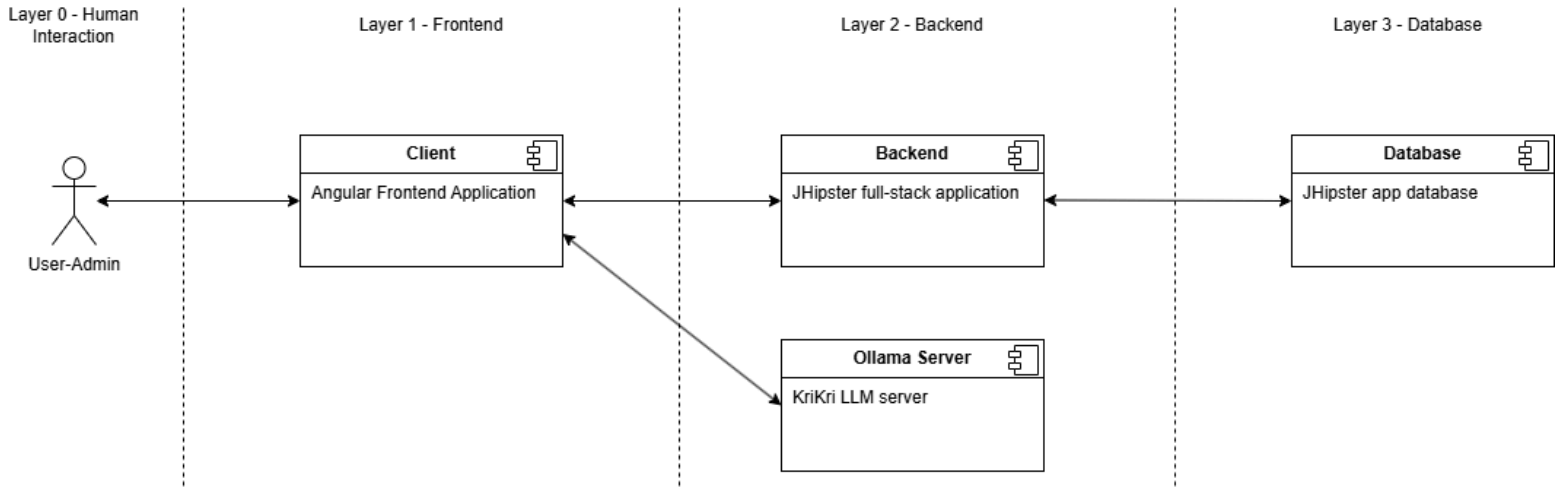
controllers για κάθε οντότητα, διευκολύνοντας την επικοινωνία με το frontend. Όλα τα δεδομένα – δηλώσεις, χρήστες, καταχωρήσεις – αποθηκεύονται σε σχεσιακή βάση δεδομένων. Η αυθεντικοποίηση των χρηστών υλοποιείται μέσω δύο οδών. Η πρώτη οδός είναι η σύνδεση με τοπικα διαπιστευτήρια της εφαρμογής με μηχανισμό αυθεντικοποίησης json web token (jwt), ενώ η δεύτερη οδός είναι με σύνδεση με την κρατική πλατφόρμα TaxisNet (gsis) με πρωτόκολλο OAuth2.0, όπου ο χρήστης ανακατευθύνεται στην επίσημη σελίδα για ταυτοποίηση, και στη συνέχεια επιστρέφει στην εφαρμογή με έγκυρο access token, το οποίο του δίνει πρόσβαση με βάση τον ρόλο του (π.χ. απλός χρήστης ή διαχειριστής).

Η πιο ιδιαίτερη και καινοτόμα συνιστώσα της αρχιτεκτονικής είναι το εξωτερικό υποσύστημα του chatbot. Πρόκειται για έναν αυτόνομο server υλοποιημένο σε Python, ο οποίος τρέχει μια Flask εφαρμογή, στην οποία έχει ενσωματωθεί το ελληνικό LLM Krikri με backend λειτουργία μέσω Ollama. Το chatbot αυτό αποτελεί ξεχωριστή υπηρεσία, που φιλοξενείται ανεξάρτητα από το κύριο backend, και επικοινωνεί αποκλειστικά με το frontend μέσω HTTP API. Η επικοινωνία αφορά δύο είδη πληροφορίας: (1) ένα στατικό αρχείο, το οποίο περιέχει το ιστορικό όλων των καταχωρήσεων (submissions), και (2) το ερώτημα του χρήστη, διατυπωμένο σε φυσική γλώσσα.

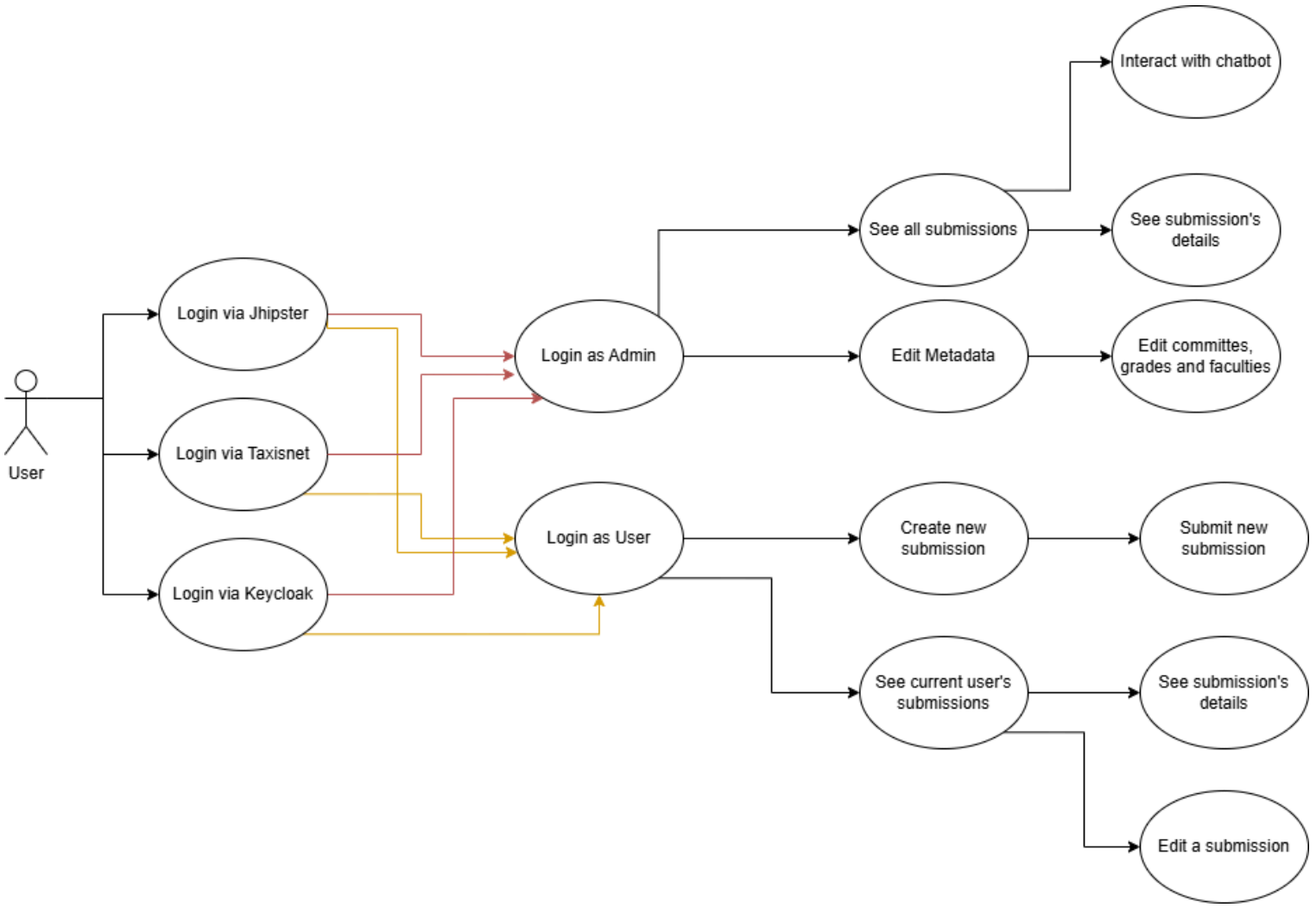
Κατά την εκτέλεση, ο χρήστης πληκτρολογεί ένα ερώτημα στο UI, το οποίο αποστέλλεται μέσω POST request στο Flask server. Εκεί, το chatbot με χρήση LLM μοντέλου αναλύει την πρόθεση του χρήστη (intent), αντιστοιχίζει το ερώτημα με τα δεδομένα του αρχείου που του έχει δοθεί και επιστρέφει οργανωμένη απάντηση, κατανοητή και χρήσιμη, χωρίς ο χρήστης να χρειάζεται τεχνικές γνώσεις. Ολόκληρη η ροή της επικοινωνίας αυτής παραμένει απομονωμένη από την κύρια βάση δεδομένων, διασφαλίζοντας έτσι ότι δεν εκτίθενται ευαίσθητα δεδομένα σε μη εξουσιοδοτημένες υπηρεσίες.

Αυτή η αρχιτεκτονική προσέγγιση —με τον καθαρό διαχωρισμό του UI (Angular), της επιχειρησιακής λογικής (JHipster/Spring) και του νοηματικού υποσυστήματος AI (Python/Krikri)— επιτρέπει όχι μόνο την εύκολη συντήρηση και μελλοντική επεκτασιμότητα, αλλά και την ασφαλή ενσωμάτωση καινοτόμων λειτουργιών χωρίς να απαιτείται τροποποίηση του βασικού συστήματος. Τα τρία υποσυστήματα επικοινωνούν μεταξύ τους μέσω http requests, διατηρώντας την ανεξαρτησία και επαναχρησιμοποίηση του κάθε μέρους. Η επιλογή αυτής της αρχιτεκτονικής αποδείχθηκε στην πράξη ευέλικτη, στιβαρή και συμβατή με τις σύγχρονες απαιτήσεις διαλειτουργικότητας και modular ανάπτυξης.

Παρακάτω δίνεται ένα διάγραμμα της αρχιτεκτονικής της εφαρμογής μας καθώς και ένα διάγραμμα των λειτουργικοτήτων της εφαρμογής.



Διάγραμμα Αρχιτεκτονικής Εφαρμογής



Διάγραμμα Λειτουργιών Εφαρμογής

2.3 Παρουσίαση

Η υφιστάμενη πλατφόρμα Πόθεν Έσχες, όπως λειτουργεί σήμερα υπό την εποπτεία της Γενικής Γραμματείας Πληροφοριακών Συστημάτων και Ψηφιακής Διακυβέρνησης, αποτελεί ένα ολοκληρωμένο πληροφοριακό σύστημα για την υποβολή των Δηλώσεων Περιουσιακής Κατάστασης (ΔΠΚ) και των Δηλώσεων Οικονομικών Συμφερόντων (ΔΟΣ). Στόχος της πλατφόρμας είναι η ενίσχυση της διαφάνειας, η απλοποίηση της διαδικασίας για τους υπόχρεους και η αποτελεσματική παρακολούθηση των δηλώσεων από τους αρμόδιους φορείς ελέγχου.

Η πρόσβαση των χρηστών πραγματοποιείται μέσω αυθεντικοποίησης με τα στοιχεία του TaxisNet, διασφαλίζοντας υψηλό επίπεδο ασφάλειας και διασύνδεσης με κρατικές βάσεις δεδομένων. Μετά τη σύνδεση, η πλατφόρμα προσφέρει αυτοματοποιημένη ανάκτηση φορολογικών στοιχείων (όπως δεδομένα από Ε1 και Ε9), πληροφορίες για ακίνητα, λογαριασμούς και επενδυτικά προϊόντα, μέσω διαλειτουργικότητας με τρίτους φορείς όπως τράπεζες και δημόσιες υπηρεσίες. Αυτή η διασύνδεση μειώνει σημαντικά το διαχειριστικό βάρος των χρηστών και περιορίζει την πιθανότητα σφαλμάτων κατά την υποβολή.

Ένα ακόμα χαρακτηριστικό της νέας μορφής της πλατφόρμας είναι η διαφοροποίηση του ρόλου των συζύγων. Σύμφωνα με τον νόμο 5026/2023, πλέον και οι σύζυγοι των υπόχρεων θεωρούνται αυτοτελώς υπόχρεοι, γεγονός που απαιτεί από αυτούς να διαχειρίζονται ανεξάρτητα τις δικές τους δηλώσεις. Παράλληλα, η πλατφόρμα ενσωματώνεται με το Εθνικό Μητρώο Επικοινωνίας (emep.gov.gr), διασφαλίζοντας την άμεση και έγκαιρη ενημέρωση των πολιτών για τις υποχρεώσεις τους.

Παρότι η υφιστάμενη πλατφόρμα επιτελεί έναν σημαντικό ρόλο στη διαδικασία υποβολής δηλώσεων Πόθεν Έσχες, η λειτουργία της εξακολουθεί να παρουσιάζει ορισμένες προκλήσεις, τόσο σε επίπεδο εμπειρίας χρήστη όσο και σε δυνατότητες προσαρμογής. Η διεπαφή παραμένει σχετικά στατική, η αλληλεπίδραση με το σύστημα είναι περιορισμένη και η αναζήτηση πληροφοριών βασίζεται σε σταθερά φίλτρα και δομές. Η ανάγκη για περαιτέρω εκσυγχρονισμό, με στόχο μια πιο διαδραστική, φιλική και «έξυπνη» προσέγγιση, αποτελεί το έναυσμα για την ανάπτυξη της εναλλακτικής λύσης που παρουσιάζεται σε αυτή τη διπλωματική εργασία.

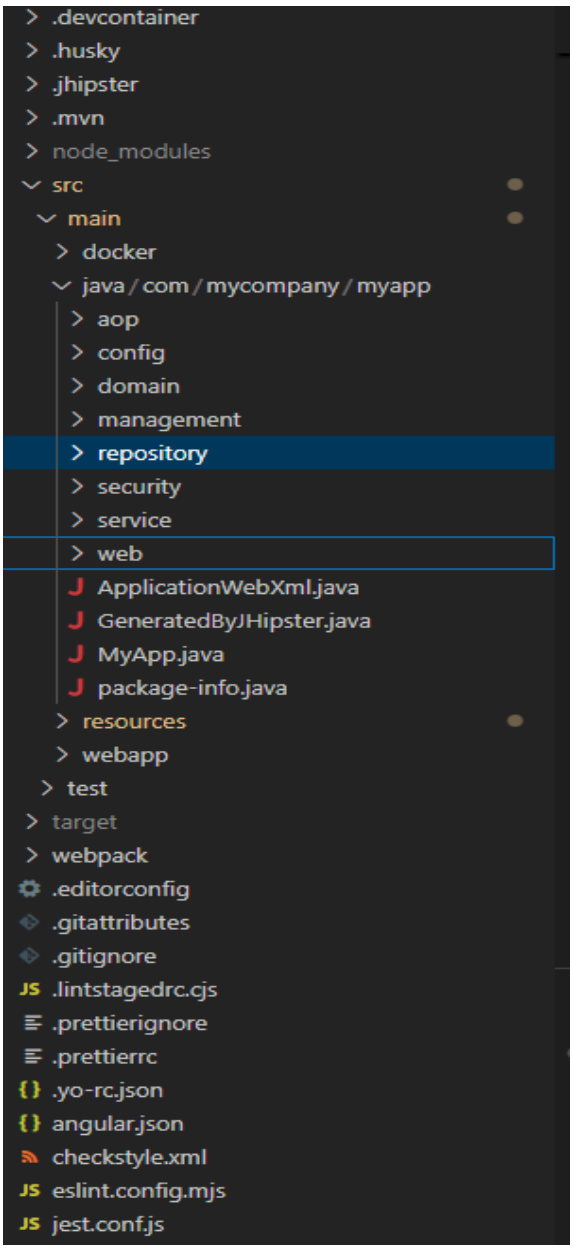
Η υλοποιημένη εφαρμογή δεν φιλοδοξεί να αντικαταστήσει το υφιστάμενο κρατικό σύστημα, αλλά να διερευνήσει πώς σύγχρονες τεχνολογίες, όπως τα μεγάλα γλωσσικά μοντέλα και οι αρχιτεκτονικές SPA, μπορούν να προσδώσουν προστιθέμενη αξία στη διαδικασία δήλωσης Πόθεν Έσχες, προσφέροντας αυξημένη ευχρηστία, διαδραστικότητα και δυνατότητα επεκτασιμότητας.

ΚΕΦΑΛΑΙΟ 3. ΑΝΑΛΥΣΗ ΠΛΑΤΦΟΡΜΑΣ ΚΑΙ LLM

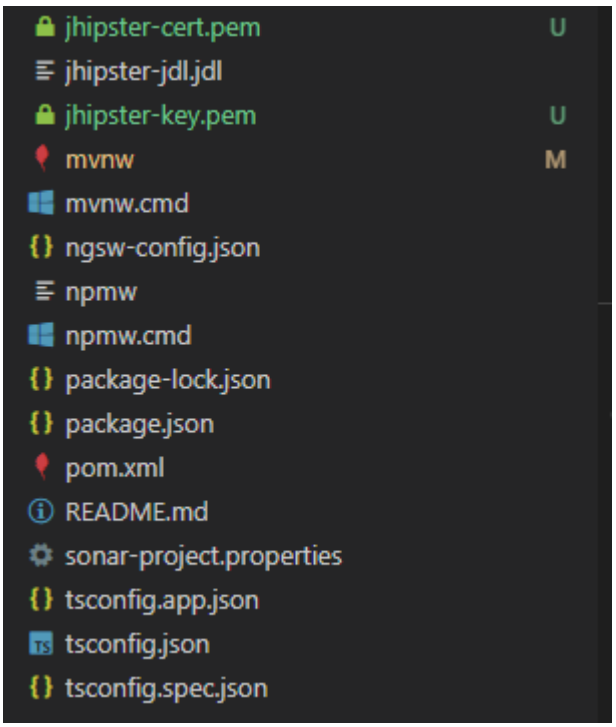
3.1. Παρουσίαση Jhipster και εναλλακτικών εργαλείων

Το JHipster αποτελεί ένα ισχυρό open-source framework σχεδιασμένο για τη δημιουργία full stack εφαρμογών με έμφαση στη γρήγορη ανάπτυξη κώδικα, την καθαρή αρχιτεκτονική και την ενσωματωμένη υποστήριξη για σύγχρονα πρότυπα ασφαλείας και DevOps. Η επιλογή του στην παρούσα διπλωματική εργασία έγινε για να γίνει ακαδημαϊκού εξερεύνηση στο συγκεκριμένο framework μιας και δεν είναι τόσο ευρέως διαδεδομένο ιδιαίτερα στον κόσμο της αγοράς.

Το JHipster παρέχει τη δυνατότητα δημιουργίας μίας πλήρους εφαρμογής που περιλαμβάνει ένα backend κομματι σε Java και συγκεκριμένα σε Spring Boot, ένα frontend κομμάτι σε Angular, React ή Vue (δίνει τη δυνατότητα στο χρήστη να διαλέξει τι προτιμάει), ενσωμάτωση Hibernate ORM για διαχείριση της βάσης δεδομένων, υποστήριξη REST APIs ή GraphQL και εργαλεία για CI/CD (π.χ. Docker, Kubernetes, Jenkins). Η σημαντικότερη ίσως δυνατότητα του JHipster είναι η διαδραστική διαδικασία δημιουργίας εφαρμογής μέσω CLI, που επιτρέπει στον προγραμματιστή να επιλέξει με ευκολία την τεχνολογική στοίβα, τα πρότυπα ασφαλείας, τη βάση δεδομένων και το deployment περιβάλλον, μειώνοντας σημαντικά τον χρόνο αρχικοποίησης (scaffolding) [4]



Δομή αρχείων JHipster 1



Δομή αρχείων JHipster 2

Για να καταλάβει κανείς την έκταση της εφαρμογής που δημιουργεί από μόνο του το JHipster είναι χρήσιμο να ανατρέξει στις παραπάνω δύο εικόνες. Κατά τη δημιουργία της εφαρμογής, το JHipster παράγει μια πλήρη και καλά οργανωμένη δομή αρχείων που διαχωρίζει καθαρά τα επίπεδα του backend και του frontend, σύμφωνα με τις αρχές της αρχιτεκτονικής full-stack. Στη ρίζα του έργου δημιουργείται φάκελος με τα απαραίτητα αρχεία ρύθμισης, όπως τα pom.xml (για Maven builds), .yo-rc.json (που περιέχει τις ρυθμίσεις της εφαρμογής από το generator) και τα αρχεία Docker, CI/CD ή Kubernetes, αν έχουν επιλεγεί κατά τη δημιουργία. Ο φάκελος src/main/java περιέχει την επιχειρησιακή λογική της εφαρμογής, οργανωμένη σε πακέτα όπως domain (οντότητες), repository (interfaces JPA), service (επιχειρησιακή λογική), και web.rest (REST controllers). Το src/main/resources περιέχει τα αρχεία ρυθμίσεων (application.yml), τα templates μηνυμάτων, και τα Liquibase changelogs για τη βάση δεδομένων. Αντίστοιχα, το frontend τοποθετείται στον φάκελο src/main/webapp, με δομή βασισμένη στο framework που έχει επιλεγεί (π.χ. Angular), όπου περιλαμβάνονται οι φάκελοι app/ (components, modules, routing), content/ (CSS, εικόνες), και i18n/ (πολυγλωσσική υποστήριξη). Η συγκεκριμένη δομή αρχείων διευκολύνει τη συντήρηση, την επέκταση και τη συνεργασία μεταξύ διαφορετικών ομάδων ανάπτυξης, επιτρέποντας τον παράλληλο χειρισμό backend και frontend στοιχείων χωρίς επικαλύψεις ή σύγχυση.

Παρόλο που το JHipster προσφέρει εντυπωσιακές δυνατότητες για γρήγορη παραγωγή εφαρμογών πλήρους στοίβας, η επέμβαση και η επέκταση του παραγόμενου κώδικα ενδέχεται να αποδειχθούν απαιτητικές, ιδιαίτερα σε πιο σύνθετα ή μη τυποποιημένα σενάρια. Η αυτοματοποιημένη δημιουργία αρχείων – είτε μέσω CLI είτε μέσω JDL – βασίζεται σε προκαθορισμένες δομές και patterns, τα οποία

λειτουργούν εξαιρετικά για CRUD εφαρμογές, αλλά γίνονται πιο περιοριστικά όταν επιχειρείται η εισαγωγή προσαρμοσμένης επιχειρησιακής λογικής, custom validations ή πολύπλοκων σχέσεων μεταξύ οντοτήτων. Επιπλέον, κάθε φορά που απαιτείται αλλαγή στο μοντέλο δεδομένων (π.χ. προσθήκη πεδίου ή νέας σχέσης), η διαδικασία regeneration του κώδικα μπορεί να οδηγήσει σε επικαλύψεις ή ακόμα και απώλεια χειροποίητων επεμβάσεων, εάν δεν υπάρχει σαφής διαχωρισμός μεταξύ auto-generated και custom αρχείων. Ακόμη και σε πιο modular κομμάτια, όπως οι REST controllers ή τα service layers, η χρήση annotations και auto-wiring του Spring καθιστά την τροποποίηση λιγότερο προβλέψιμη και συχνά απαιτεί σε βάθος γνώση του πλαισίου. Στην πράξη, η προσθήκη νέας λειτουργικότητας εκτός των ορίων του scaffolding απαιτεί προσοχή, καλή κατανόηση της εσωτερικής ροής της εφαρμογής και συνεχή τεκμηρίωση, καθώς οποιαδήποτε μη συστηματοποιημένη αλλαγή μπορεί να διαρρήξει την ευστάθεια του έργου ή να δυσχεράνει μελλοντικές ενημερώσεις.

Εμβαθύνοντας στις λειτουργικότητες του JHipster που αξιοποιήθηκαν στο πλαίσιο αυτής της διπλωματικής, ιδιαίτερη σημασία είχε η χρήση του JDL Studio (JHipster Domain Language), ενός εργαλείου που επιτρέπει την περιγραφή των οντοτήτων της εφαρμογής με έναν απλό, declarative τρόπο. Μέσω αυτού του εργαλείου, ο προγραμματιστής μπορεί να ορίσει τα entities, τα πεδία τους, τις σχέσεις μεταξύ τους και τα constraints, σε μια ενιαία, ευανάγνωστη DSL σύνταξη. Κατά την εκτέλεση της JDL, το JHipster δημιουργεί αυτόματα το αντίστοιχο μοντέλο της βάσης δεδομένων, τους JPA entities, τα repositories, τους REST controllers και τα αντίστοιχα Angular modules, χτίζοντας έτσι παραλληλα τόσο τη λογική του backend όσο και το αντίστοιχο κομμάτι του frontend.

Ένα επιπλέον πλεονέκτημα που αξιοποιήθηκε στην παρούσα εργασία είναι η αυτόματη δημιουργία διαχειριστικού περιβάλλοντος (administration UI) για κάθε entity που ορίζεται. Το JHipster δημιουργεί αυτόματα φόρμες δημιουργίας, επεξεργασίας και προβολής, με πλήρη λειτουργικότητα CRUD (Create Read Update Delete), παρέχοντας έτσι ένα έτοιμο backend panel χωρίς την ανάγκη περαιτέρω υλοποίησης.

Παράλληλα, το JHipster ενσωματώνει αυτόματα ένα Swagger UI (OpenAPI 3 interface) για όλα τα διαθέσιμα endpoints της εφαρμογής. Το Swagger παράγεται δυναμικά κατά την εκκίνηση του backend και προσφέρει τεκμηρίωση, δοκιμαστικά calls και αναλυτική προβολή των API responses. Αυτό αποδείχθηκε εξαιρετικά χρήσιμο τόσο στη φάση του debugging όσο και στην επικοινωνία μεταξύ frontend και backend, καθώς παρείχε έναν ξεκάθαρο και δομημένο τρόπο ελέγχου της API σύμβασης μεταξύ των μερών.

Συνολικά, οι λειτουργίες αυτές καθιστούν το JHipster ένα **παραγωγικά πολύτιμο εργαλείο** για μικρές αλλά και σύνθετες εφαρμογές, καθώς μειώνουν τον χρόνο ανάπτυξης, διατηρούν τη συνοχή του κώδικα και προσφέρουν ενσωματωμένες best practices για ασφάλεια, modularity και διαλειτουργικότητα.

Αναλυτικά παρατίθεται ο κώδικας στο APPENDIX B

Υπάρχουν όμως και άλλα σύγχρονα εργαλεία που επιδιώκουν να καλύψουν αντίστοιχες ανάγκες, και αξίζει να παρουσιαστούν ως εναλλακτικές του JHipster, παρόλο που δε θα ασχοληθούμε με αυτά στην παρούσα διπλωματική.

Spring Roo

Το Spring Roo είναι ένα εργαλείο scaffolding της ίδιας οικογένειας τεχνολογιών με το JHipster (Spring), εστιασμένο καθαρά στον κόσμο της Java. Αν και προσφέρει ταχεία δημιουργία CRUD εφαρμογών με Spring Boot, δεν δίνει έμφαση στο frontend, ενώ η φιλοσοφία του είναι πιο "opinionated", με λιγότερη ευελιξία για προσαρμογές. Είναι κατάλληλο για πολύ γρήγορο πρωτοτυποποίηση εφαρμογών εντός αμιγώς Spring περιβάλλοντος, αλλά όχι για πλήρη full-stack συστήματα με έμφαση στο UI. [5]

Yeoman

Το JHipster βασίζεται τεχνικά στο Yeoman, έναν γενικού σκοπού scaffolder με μεγάλη κοινότητα και ποικιλία από generators (για React apps, Node.js APIs, Electron apps κ.ά.). Παρότι το Yeoman είναι πιο ευέλικτο και προσαρμόσιμο, απαιτεί περισσότερη δουλειά για να φτάσει στο επίπεδο παραγωγής μιας εφαρμογής JHipster, καθώς δεν προσφέρει ενιαία διαχείριση αρχιτεκτονικής, authentication ή DevOps. [6]

Nx (by Nrwl)

Το Nx είναι ένα εξελιγμένο εργαλείο ανάπτυξης enterprise-grade εφαρμογών, με υποστήριξη για monorepos, micro-frontends και πλήρες integration με Angular, React, NestJS και Node. Εστιάζει κυρίως στο οικοσύστημα JavaScript/TypeScript και απευθύνεται σε ομάδες που χρειάζονται υψηλό modularity και tooling σε μεγάλα έργα. Μπορεί να καλύψει τις ίδιες ανάγκες με το JHipster, κυρίως όταν η στοίβα είναι καθαρά JS-based, αλλά στερείται της ενσωμάτωσης με Spring/Java περιβάλλοντα. [7]

LoopBack (IBM)

Το LoopBack είναι ένα Node.js framework που παρέχει ένα ισχυρό CLI για την ταχεία δημιουργία REST APIs, υποστήριξη OpenAPI/Swagger και connectors για διάφορες βάσεις δεδομένων. Αν και πολύ παραγωγικό για API-first αρχιτεκτονικές, δεν συνοδεύεται από πλήρη frontend scaffolding ή ρυθμίσεις ασφαλείας επιπέδου OAuth2/JWT με το ίδιο βάθος όπως το JHipster. Ιδανικό για γρήγορη δημιουργία backend APIs με λιγότερη πολυπλοκότητα. [8]

ABP Framework

Το ABP αποτελεί την πλησιέστερη εναλλακτική του JHipster στο οικοσύστημα .NET. Είναι modular, υποστηρίζει domain-driven design, πολυμισθωτικότητα (multi-tenancy), ενσωματωμένα patterns για authentication, permissions και unit of work, και συνοδεύεται από Angular ή Blazor frontend templates. Προσφέρει πλήρες

tooling για παραγωγική ανάπτυξη .NET εφαρμογών, και μπορεί να θεωρηθεί το “JHipster του .NET”.[9]

3.2 Δυνατότητες πλατφόρμας

Η πλατφόρμα που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής αξιοποιεί στο έπακρο τις δυνατότητες του JHipster, δημιουργώντας ένα πλήρως λειτουργικό, ασφαλές και επεκτάσιμο πληροφοριακό σύστημα που υποστηρίζει διακριτούς ρόλους χρήσης, τεχνολογίες αιχμής και σύγχρονες πρακτικές ανάπτυξης. Η εφαρμογή έχει σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να διαχειρίζεται διαφορετικούς τύπους χρηστών, να προσαρμόζεται στις ανάγκες κάθε ρόλου και να συνδυάζει τον παραδοσιακό τρόπο χρήσης (μέσω φορμών) με καινοτόμες προσεγγίσεις όπως η χρήση φυσικής γλώσσας.

Συγκεκριμένα, υποστηρίζονται δύο βασικοί ρόλοι: ο χρήστης (user) και ο διαχειριστής (admin). Ο απλός χρήστης έχει τη δυνατότητα να δημιουργεί και να επεξεργάζεται δηλώσεις Πόθεν Έσχες, να παρακολουθεί την πορεία τους και να προβάλει το ιστορικό των ενεργειών του. Από την άλλη πλευρά, ο admin διαθέτει προνομιακή πρόσβαση σε όλες τις καταχωρήσεις των χρηστών, έχει δικαιώματα επεξεργασίας συγκεκριμένων μεταδεδομένων, όπως ιδιότητα, επιτροπή ή βαθμός, και επιπλέον μπορεί να αλληλεπιδρά με το ενσωματωμένο AI chatbot προκειμένου να αναζητήσει πληροφορίες σχετικά με το ιστορικό των υποβολών. Η αλληλεπίδραση αυτή γίνεται με τη χρήση φυσικής γλώσσας, επιτρέποντας στον διαχειριστή να εκφράσει ελεύθερα τα ερωτήματά του, όπως π.χ. “ποιες δηλώσεις άλλαξαν την τελευταία εβδομάδα” ή “πόσες δηλώσεις έκανε ο χρήστης X”.

Η πλατφόρμα υποστηρίζει δύο εναλλακτικούς τρόπους σύνδεσης: είτε μέσω τοπικών διαπιστευτηρίων που δημιουργούνται στο σύστημα, είτε μέσω OAuth 2.0 αυθεντικοποίησης μέσω της επίσημης πύλης TaxisNet (gsis.gr), εξασφαλίζοντας έτσι εναλλακτικές οδούς πρόσβασης ανάλογα με τις ανάγκες και τον τύπο χρήστη. Το σύστημα ταυτοποίησης βασίζεται στο Spring Security, με role-based access control, ενώ κάθε endpoint του backend προστατεύεται από φίλτρα ασφαλείας ανάλογα με τα δικαιώματα πρόσβασης του συνδεδεμένου χρήστη.

Από τεχνική άποψη, το backend αξιοποιεί πλήρως τις δυνατότητες που παρέχει το JHipster, δημιουργώντας entity-driven services που υποστηρίζουν pagination, filtering, auditing και versioning. Η παρακολούθηση αλλαγών γίνεται μέσω audit logs, ενώ η διαχείριση των μεταναστεύσεων της βάσης δεδομένων υλοποιείται με το εργαλείο Liquibase, επιτρέποντας versioned schema updates με πλήρη traceability.

Στην πλευρά του frontend, το Angular framework χρησιμοποιήθηκε για την υλοποίηση ενός αυστηρά οργανωμένου UI, με κατανομή του κώδικα σε modules, components και services, όπως επιτάσσουν οι σύγχρονες αρχές του SPA development. Το σύστημα routing του Angular διαθέτει role-based guards, οι οποίοι φροντίζουν ώστε κάθε χρήστης να έχει πρόσβαση αποκλειστικά στις λειτουργίες που του αναλογούν. Η επικοινωνία με το backend γίνεται μέσω του HttpClient, με χρήση

interceptors για την αυτόματη προσθήκη authentication headers (JWT ή access tokens από το TaxisNet), καθώς και για την κεντρική διαχείριση σφαλμάτων.

Επιπλέον, η πλατφόρμα αξιοποιεί σύγχρονα εργαλεία όπως το Swagger UI, το οποίο προσφέρει διαδραστική τεκμηρίωση των REST APIs της εφαρμογής. Το MapStruct χρησιμοποιείται για τον μετασχηματισμό DTOs σε entities και αντίστροφα, επιτρέποντας καθαρή διαχείριση δεδομένων μεταξύ των layers της εφαρμογής. Αν και στην παρούσα υλοποίηση δεν έγινε χρήση μικροϋπηρεσιών, το JHipster υποστηρίζει το JHipster Registry, ένα εργαλείο διαχείρισης microservices που μπορεί να αξιοποιηθεί μελλοντικά για κλιμάκωση της πλατφόρμας.

Η συνολική υλοποίηση καταδεικνύει ότι η πλατφόρμα είναι τεχνικά στιβαρή, λειτουργικά ευέλικτη και αρχιτεκτονικά επεκτάσιμη. Ο διαχωρισμός ρόλων, η διασύνδεση με κρατικούς παρόχους, η ενσωμάτωση μηχανισμών auditing και η χρήση AI τεχνολογιών συνθέτουν ένα σύστημα που όχι μόνο ανταποκρίνεται στις απαιτήσεις του πεδίου εφαρμογής του, αλλά θέτει και τις βάσεις για μελλοντική εξέλιξη.

3.3 Επεκτασιμότητα και περιορισμοί

Η αρχιτεκτονική του JHipster προσφέρει σημαντικές δυνατότητες επεκτασιμότητας, ιδίως όταν συνδυάζεται με σύγχρονες πρακτικές containerization και DevOps. Η παραγωγή του κώδικα είναι συμβατή με Docker και μπορεί εύκολα να ενορχηστρωθεί με Kubernetes, προσφέροντας έναν ισχυρό μηχανισμό scaling τόσο σε οριζόντιο όσο και σε κατανεμημένο επίπεδο. Η πλατφόρμα ενσωματώνεται ομαλά με CI/CD pipelines μέσω εργαλείων όπως Jenkins, GitLab CI ή GitHub Actions, επιτρέποντας την αυτοματοποιημένη διαδικασία build, test και deployment. Αυτή η δυνατότητα αποδεικνύεται ιδιαίτερα χρήσιμη σε περιβάλλοντα που απαιτούν συνεχή ενσωμάτωση και παράδοση (continuous integration/delivery), παρέχοντας σταθερότητα και επαναληψιμότητα σε κάθε φάση του κύκλου ζωής της εφαρμογής.

Ωστόσο, παρά τα παραπάνω πλεονεκτήματα, η επεκτασιμότητα του JHipster σε επίπεδο κώδικα παρουσιάζει συγκεκριμένους περιορισμούς. Αν και είναι εξαιρετικό εργαλείο για την αρχική δημιουργία (scaffolding) μιας εφαρμογής, το μοντέλο δημιουργίας που χρησιμοποιεί – ειδικά με χρήση JDL – γίνεται σταδιακά πιο άκαμπτο όταν προκύπτουν ανάγκες για πιο εξειδικευμένη επιχειρησιακή λογική ή μη τυπικές σχέσεις μεταξύ οντοτήτων. Οι αλλαγές στη βάση δεδομένων, για παράδειγμα, απαιτούν επανεκτέλεση του generator, κάτι που μπορεί να δημιουργήσει προβλήματα εάν έχει ήδη προστεθεί custom business logic στον παραγόμενο κώδικα. Αν δεν υπάρχει σαφής διαχωρισμός ανάμεσα στον "generated" και τον "χειροποίητο" κώδικα, υπάρχει σοβαρός κίνδυνος απώλειας αλλαγών.

Επιπλέον, η τροποποίηση της backend λογικής που δημιουργείται από το JHipster απαιτεί εξοικείωση με το πλήρες οικοσύστημα του Spring Boot και των εργαλείων του, γεγονός που μπορεί να αποθαρρύνει λιγότερο έμπειρους προγραμματιστές. Η πυκνή χρήση annotations, configurations και layered abstractions καθιστά το

debugging πιο σύνθετο, ειδικά όταν πρόκειται για περίπλοκα authorization policies ή custom services που ξεφεύγουν από το βασικό CRUD pattern.

Από την άλλη πλευρά, η Angular πλευρά της εφαρμογής αποδείχθηκε πολύ πιο ευέλικτη. Η φιλοσοφία του component-based UI επιτρέπει την εύκολη προσθήκη ή αντικατάσταση λειτουργιών χωρίς να διαταράσσεται η υπόλοιπη εφαρμογή. Νέα modules, routing paths, guards και UI widgets μπορούν να ενσωματωθούν σταδιακά, με σταθερό έλεγχο της λειτουργικότητας. Επιπλέον, το Angular CLI (command line interface) προσφέρει ανάλογες scaffolding δυνατότητες με το JHipster, αλλά με μικρότερη εξάρτηση από την κεντρική δομή του project.

Συνοψίζοντας, η πλατφόρμα επιτρέπει επεκτασιμότητα σε επίπεδο υποδομής και αρχιτεκτονικής, ειδικά μέσω των εργαλείων containerization και DevOps, αλλά η επέκταση του παραγόμενου κώδικα στο backend απαιτεί ιδιαίτερη προσοχή, γνώση του JHipster οικοσυστήματος και καθαρή στρατηγική διαχείρισης του τεχνικού χρέους που προκύπτει από την auto-generated φύση της αρχικής εφαρμογής. Η επιτυχής εξέλιξη του έργου εξαρτάται από την ικανότητα της ομάδας να συνδυάζει τη δύναμη του scaffolding με χειροποίητη παρέμβαση εκεί όπου απαιτείται ευελιξία και προσαρμογή.

3.4 Modules και επεκτάσεις που υλοποιήθηκαν

Στο πλαίσιο της διπλωματικής, δημιουργήθηκαν πλήρως παραμετροποιημένα modules για τη διαχείριση δηλώσεων, την υποβολή αιτήσεων, καθώς και για το audit των αλλαγών και την ανίχνευση χειρισμών από χρήστες και διαχειριστές. Το audit σύστημα παρακολουθεί αλλαγές σε επίπεδο πεδίου και αποθηκεύει το ιστορικό, με δυνατότητα προβολής μέσω του UI από εξουσιοδοτημένους ρόλους.

Η πιο καινοτόμα επέκταση αφορά το AI Chatbot, το οποίο ενσωματώθηκε ως εξωτερική υπηρεσία (microservice), η οποία λαμβάνει queries σε φυσική γλώσσα και επιστρέφει οργανωμένες απαντήσεις βασισμένες σε δεδομένα του συστήματος. Το chatbot χρησιμοποιεί custom endpoints του backend, έχει δικαιώματα μόνο ανάγνωσης και μπορεί να λειτουργεί σε συνθήκες περιορισμένης προσβασιμότητας, διατηρώντας ασφάλεια και απομόνωση από κρίσιμα δεδομένα χρήστη.

Ένα αρκετά χρήσιμο component – module που υλοποιήθηκε είναι από πλευράς frontend και έχει το όνομα middleware. Όπως περιγράφει και το όνομά του λειτουργεί ως ενδιάμεσος πόλος διασύνδεσης μέσω των διαφορετικών επιλογών εισόδου όπως π.χ. της πύλης του gsis.gr-taxisnet και του keycloak openid-connect (που θα αναλυθεί παρακάτω). Το module αυτό είναι υπεύθυνο για την επικοινωνία του frontend και του backend σε επίπεδο λογικής εισόδου στην εφαρμογή εξασφαλίζοντας για το εκάστοτε use case την αντίστοιχη διαδικασία εισόδου λαμβάνοντας από το third-party server τα απαραίτητα στοιχεία που χρειάζονται και ανταλλάσσοντάς τα με ένα token που χρησιμοποιεί η δική μας εφαρμογή για αυθεντικοποίηση.

Παρακάτω φαίνεται η υλοποίηση σε κώδικα του συγκεκριμένου module.

```typescript
export class MiddlewareComponent implements OnInit {
  constructor(private authService: AuthService, private router: Router, private route: ActivatedRoute){}

  ngOnInit(): void {
    this.route.queryParams.subscribe((params) => {
      const code = params['code']; // Get 'code' from the URL
      const iss = params['iss'];
      const sessionState = params['session_state'];
      const paramKeys = Object.keys(params);

      const isKeycloakLogin =
        code && iss && sessionState &&
        paramKeys.length === 3;

      if(isKeycloakLogin){
        console.log('Keycloak login detected');
        console.log('Code found in URL:', code);
        this.authService.loginKeycloak(code).subscribe(response => {
          this.authService.saveToken(response.id_token);
          this.authService.saveOAuthMethod('keycloak');
          const roles = ['ROLE_USER']
          this.authService.saveRoles(['ROLE_USER']);

          // Redirect based on role
          if (roles.includes('ROLE_ADMIN')) {
            this.router.navigate(['/admin/home']);
          } else if (roles.includes('ROLE_USER')) {
            this.router.navigate(['/user/home']);
          }
        }, error => {
          console.error('Oauth2 Login failed', error);
          alert('Απέτυχε η αυθεντικοποίηση μέσω keycloak')
          this.router.navigate(['/login'])
        })
      }
      else if (code) {
        console.log('Code found in URL:', code);
        this.authService.loginOauth2(code).subscribe(response => {
          //console.log('Login successful!', response);
          this.authService.saveToken(response.id_token);
          this.authService.saveOAuthMethod('oauth2');
          const roles = this.authService.getRoles();
          this.authService.saveRoles(roles);

          // Redirect based on role
          if (roles.includes('ROLE_ADMIN')) {
            this.router.navigate(['/admin/home']);
          } else if (roles.includes('ROLE_USER')) {
            this.router.navigate(['/user/home']);
          }
        }, error => {
          console.error('Oauth2 Login failed', error);
          alert('Απέτυχε η αυθεντικοποίηση μέσω taxisnet')
          this.router.navigate(['/login'])
        })
      }
      else if (this.authService.isAuthenticated()) {
        const roles = this.authService.getRoles();

        // Redirect based on authenticated role
        if (roles.includes('ROLE_ADMIN')) {
          this.router.navigate(['/admin/home']);
        } else if (roles.includes('ROLE_USER')) {
          this.router.navigate(['/user/home']);
        }
      }
      else {
        console.log('No code found and not authenticated, redirecting to login');
        this.router.navigate(['/login']);
      }
    });
  }
}
```

3.5 Τεχνολογίες LLMs: Mistral, Vicuna, DeepSeek

Ένα από τα πιο καινοτόμα και ερευνητικά ενδιαφέροντα στοιχεία της παρούσας διπλωματικής ήταν η αξιοποίηση μοντέλων Τεχνητής Νοημοσύνης τύπου LLM (Large Language Models) για την ενίσχυση της πλατφόρμας με δυνατότητες κατανόησης φυσικής γλώσσας. Σκοπός ήταν η δημιουργία ενός chatbot που να μπορεί να επεξεργάζεται ερωτήματα χρηστών με βάση τα audit δεδομένα των δηλώσεων Πόθεν Έσχες και να επιστρέφει χρήσιμες, στοχευμένες απαντήσεις χωρίς τη χρήση SQL ή τεχνικής αναζήτησης.

Για την υλοποίηση χρησιμοποιήθηκε η υποδομή του **Ollama**, ένα lightweight περιβάλλον εκτέλεσης LLMs τοπικά (on-device), ιδανικό για δοκιμές, development και παραγωγή με χαμηλό latency. Η χρήση του Ollama επέτρεψε τη φόρτωση και λειτουργία μοντέλων όπως το **Mistral**, το **DeepSeek** και το **Vicuna**, προσφέροντας ευελιξία στην επιλογή του κατάλληλου μοντέλου ανάλογα με τη χρήση και τα δεδομένα.

Πάνω σε αυτή τη βάση, αξιοποιήθηκε το **Krikri**, ένα ανοιχτού κώδικα εργαλείο που λειτουργεί ως wrapper/chatbot framework για μοντέλα LLM, προσφέροντας διεπαφή ερωτήσεων/απαντήσεων, υποστήριξη ιστορικού και βασική ανάλυση intents. Το Krikri διασυνδέθηκε με την εφαρμογή μέσω Flask API.

Το **Krikri** αποτελεί μια από τις πιο καινοτόμες ελληνικές προσπάθειες στον χώρο της Τεχνητής Νοημοσύνης, εστιάζοντας στη δημιουργία διαλογικών συστημάτων που βασίζονται σε μεγάλα γλωσσικά μοντέλα (LLMs). Χτισμένο πάνω στο τεχνολογικό υπόβαθρο του **LLaMA** (Large Language Model Meta AI), το Krikri ξεχωρίζει όχι μόνο για την τοπική του ανάπτυξη, αλλά και για την ενσωμάτωσή του σε πρακτικές, παραγωγικές εφαρμογές που απαιτούν επεξεργασία φυσικής γλώσσας με ακρίβεια, ταχύτητα και συμβατότητα με ελληνικά δεδομένα. Στο πλαίσιο αυτής της διπλωματικής, το Krikri αξιοποιήθηκε για την υποστήριξη ενός conversational AI υποσυστήματος που επεξεργάζεται audit logs από δηλώσεις Πόθεν Έσχες, απαντώντας σε ερωτήματα χρηστών σε απλή φυσική γλώσσα. Η ευκολία ενσωμάτωσης, η ταχύτητα απόκρισης και η υποστήριξη ελληνικής γλώσσας καθιστούν το Krikri ιδιαίτερα κατάλληλο για εφαρμογές στον δημόσιο τομέα, ενισχύοντας την προσβασιμότητα και την κατανόηση δεδομένων από μη τεχνικούς χρήστες. [16]

Στο επίπεδο των LLMs, εξετάστηκαν και αξιολογήθηκαν τρία διαφορετικά μοντέλα:

- **Vicuna**: Πρόκειται για μοντέλο βασισμένο στο LLaMA της Meta, fine-tuned για συνομιλιακή κατανόηση και αλληλεπίδραση. Αποδείχθηκε εξαιρετικό στην κατανόηση ερωτήσεων σε φυσική γλώσσα και στην παραγωγή ροών διαλόγου με συνέπεια και φυσικότητα.[10]

- **DeepSeek**: Ένα ανοιχτού κώδικα μοντέλο από την κινεζική εταιρεία DeepSeek AI, με έμφαση στη λογική και τη δομημένη παραγωγή. Κατάλληλο για queries που απαιτούν ακρίβεια, ειδικά όταν περιλαμβάνουν μαθηματικά ή SQL-style reasoning.[11]
- **Mistral**: Ένα μικρό, γρήγορο μοντέλο το οποίο αποδείχθηκε ιδιαίτερα χρήσιμο για ερωτήματα μικρού εύρους. Αν και μειονεκτούσε ελαφρώς στη διαχείριση εκτεταμένου context, προσέφερε πολύ χαμηλό latency.[12]

Κατά τη φάση αξιολόγησης και benchmarking, διαπιστώθηκε ότι το **Vicuna** είχε την καλύτερη ισορροπία μεταξύ φυσικότητας απάντησης, συνέπειας και κατανόησης ερωτημάτων. Το **DeepSeek** ήταν πιο «τεχνικά ακριβές», ιδανικό για σύνθετα φίλτρα και queries με αναφορές σε μεταδεδομένα, αλλά η απαντητική του ροή ήταν λιγότερο φιλική προς τον χρήστη. Το **Mistral**, από την άλλη, ξεχώρισε για την ταχύτητά του αλλά υστερούσε σε context awareness. Με βάση αυτά τα αποτελέσματα, επιλέχθηκε τελικά το **Vicuna** ως η προεπιλεγμένη επιλογή για το chatbot, συνδυάζοντας ακρίβεια και εμπειρία χρήστη.

Η προσθήκη του υποσυστήματος αυτού ενίσχυσε καθοριστικά την προσβασιμότητα της πλατφόρμας, καθώς επέτρεψε σε χρήστες —ιδίως σε admins— να εκτελούν πολύπλοκες αναζητήσεις χωρίς τεχνικές γνώσεις, με τρόπο διαδραστικό και ανθρώπινο. Η ενοποίηση του Krikri με το UI της εφαρμογής έγινε μέσω απλών POST requests προς τον Flask server, ο οποίος ερμηνεύει το αίτημα και επιστρέφει απάντηση βασισμένη στο audit αρχείο που του έχει δοθεί.

3.6 Συμπεράσματα

Η ανάπτυξη της πλατφόρμας κατέδειξε τη σημασία της στρατηγικής επιλογής εργαλείων, ανάλογα με τις ανάγκες κάθε project. Το JHipster αποδείχθηκε κατάλληλο για τον σχεδιασμό ενός κρατικού τύπου πληροφοριακού συστήματος, εξασφαλίζοντας την ταχεία υλοποίηση, την ασφάλεια και τη δυνατότητα κλιμάκωσης. Το Angular παρείχε τη βάση για modular UI ανάπτυξη και role-based rendering, ενώ η ενσωμάτωση LLMs μέσω Vicuna εμπλούτισε την εμπειρία χρήστη με φυσική γλώσσα, μειώνοντας το τεχνολογικό φράγμα στην πρόσβαση δεδομένων.

Είναι πλέον προφανές ότι η τεχνητή νοημοσύνη δεν είναι μια “προσθήκη πολυτελείας” αλλά ένα αναπόσπαστο μέρος της αρχιτεκτονικής και της σχεδίασης. Τα εργαλεία που επιλέγονται σήμερα θα καθορίσουν όχι μόνο τη λειτουργικότητα, αλλά και τη βιωσιμότητα μιας εφαρμογής στο μέλλον. Για το είδος του προβλήματος που πραγματεύεται η εργασία – ένα ψηφιακό εργαλείο με θεσμική βαρύτητα και απαίτηση για προσβασιμότητα – η επιλογή συνδυασμένων τεχνολογιών backend, frontend και AI αποδείχθηκε ουσιώδης και απόλυτα εύστοχη.

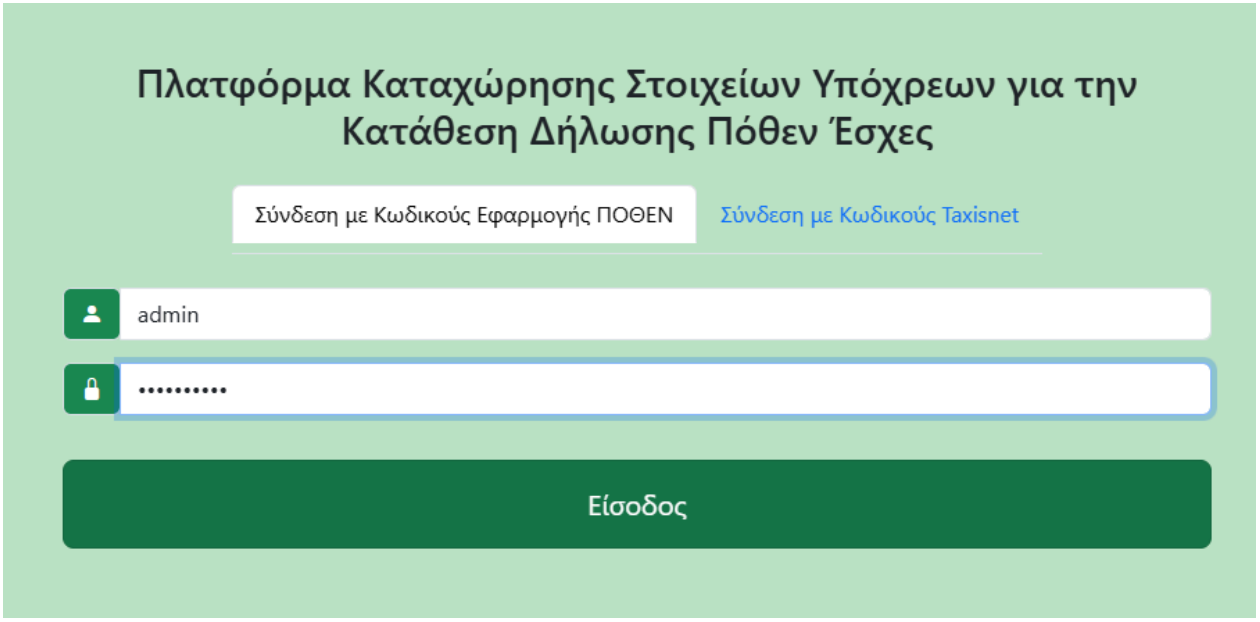
ΚΕΦΑΛΑΙΟ 4. ΑΝΑΛΥΣΗ ΔΙΑΔΙΚΑΣΙΩΝ ΚΑΙ ΔΟΜΗΣ

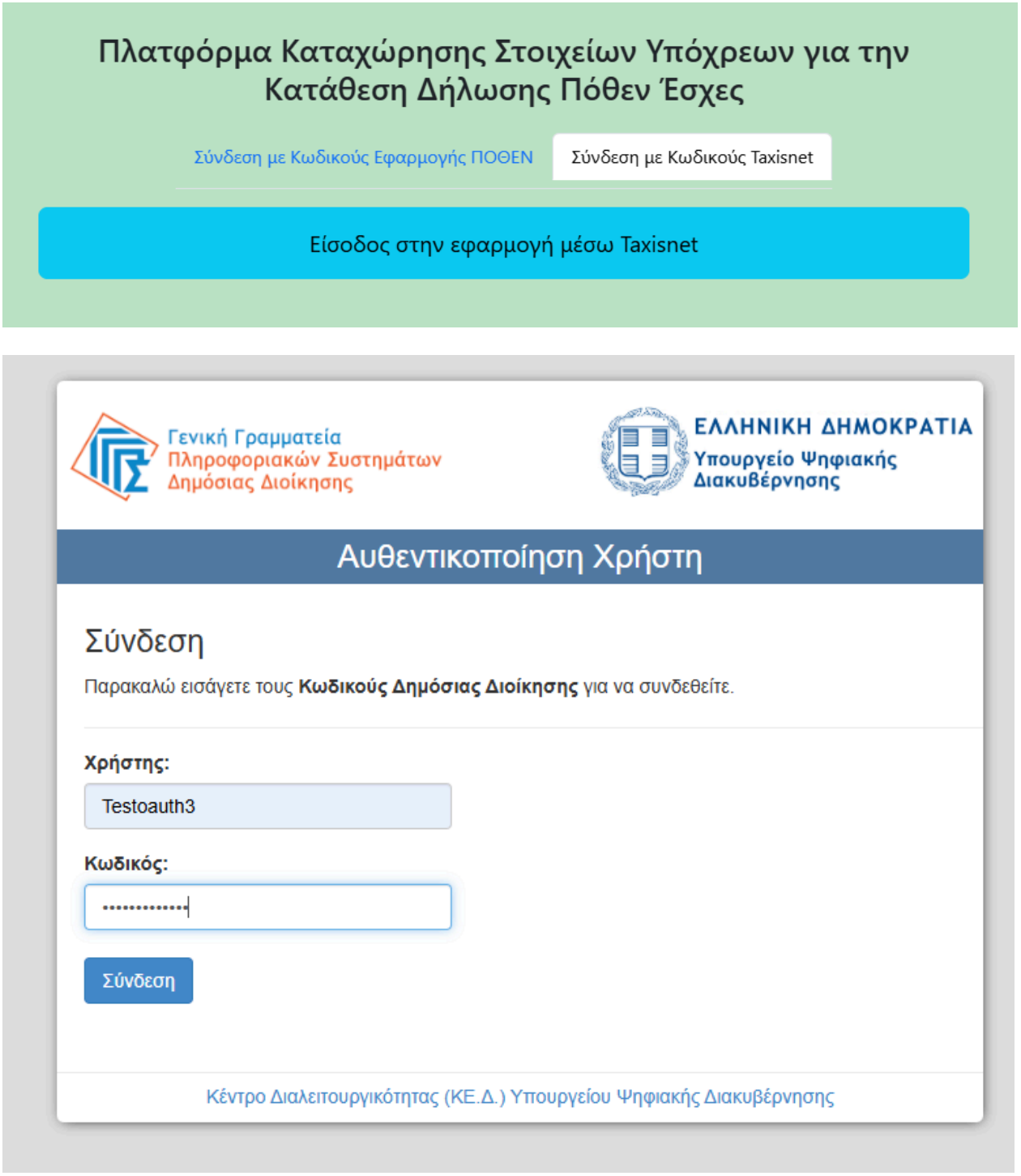
Η λειτουργία της πλατφόρμας βασίζεται σε έναν αριθμό κρίσιμων ροών πληροφορίας και αλληλεπιδράσεων μεταξύ χρηστών, διακομιστών και εξωτερικών υπηρεσιών. Οι ροές αυτές μπορούν να απεικονιστούν ως διαγράμματα διαδοχής (sequence diagrams), που παρουσιάζουν την ακολουθία ενεργειών από την πλευρά του client, μέσω των APIs, μέχρι την επεξεργασία και αποθήκευση στο backend. Παρακάτω αναλύονται τα σημαντικότερα sequence flows της εφαρμογής.

4.1 Διαδικασία σύνδεσης χρήστη (login)

Υπάρχουν δύο διαφορετικοί τρόποι εισόδου στο σύστημα. Ο πρώτος χρησιμοποιεί απλό μηχανισμό JWT authentication για ανάπτυξη/δοκιμές και χρήστες χωρίς TaxisNet. Ο δεύτερος αφορά την παραγωγική χρήση της εφαρμογής και βασίζεται στο OAuth2 πρωτόκολλο μέσω της αυθεντικοποίησης με την πλατφόρμα TaxisNet.

Κατά την είσοδο μέσω TaxisNet, ο χρήστης ανακατευθύνεται στην πύλη αυθεντικοποίησης της ΑΑΔΕ, εισάγει τα διαπιστευτήριά του και μετά από επιβεβαίωση, επιστρέφει στο frontend με access token. Το token αυτό αποθηκεύεται προσωρινά στο client και χρησιμοποιείται για επικοινωνία με τα προστατευμένα endpoints του backend. Από την πλευρά του server, γίνεται αποκωδικοποίηση του token και ταυτοποίηση του ρόλου του χρήστη (user ή admin) μέσω του embedded JWT claim. Παρακάτω φαίνονται σε εικόνες οι δύο τρόποι σύνδεσης στην εφαρμογή:





Login page μέσω jwt και μέσω taxisnet

Μια σημαντική λειτουργική επέκταση που υλοποιήθηκε στο πλαίσιο της διπλωματικής, με σκοπό την ενίσχυση της ευελιξίας και της φορητότητας της πλατφόρμας, ήταν η ενσωμάτωση του **Keycloak** ως εναλλακτικού παρόχου ταυτοποίησης, μέσω του πρωτοκόλλου **OpenID Connect (OIDC)**. Το Keycloak είναι ένα σύγχρονο, open-source identity and access management σύστημα, σχεδιασμένο για τη διαχείριση ταυτότητας χρηστών, authentication flows, role-based access

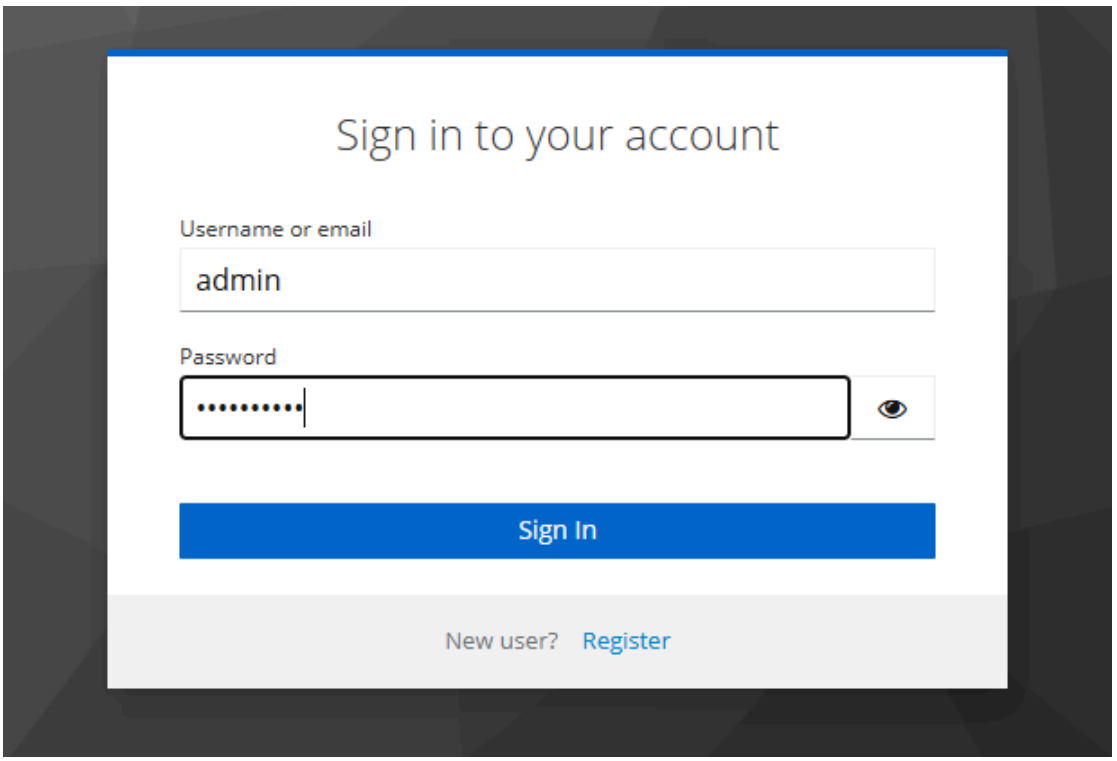
control (RBAC) και Single Sign-On (SSO), σε περιβάλλοντα εφαρμογών web και mobile. Αποτελεί μια από τις πιο διαδεδομένες λύσεις στην αγορά, καθώς προσφέρει δυνατότητα ενσωμάτωσης με πλήθος από πρωτόκολλα (OIDC, OAuth2, SAML 2.0), υποστήριξη για federation με εξωτερικούς παρόχους (Google, GitHub, LDAP, κ.ά.), ενώ διαθέτει ενσωματωμένο διαχειριστικό περιβάλλον για ρυθμίσεις χρηστών, ομάδων, clients και ρόλων.

Αρχιτεκτονικά, το Keycloak αποτελεί ένα standalone server-based σύστημα, το οποίο επικοινωνεί με την εφαρμογή ως **Authorization Server**. Ο client (δηλαδή το frontend της εφαρμογής) ανακατευθύνει τον χρήστη στον Keycloak για authentication και, μετά την επιτυχή ταυτοποίηση, λαμβάνει ένα **ID token** και ένα **access token**, τα οποία χρησιμοποιούνται για την επικύρωση της ταυτότητας και τη διαχείριση των session. Ο server της εφαρμογής (backend – JHipster/Spring Boot) επαληθεύει τα tokens και χορηγεί πρόσβαση βάσει του ρόλου που καθορίζεται είτε από claims εντός του token είτε από integration με το Keycloak UserInfo endpoint.[17]

Η συγκεκριμένη υλοποίηση σχεδιάστηκε ώστε να λειτουργεί **παράλληλα με την υφιστάμενη OAuth2 διασύνδεση με το TaxisNet**, δίνοντας τη δυνατότητα στην εφαρμογή να λειτουργήσει είτε με κρατική ταυτοποίηση είτε με αυτοδιαχειριζόμενο σύστημα χρηστών. Αυτό καθιστά την πλατφόρμα περισσότερο προσαρμόσιμη σε περιβάλλοντα όπου το TaxisNet δεν είναι διαθέσιμο (π.χ. σε εκπαιδευτικά ιδρύματα, οργανισμούς εκτός Ελλάδας ή για σκοπούς εσωτερικών δοκιμών/ανάπτυξης). Η χρήση του OpenID Connect – ως επάνω επίπεδο του OAuth2 – διασφαλίζει πλήρη συμβατότητα με τα σύγχρονα πρότυπα ασφαλείας και διευκολύνει την επέκταση σε σενάρια με πολλαπλούς παρόχους (multi-tenant authentication flows).

Επιπλέον, μέσω του Keycloak έγινε δυνατή η **εύκολη δημιουργία custom realms**, δηλαδή διακριτών χώρων χρήσης με ανεξάρτητα credentials και πολιτικές, κάτι που δίνει προοπτικές για μελλοντική υποστήριξη πολλαπλών οργανισμών ή φορέων εντός της ίδιας εφαρμογής. Η όλη υλοποίηση αποδείχθηκε τεχνικά εύρωστη, ασφαλής και πλήρως επεκτάσιμη, ενισχύοντας την ανεξαρτησία της πλατφόρμας από συγκεκριμένους παρόχους ταυτοποίησης και δίνοντας τη δυνατότητα ευρύτερης αξιοποίησής της σε πολυμορφικά περιβάλλοντα.

Αναλυτικά παρατίθεται ο κώδικας στο APPENDIX C



Login Page μέσω redirect στο keycloak UI

4.2 Υποβολή και τροποποίηση δήλωσης από το χρήστη

Αφού ο χρήστης συνδεθεί, έχει τη δυνατότητα να υποβάλει μία νέα δήλωση ΠΟΘΕΝ ΕΣΧΕΣ ή να τροποποιήσει προηγούμενη. Η διαδικασία περιλαμβάνει:

1. Το frontend εμφανίζει φόρμα με πεδία όπως ονοματεπώνυμο, ιδιότητα, επιτροπή, βαθμός, περιουσιακά στοιχεία κ.ά.
2. Με την υποβολή, τα δεδομένα αποστέλλονται στο backend μέσω POST request σε REST endpoint (/api/submissions).
3. Το backend επεξεργάζεται τα δεδομένα, τα αποθηκεύει στη βάση μέσω repository και δημιουργεί ένα audit log καταγράφοντας το event της υποβολής.
4. Αν ο χρήστης επιλέξει να τροποποιήσει υπάρχουσα δήλωση, πραγματοποιείται PUT ή PATCH request και το νέο περιεχόμενο καταγράφεται επίσης στα submission-audits.

Το σύστημα διατηρεί ιστορικό εκδόσεων, ώστε να είναι εφικτό κάθε audit να δείχνει το ακριβές περιεχόμενο πριν και μετά από κάθε αλλαγή.

Αναλυτικά παρατίθεται σχετικός κώδικας στο APPENDIX C

Παρακάτω παρατίθεται ένα κομμάτι κώδικα για τη διαχείριση των δηλώσεων από πλευράς angular εφαρμογής.

```
// Method to post a new submission
createSubmission(submission: any): Observable<any> {
  const headers = this.getAuthHeaders();
  return this.http.post<any>(this.apiUrl, submission, { headers });
}

// Method to update an existing submission
updateSubmission(submissionId: number, updatedSubmission: any): Observable<any> {
  const headers = this.getAuthHeaders();
  return this.http.put<any>(`${this.apiUrl}/${submissionId}`, updatedSubmission, { headers });
}

// Fetch paginated submissions
getSubmissions(page: number = 0, size: number = 20, eagerload: boolean = true): Observable<any> {
  const headers = this.getAuthHeaders();
  const params = {
    page: page.toString(),
    size: size.toString(),
    eagerload: eagerload.toString()
  };
  return this.http.get<any>(this.apiUrl, { headers, params });
}
```

Κώδικας υπεύθυνος για τη διαχείριση των δηλώσεων από πλευράς frontend

4.3 Διαχείριση πεδίων από τον admin

Ο διαχειριστής (admin) έχει πρόσβαση σε όλες τις δηλώσεις του συστήματος. Από το διαχειριστικό περιβάλλον μπορεί να τροποποιήσει κρίσιμα μετα-δεδομένα των δηλώσεων, όπως:

- Το όνομα της επιτροπής
- Τον βαθμό του υποβάλλοντος
- Την ιδιότητα ή κατηγορία του χρήστη

Οι αλλαγές αυτές ενημερώνουν άμεσα τα αντίστοιχα πεδία στη βάση και δημιουργούν νέα audit entries. Αυτή η δυνατότητα είναι κρίσιμη για την ορθή κατηγοριοποίηση των υποβαλλόμενων δηλώσεων και την προετοιμασία για έλεγχο από εποπτικές αρχές.

Παρακάτω παρατίθεται ενδεικτικά το κομμάτι κώδικα υπεύθυνο για τη διαχείριση των πεδίων των επιτροπών. Με αντίστοιχο τρόπο υλοποιήθηκαν η διαχείριση των πεδίων των βαθμών και των ιδιοτήτων.

```typescript
// Fetch paginated committees
getCommittees(page: number = 0, size: number = 20, eagerload: boolean = true): Observable<any> {
  const headers = this.getHeaders();
  const params = {
    page: page.toString(),
    size: size.toString(),
    eagerload: eagerload.toString(),
  };

  return this.http.get<any>(this.apiUrl, { headers, params });
}

// Create a new committee
createCommittee(committee: { name: string }): Observable<any> {
  const headers = this.getHeaders();
  return this.http.post<any>(this.apiUrl, committee, { headers });
}

// Update an existing committee
updateCommittee(id: number, committee: { name: string }): Observable<any> {
  const headers = this.getHeaders();
  return this.http.put<any>(`${this.apiUrl}/${id}`, { id, name: committee.name }, { headers });
}

// Delete a committee
deleteCommittee(id: number): Observable<any> {
  const headers = this.getHeaders();
  return this.http.delete<any>(`${this.apiUrl}/${id}`, { headers });
}
```

Κώδικας για την τροποποίηση πεδίων επιτροπών

4.4 Παρουσίαση των APIs και εξωτερικών επαφών

Η αρχιτεκτονική της πλατφόρμας βασίζεται εξ ολοκλήρου στην αρχή της αποκέντρωσης λειτουργιών μέσω RESTful APIs, τόσο για την εσωτερική επικοινωνία μεταξύ frontend και backend όσο και για την εξωτερική διασύνδεση με τρίτα συστήματα. Κάθε βασική λειτουργία της εφαρμογής αντιστοιχεί σε συγκεκριμένο API resource, το οποίο χειρίζεται την ανάκτηση, τροποποίηση και διαγραφή δεδομένων σύμφωνα με το πρότυπο REST και την αρχή του separation of concerns.

Εσωτερική Επικοινωνία: RESTful API Backend

Στο backend, τα APIs έχουν δομηθεί με βάση το Spring Boot και τη γενιά του JHipster, και καλύπτουν όλες τις βασικές λειτουργίες διαχείρισης δηλώσεων, χρηστών, ρόλων και ιστορικών μεταβολών. Τα endpoints είναι προσβάσιμα μέσω του προθέματος /api/ και προστατεύονται με JWT-based authentication, ενώ η προσβασιμότητα ρυθμίζεται βάσει ρόλων (admin/user) μέσω του μηχανισμού Spring Security.

Βασικά resources της εφαρμογής περιλαμβάνουν:

- **Submissions**: CRUD λειτουργίες μέσω των endpoints /api/submissions, για τη δημιουργία, επεξεργασία και ανάκτηση δηλώσεων Πόθεν Έσχες.

- **Submission Audits**: Ιστορικό ενεργειών και μεταβολών σε δηλώσεις, προσβάσιμο μέσω /api/submission-audits και φίλτρα ανά submission ID.
- **Users**: Πλήρες management χρηστών για τους διαχειριστές, μέσω /api/admin/users, με δυνατότητα δημιουργίας, ενημέρωσης και διαγραφής.
- **Entities υποστήριξης**: Περιλαμβάνουν τις οντότητες positions, grades, committees, που σχετίζονται με τα μεταδεδομένα κάθε δήλωσης και διαχειρίζονται μέσω ξεχωριστών endpoints.
- **Account Management**: Υποστήριξη εγγραφής, ενεργοποίησης, αλλαγής και ανάκτησης κωδικών μέσω του /api/account.
- **Authentication**: Υλοποιείται μέσω POST κλήσεων στο /api/authenticate για τοπικά διαπιστευτήρια και /api/authenticate-oauth2 για αυθεντικοποίηση μέσω TaxisNet.

Εξωτερικές Επαφές

Η πλατφόρμα διασυνδέεται με δύο βασικά εξωτερικά συστήματα:

1. **TaxisNet (OAuth2 Provider)**
   Ο μηχανισμός σύνδεσης μέσω TaxisNet υλοποιείται με βάση τη ροή Authorization Code του OAuth2. Ο χρήστης ανακατευθύνεται στον πάροχο (/api/authenticate-oauth2) και μετά το επιτυχές login επιστρέφει στην εφαρμογή με access token. Η πλατφόρμα κάνει χρήση αυτού του token για ασφαλή session management και αναγνώριση ρόλων. Η διασύνδεση αυτή διασφαλίζει κρατικά επαληθευμένη ταυτοποίηση, απαραίτητη για την αξιοπιστία της διαδικασίας Πόθεν Έσχες.

2. **Διασύνδεση με το AI Chatbot (Flask/Ollama/Krikri)**

Η πιο καινοτόμα εξωτερική επαφή της πλατφόρμας είναι το **AI chatbot**, το οποίο φιλοξενείται σε ανεξάρτητο server (Flask) και παρέχει endpoint στο εξής URL:

https://147.102.74.178:5000/generate

Το Angular application αναλαμβάνει να διαμορφώσει και να στείλει ένα **POST request**, που περιλαμβάνει το audit log σε μορφή φυσικής γλώσσας (π.χ. “ID: 1251, Τροποποιήθηκε από: 660074100...”) καθώς και το prompt του χρήστη (π.χ. “Πες μου για αυτή την καταχώρηση”). Το μοντέλο που ενεργοποιείται στην άλλη πλευρά είναι το:

"model": "pothen-assistant:latest"

Το backend του chatbot είναι υλοποιημένο με το **Krikri** framework, πάνω σε **Ollama** (τοπικό LLM runtime), το οποίο επιτρέπει την αξιολόγηση prompts και την παραγωγή απάντησης. Ο server αναλύει την είσοδο, κάνει reasoning και επιστρέφει δομημένη απάντηση JSON, με πλήρη ανάλυση, στατιστικά και φυσικό κείμενο.

Παράδειγμα πλήρους ροής

Ο admin χρήστης βλέπει την ενότητα "Ιστορικό Δηλώσεων".

Το Angular καλεί το API /api/submission-audits για να πάρει audit δεδομένα.

Τα δεδομένα μορφοποιούνται σε “Audit File” τύπου plain-text prompt.

Το Angular αποστέλλει αυτά τα δεδομένα στο Flask API (/generate) με POST request:

{

"model": "pothen-assistant:latest",

"prompt": "πες μου για αυτη την καταχωρηση\n\nΑρχείο Καταγραφής:\nID: 1251...",

"stream": false

}

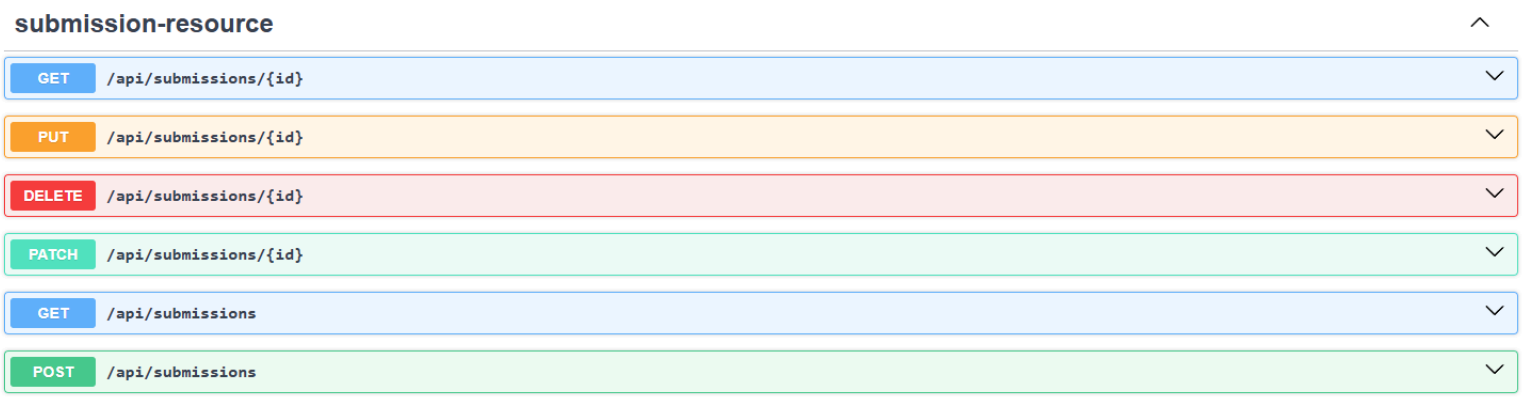
Το chatbot απαντά με δομημένο JSON που περιλαμβάνει:

- Ανάλυση του audit
- Αριθμό τροποποιήσεων
- Ταυτότητα χρηστών
- Ημερομηνίες

Το Angular app εμφανίζει την απάντηση σε απλή, ανθρώπινη μορφή στο UI.

Αυτή η ροή αποδεικνύει την αποδοτική συνεργασία ανεξάρτητων συστημάτων: το backend για ανάκτηση των δεδομένων, το frontend για μετασχηματισμό και routing, και το AI layer για επεξεργασία φυσικής γλώσσας και επεξήγηση των πληροφοριών με "ανθρώπινο" τρόπο.

Παρακάτω παρατίθενται αναλυτικά οι πίνακες των api μέσα από το swagger της εφαρμογής του JHipster.



submission-audit-resource

- GET /api/submission-audits/{id}
- PUT /api/submission-audits/{id}
- DELETE /api/submission-audits/{id}
- PATCH /api/submission-audits/{id}
- GET /api/submission-audits
- POST /api/submission-audits
- GET /api/submission-audits/by-submission/{submissionId}

position-resource

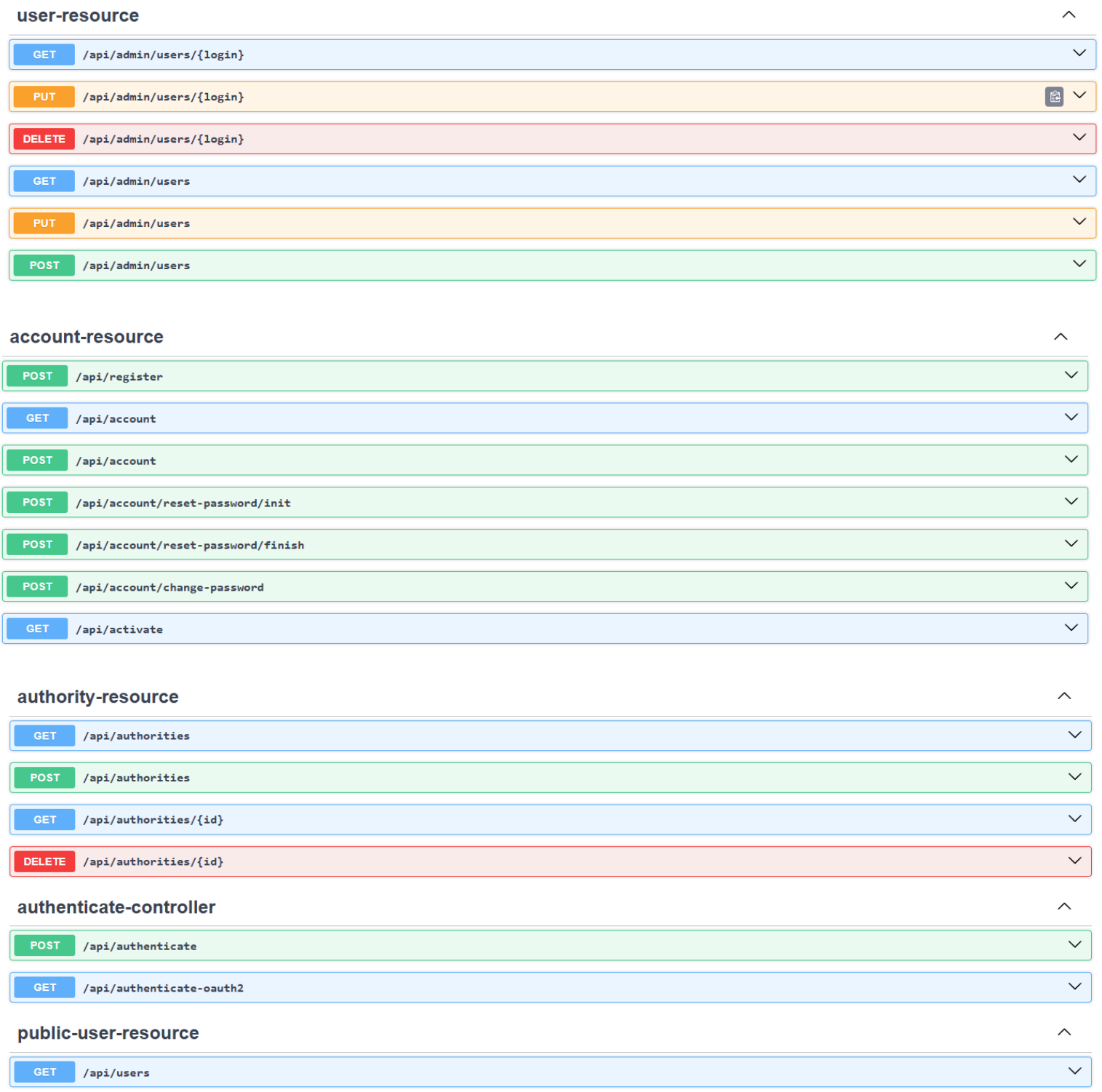
- GET /api/positions/{id}
- PUT /api/positions/{id}
- DELETE /api/positions/{id}
- PATCH /api/positions/{id}
- GET /api/positions
- POST /api/positions

grade-resource

- GET /api/grades/{id}
- PUT /api/grades/{id}
- DELETE /api/grades/{id}
- PATCH /api/grades/{id}
- GET /api/grades
- POST /api/grades

committee-resource

- GET /api/committees/{id}
- PUT /api/committees/{id}
- DELETE /api/committees/{id}
- PATCH /api/committees/{id}
- GET /api/committees
- POST /api/committees



Αναλυτικά το API της εφαρμογής στο APPENDIX A

4.5 Διαδικασία δημιουργίας Chatbot

Το chatbot το οποίο αναπτύχθηκε για χάρη της εργασίας αναπτύχθηκε με το ollama framework το οποίο ουσιαστικά δημιουργεί έναν server στον οποίο τρέχουν κάποια LLMs. Πιο συγκεκριμένα σε ένα μηχάνημα με λειτουργικό σύστημα τύπου Debian τρέχουμε την εντολή

apt install ollama

για να κάνουμε εγκατάσταση του πακέτου. Κατόπιν τρέχοντας

ollama serve

σηκώνεται και είναι up and running ο ollama server. Στη συνέχεια μπορούμε να κατεβάσουμε οποιοδήποτε μοντέλο που παρέχεται από το framework και να το χρησιμοποιήσουμε για ιδία χρήση.

Εμείς επειδή η εφαρμογή μας απευθύνεται σε ελληνικό κοινό και πόσο μάλλον σε πολίτες που θέλουν να χρησιμοποιήσουν πρακτικά μια κρατική εφαρμογή επιλέξαμε το **Krikri LLM** το οποίο είναι ένα pre trained μοντέλο που υποστηρίζει την ελληνική γλώσσα. Τρέχοντας λοιπόν την εντολή

ollama pull ilsp/llama-krikri-8b-instruct:latest

κάνουμε εγκατάσταση του παραπάνω μοντέλου. Τώρα το μοντέλο μας είναι διαθέσιμο και έτοιμο για χρήση. Επειδή όμως εμείς το χρειαζόμαστε για μια συγκεκριμένη εργασία έχουμε τη δυνατότητα να του δώσουμε ένα system prompt, ένα κείμενο δηλαδή το οποίο θα του λέει πρακτικά ποιος είναι ο σκοπός του και με ποιον τρόπο θα πρέπει να απαντάει στα διάφορα ερωτήματα που του θέτουν οι χρήστες. Έτσι δημιουργήσαμε ένα Modelfile και με τη εντολή

ollama create pothen-assistant -f Modelfile

δημιουργούμε ένα ειδικό version του Krikri LLM με όνομα pothen-assistant το οποίο πλέον χρησιμοποιεί την εκπαίδευση του πρώτου αλλά διατηρεί τα δικά του χαρακτηριστικά ως προς τον τρόπο με τον οποίο απαντάει.

Παρακάτω παρατίθεται το Modelfile με το οποίο τροφοδοτείται το LLM για να ΄΄γνωρίζει΄΄ πώς να απαντάει σαν ψηφιακός βοηθός ΠΟΘΕΝ

```
FROM ilsp/llama-krikri-8b-instruct:latest

SYSTEM "Είσαι ένας ψηφιακός βοηθός που αναλύει αρχεία καταγραφής τροποποιήσεων.

Οδηγίες:
- Τα παρακάτω δεδομένα είναι ένα αρχείο καταγραφής τροποποιήσεων υποβολών.
- Κάθε εγγραφή περιέχει ένα ID, τον χρήστη που έκανε την τροποποίηση και την ημερομηνία τροποποίησης.
- Η αποστολή σου είναι να συνοψίσεις το αρχείο καταγραφής αναφέροντας πόσες τροποποιήσεις έγιναν και ποιοι ήταν οι μοναδικοί χρήστες που τις πραγματοποίησαν.

Παράδειγμα Αρχείου Καταγραφής:
ID: 1251, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19
ID: 1252, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19
ID: 1253, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19

Αναμενόμενο Αποτέλεσμα:
Να κάνεις μία περίληψη του αρχείου καταγραφής, αναφέροντας το σύνολο των τροποποιήσεων και τους μοναδικούς χρήστες."

```

Modelfile για τη δημιουργία του pothen-assistant chatbot

Τώρα έχουμε ένα LLM με συγκεκριμένο προσανατολισμό για να μας βοηθήσει σε αυτό που θέλουμε να κάνουμε. Αυτό που μένει είναι να φτιάξουμε την υποδομή ώστε το μοντέλο αυτό να γίνει προσβάσιμο μέσα από την κύρια εφαρμογή μας. Έτσι χρησιμοποιώντας τη γλώσσα python και κάποιες βιβλιοθήκες όπως flask, flask-cors και το εργαλείο gunicorn φτιάξαμε έναν server ο οποίος μπορεί να γίνει accessed από το Angular frontend μας. Περισσότερα για τον τρόπο αλληλεπίδρασης των δύο αυτών components θα αναλυθούν στο κεφάλαιο 5. Παρακάτω παρατίθεται το κομμάτι κώδικα που υλοποιεί όλα τα παραπάνω. [13][14][15]

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
import requests

app = Flask(__name__)

# Allow only requests from your domain
CORS(app, resources={r"/generate": {"origins": "https://147.102.74.34"}})

OLLAMA_URL = "http://localhost:11434/api/generate"


@app.route("/generate", methods=["POST"])
def generate():
    data = request.json
    user_prompt = data.get("prompt", "")


    request_body = {
        "model": data.get("model", "pothen-assistant:latest"),
        "prompt": user_prompt,
        "stream": False
    }

    response = requests.post(OLLAMA_URL, json=request_body)

    return jsonify(response.json())

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context=("cert.pem", "key.pem"))
```

Κώδικας chatbot server

4.6 Ροή ερωτήματος από το Chatbot και επιστροφή απάντησης

Η αλληλεπίδραση χρήστη–chatbot ακολουθεί τα εξής βήματα:

1. Ο χρήστης πληκτρολογεί ένα ερώτημα, π.χ. «Τι αλλαγές έγιναν στις δηλώσεις από 1 έως 15 Φεβρουαρίου;».
2. Το frontend στέλνει POST request στο API του chatbot με το ερώτημα σε JSON.

3. Το Python backend λαμβάνει το ερώτημα, εντοπίζει το intent, συνθέτει query (είτε SQL είτε φίλτρο) και ανακτά τα σχετικά audit entries.
4. Το αποτέλεσμα «μεταφράζεται» σε φυσική γλώσσα και αποστέλλεται πίσω στον χρήστη.

Η ικανότητα αυτή προσφέρει κάτι εξαιρετικά καινοτόμο: τη δυνατότητα **αντικατάστασης της κλασικής SQL με φυσική γλώσσα**, καθιστώντας το σύστημα προσβάσιμο ακόμη και σε χρήστες χωρίς τεχνικό υπόβαθρο. Η λειτουργία αυτή λειτουργεί ενισχυτικά στον ρόλο του admin, ο οποίος μπορεί να εντοπίζει γρήγορα πότε έγιναν αλλαγές και από ποιον, χωρίς να χρειάζεται γνώση δομής της βάσης ή συντακτικό SQL.

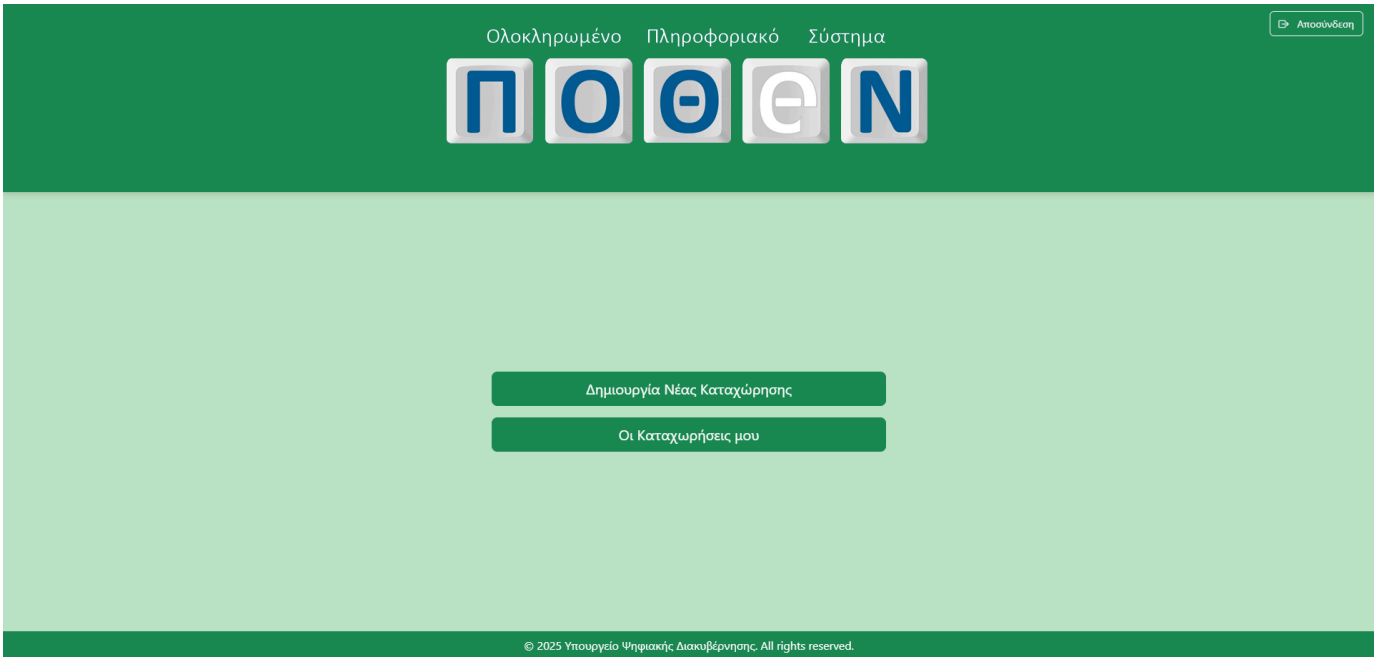
Αναλυτικά παρατίθεται ο κώδικας στο APPENDIX C

ΚΕΦΑΛΑΙΟ 5. ΧΡΗΣΗ ΠΕΡΙΠΤΩΣΕΩΝ

Η αξιολόγηση μιας σύγχρονης διαδικτυακής πλατφόρμας, ιδιαίτερα όταν αυτή διαχειρίζεται ευαίσθητα και θεσμικά κατοχυρωμένα δεδομένα όπως οι δηλώσεις Πόθεν Έσχες, δεν μπορεί να περιοριστεί σε τεχνικά χαρακτηριστικά ή αρχιτεκτονικά πλεονεκτήματα. Η ουσιαστική μέτρηση της αξίας της εφαρμογής αποτυπώνεται στην καθημερινή χρήση από τους ρόλους που την απαρτίζουν – δηλαδή στους τελικούς χρήστες και στους διαχειριστές – και στο κατά πόσο αυτή η χρήση διευκολύνεται, αυτοματοποιείται ή ενισχύεται από τις παρεχόμενες λειτουργίες. Σε αυτό το κεφάλαιο, παρουσιάζονται αναλυτικά σενάρια χρήσης (use cases), οργανωμένα ανάλογα με το ρόλο (user/admin), τη φύση της αλληλεπίδρασης και την τεχνολογική αξιοποίηση. Αυτά τα σενάρια βασίζονται σε ρεαλιστικές υποθέσεις λειτουργίας και αποσκοπούν στο να αξιολογηθεί η χρηστικότητα, η προσβασιμότητα και η αποδοτικότητα του συστήματος μέσα από πραγματικά παραδείγματα.

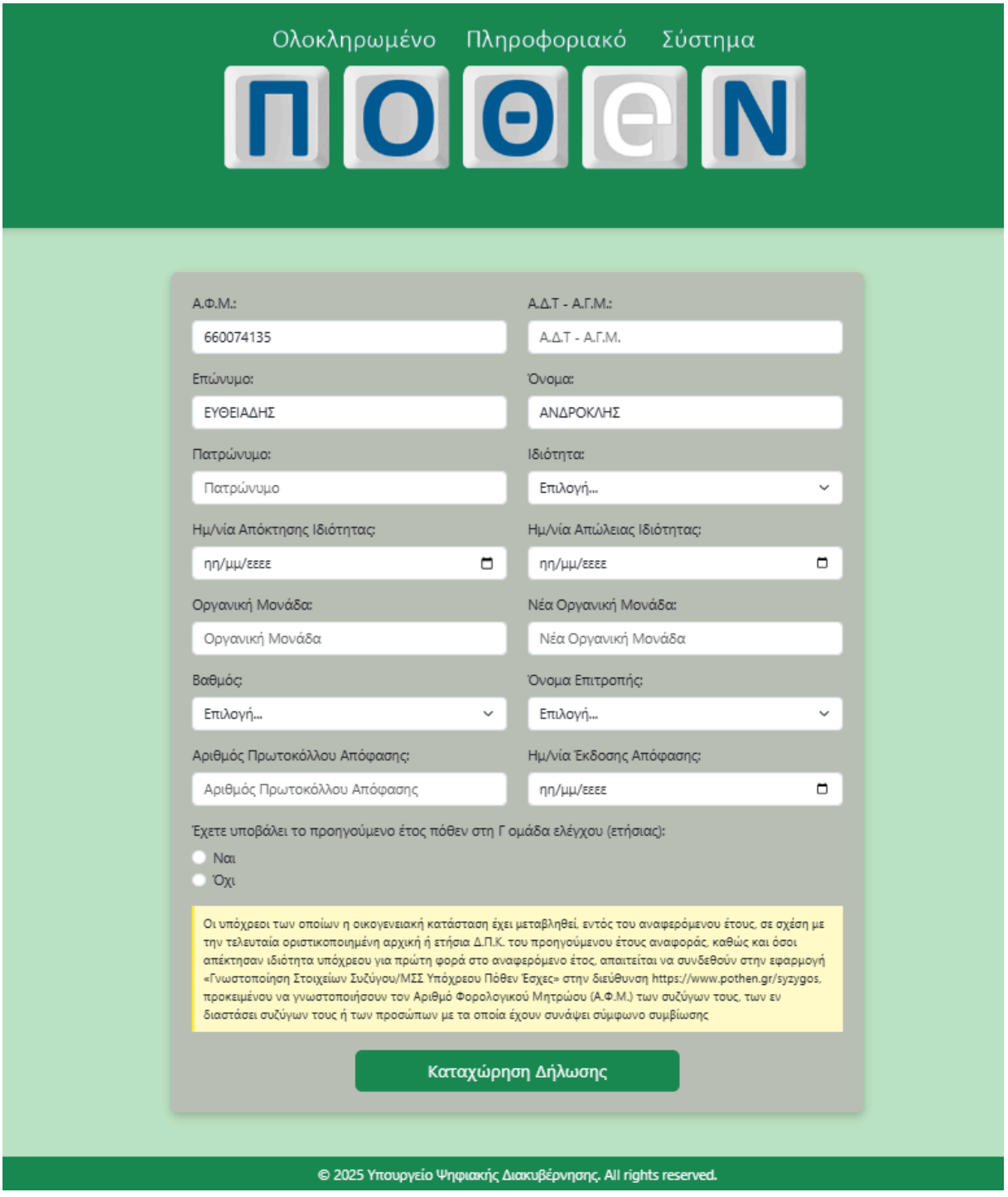
5.1 Ρόλος Χρήστη – Υποβολή νέας δήλωσης

Ο χρήστης εισερχόμενος στην εφαρμογή, επιλέγει να κάνει σύνδεση (login) με διαπιστευτήρια taxisnet και όπως αναλύθηκε στο κεφάλαιο 4, μεταπηδά στο ασφαλές περιβάλλον του υπουργείο. Εκεί εισάγει τα διαπιστευτήριά του και εφόσον είναι σωστά το σύστημα κάνει ανακατεύθυνση στην εφαρμογή και πάλι στην κεντρική σελίδα του χρήστη όπως φαίνεται παρακάτω.

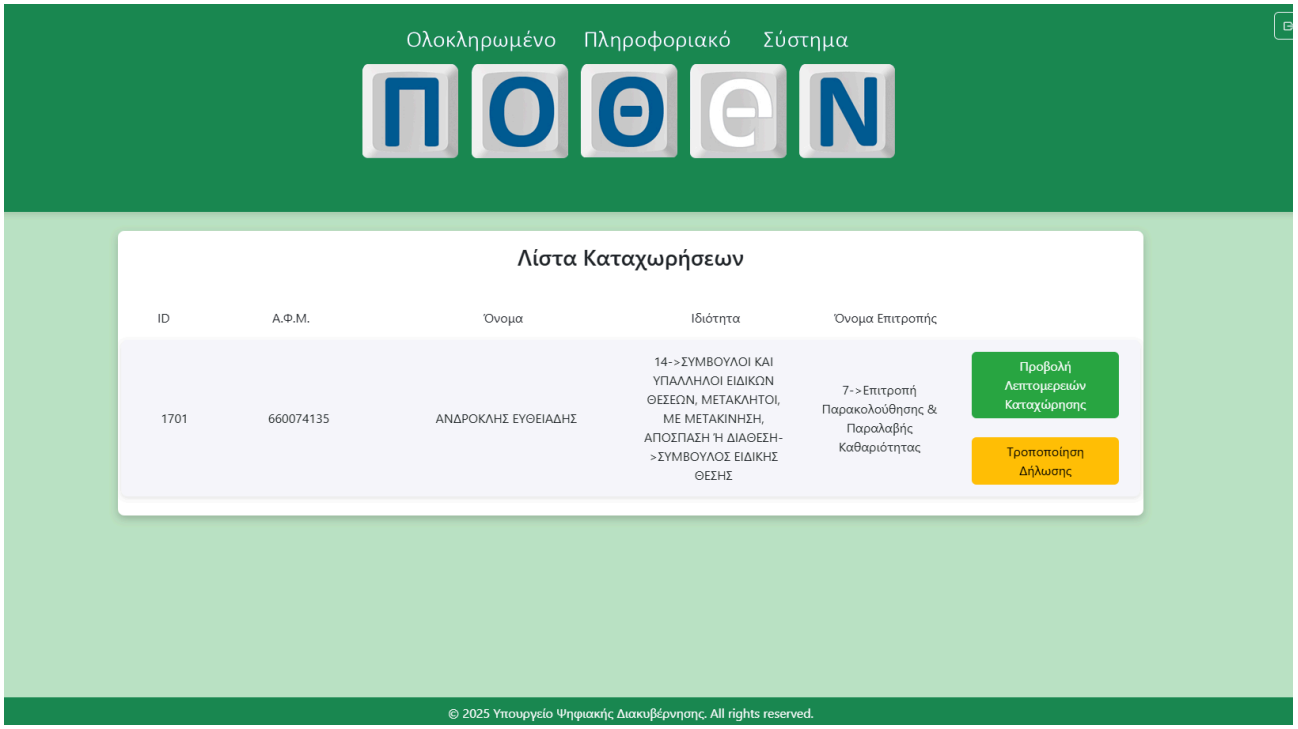


Κεντρική σελίδα για το ρόλο απλού χρήστη.

Στη συνέχεια κάνοντας κλικ στο button ΄΄Δημιουργία νέας καταχώρησης΄΄ μεταφέρεται στη φόρμα όπου πρέπει να συμπληρώσει τα στοιχεία του όπως φαίνεται παρακάτω. Να σημειώσουμε εδώ ότι τα πεδία Α.Φ.Μ. , Επώνυμο και Όνομα είναι προσυμπληρωμένα και αυτό συμβαίνει διότι αφού ο χρήστης κάνει login μέσω του υπουργείου, παίρνουμε μέσω ενός web service τα στοιχεία αυτά και τα προβάλλουμε στη φόρμα. Αφού ο χρήστη συμπληρώσει όλα τα πεδία μπορεί να πατήσει το button ΄΄Καταχώρηση Δήλωσης΄΄ και εφόσον όλα πήγαν καλά θα δει ένα alert ψηλά στο κέντρο της οθόνης ότι η δήλωσή του καταχωρήθηκε επιτυχώς. Στη συνέχεια επιστρέφοντας στο κεντρικό μενού και επιλέγοντας ΄΄Οι καταχωρήσεις μου θα είναι σε θέση να δει τη δήλωση την οποία έκανε και να προβεί αν θέλει σε κάποια τροποποίηση. Και για αυτό παραθέτουμε μια αντίστοιχη εικόνα παρακάτω.



Φόρμα Υποβολής Καταχώρησης



Λίστα Καταχωρήσεων Χρήστη

Όλα τα παραπάνω υλοποιούνται με κατάλληλα API calls από το angular frontend προς το JHipster API. Πιο συγκεκριμένα για την καταχώρηση νέας δήλωσης στέλνουμε ένα POST request στο endpoint /api/submissions με request body το json της ακόλουθης μορφής

```
{
  "id": 0,
  "afm": "328997002",
  "adt": "string",
  "lastName": "string",
  "firstName": "string",
  "fatherName": "string",
  "acquisitionDate": "2025-04-06",
  "lossDate": "2025-04-06",
  "organizationUnit": "string",
  "newOrganizationUnit": "string",
  "protocolNumber": "string",
  "decisionDate": "2025-04-06",
  "previousSubmission": true,
  "position": {
    "id": 0,
    "name": "string"
  },
  "grade": {
    "id": 0,
    "name": "string"
  },
  "committeeName": {
    "id": 0,
    "name": "string"
  },
  "user": {
    "id": 0,
    "login": "string"
  }
}
```

JSON Object for creating submissions

Για να πάρουμε τις δηλώσεις του χρήστη στέλνουμε ένα GET request στο endpoint /api/submissions. Αξίζει να σημειωθεί ότι το συγκεκριμένο endpoint επειδή γίνεται generate αυτόματα από το JHipster επιστρέφει εξ ορισμού όλες τις δηλώσεις όλων των χρηστών. Εμείς όμως θέλουμε τις δηλώσεις μόνο του παρόντος χρήστη που είναι logged in. Έτσι έπρεπε να επέμβουμε στον generated κώδικα του Controller και να τροποποιήσουμε την υλοποίηση του συγκεκριμένου endpoint ώστε να φιλτράρει τις δηλώσεις με βάση το login username του χρήστη που θέλουμε. Παρακάτω παρατίθεται ο κώδικας που είναι υπεύθυνος για τη διαδικασία αυτή.

```
150     @GetMapping("")
151     public ResponseEntity<List<SubmissionDTO>> getAllSubmissions(@org.springdoc.core.annotations.ParameterObject Pageable pageable) {
152         LOG.debug("REST request to get a page of Submissions");
153 
154         // Get the authentication details
155         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
156         String login = authentication.getName();
157 
158         // Check if the user has the ADMIN role
159         boolean isAdmin = authentication
160             .getAuthorities()
161             .stream()
162             .anyMatch(grantedAuthority -> grantedAuthority.getAuthority().equals("ROLE_ADMIN"));
163 
164         Page<SubmissionDTO> page;
165         if (isAdmin) {
166             // Admin can view all submissions
167             page = submissionService.findAll(pageable);
168         } else {
169             // Regular users can only view their own submissions
170             page = submissionService.findAllByUser(pageable, login);
171         }
172 
173         HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(ServletUriComponentsBuilder.fromCurrentRequest(), page);
174         return ResponseEntity.ok().headers(headers).body(page.getContent());
175     }
```

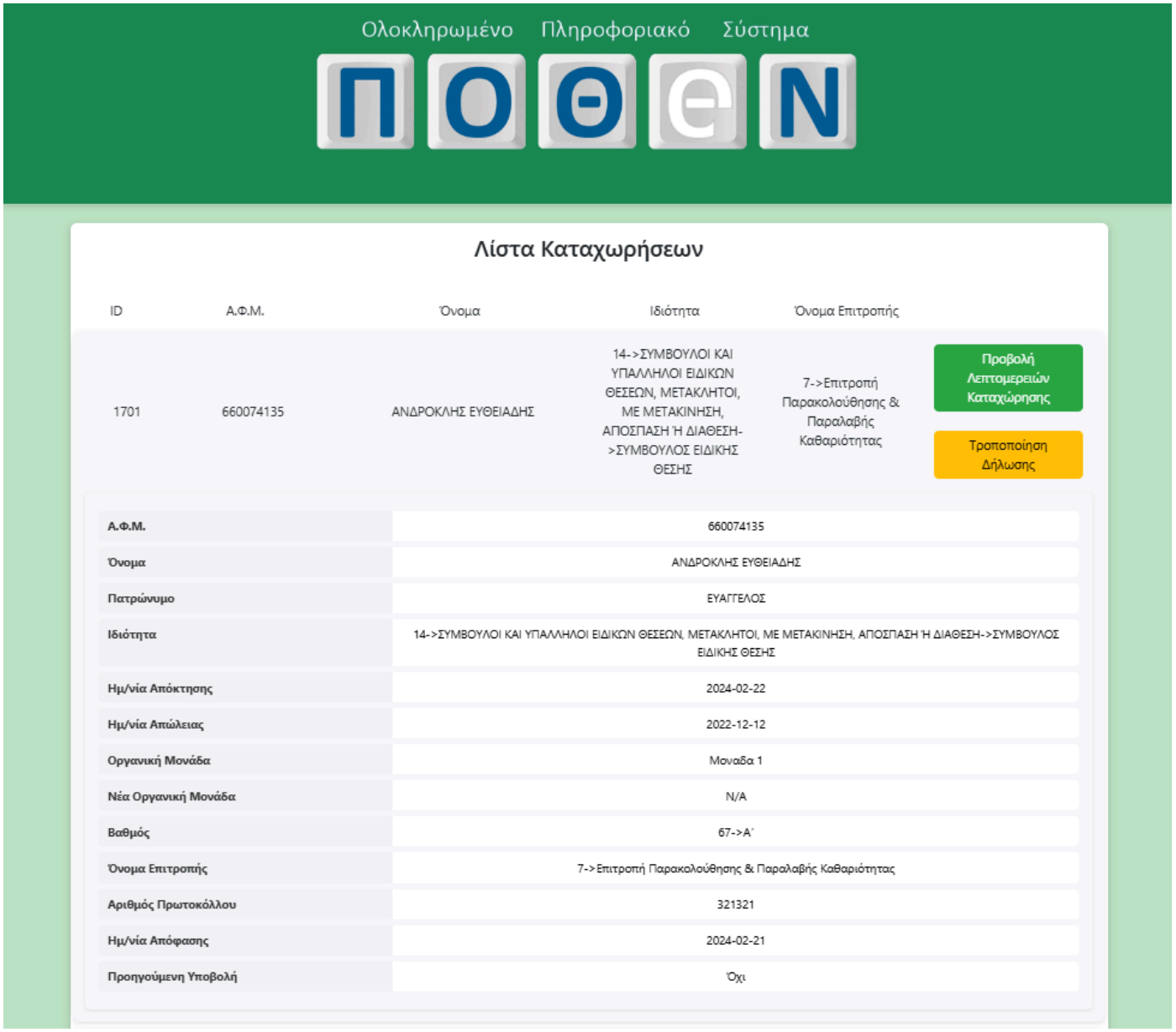
Εύκολα μπορεί να παρατηρήσει κανείς από το παραπάνω κομμάτι κώδικα ότι αν ο χρήστης που εκτελεί το GET request προς το /api/submissions/ έχει ρόλο admin, τότε και μόνο τότε μπορεί να δει όλες τις καταχωρήσεις των χρηστών. Ειδάλλως επιστρέφονται μόνο οι καταχωρήσεις του χρήστη με το τρέχον login username όπως φαίνεται στις γραμμές 167 και 170 του παραπάνω κομματιού κώδικα.

5.2 Ρόλος Χρήστη – Τροποποίηση υπάρχουσας δήλωσης

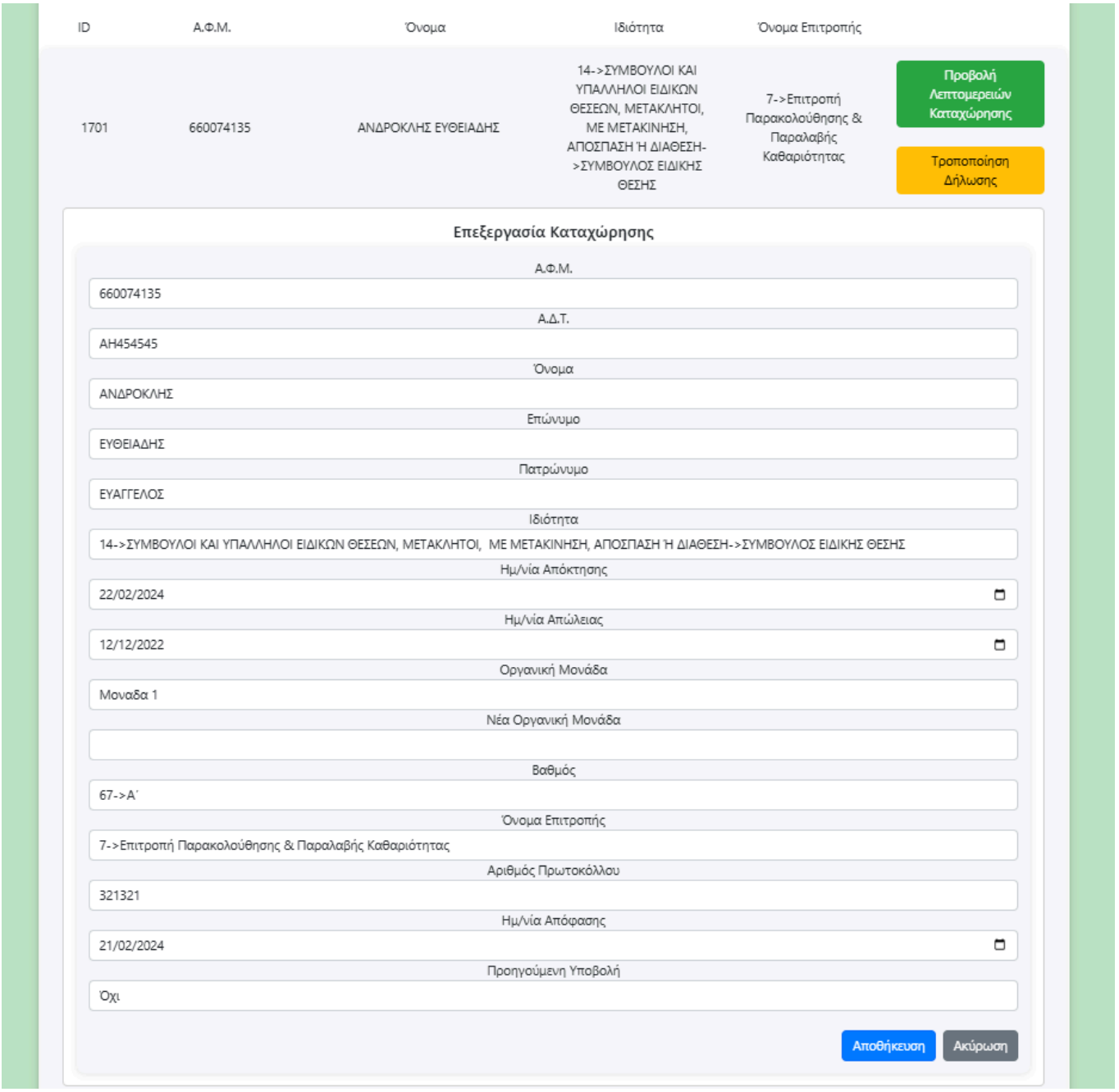
Το δεύτερο Use Case που αφορά τον απλό χρήστη είναι αυτό της τροποποίησης κάποιας δήλωσης που έχει ήδη υποβάλλει. Με τον ίδιο τρόπο με το προηγούμενο Use Case που περιγράψαμε, ο χρήστης συνδέεται μέσω διαπιστευτηρίων taxisnet και αφού εισέλθει στην εφαρμογή, αυτή τη φορά επιλέγει από το κεντρικό μενού το button ΄΄Οι Καταχωρήσεις μου΄΄ όπως δείξαμε παραπάνω. Στην καινούρια σελίδα αυτή μπορεί είτε να δει τα αναλυτικά στοιχεία των καταχωρήσεών του κάνοντας κλικ στο button ΄΄Προβολή Λεπτομερειών Καταχώρησης΄΄ -αυτό το κουμπί ανοίγει με dropdown τρόπο έναν πίνακα όπου φαίνονται αναλυτικά τα στοιχεία της καταχώρησης που επέλξε ο χρήστης να δει- είτε να τροποποιήσει κάποια από τις

δηλώσεις του κάνοντας κλικ στο button ΄΄Τροποποίηση Δήλωσης΄΄. Το τελευταίο ανοίγει μια φόρμα με προσυμπληρωμένα όλα τα στοιχεία της δήλωσης αλλά με τη δυνατότητα επεξεργασίας τους από το χρήστη. Ο χρήστης κάνει τις αλλαγές που θέλει και πατάει το κουμπί ΄΄Αποθήκευση΄΄ και αν όλα έχουν υποβληθεί σωστά η δήλωση ενημερώνεται. Παρατίθενται εικόνες με το συγκεκριμένο Use Case.

Λίστα Καταχωρήσεων Χρήστη



Προβολή Λεπτομερειών Καταχώρησης



Τροποποίηση Δήλωσης

Για την υλοποίηση του παρόντος Use Case χρησιμοποιείται και πάλι το endpoint /api/submissions στο οποίο αυτή τη φορά στέλνουμε ένα PUT request και περνάμε τις αλλαγές στο entity Submission όπως φαίνεται παρακάτω.

```
21    // Method to update an existing submission
22    updateSubmission(submissionId: number, updatedSubmission: any): Observable<any> {
23      const headers = this.getAuthHeaders();
24      return this.http.put<any>(`${this.apiUrl}/${submissionId}`, updatedSubmission, { headers });
25    }
```

Κώδικας για το request τροποποίησης δήλωσης από πλευράς frontend.

Ακόμη κρατάμε ένα αρχείο των τροποποιήσεων όλων των δηλώσεων στο entity που έχουμε δημιουργήσει με όνομα Submission-audit. Έτσι το παραπάνω call κάνει trigger τη δημιουργία μιας εγγραφής στον πίνακα των audits. Και σε αυτή την περίπτωση τροποποιήσαμε τον auto-generated κώδικα που δημιουργεί το JHipster ώστε να πετύχουμε το επιθυμητό αποτέλεσμα. Παρατίθεται το κομμάτι κώδικα υπεύθυνο για την παραπάνω διαδικασία.

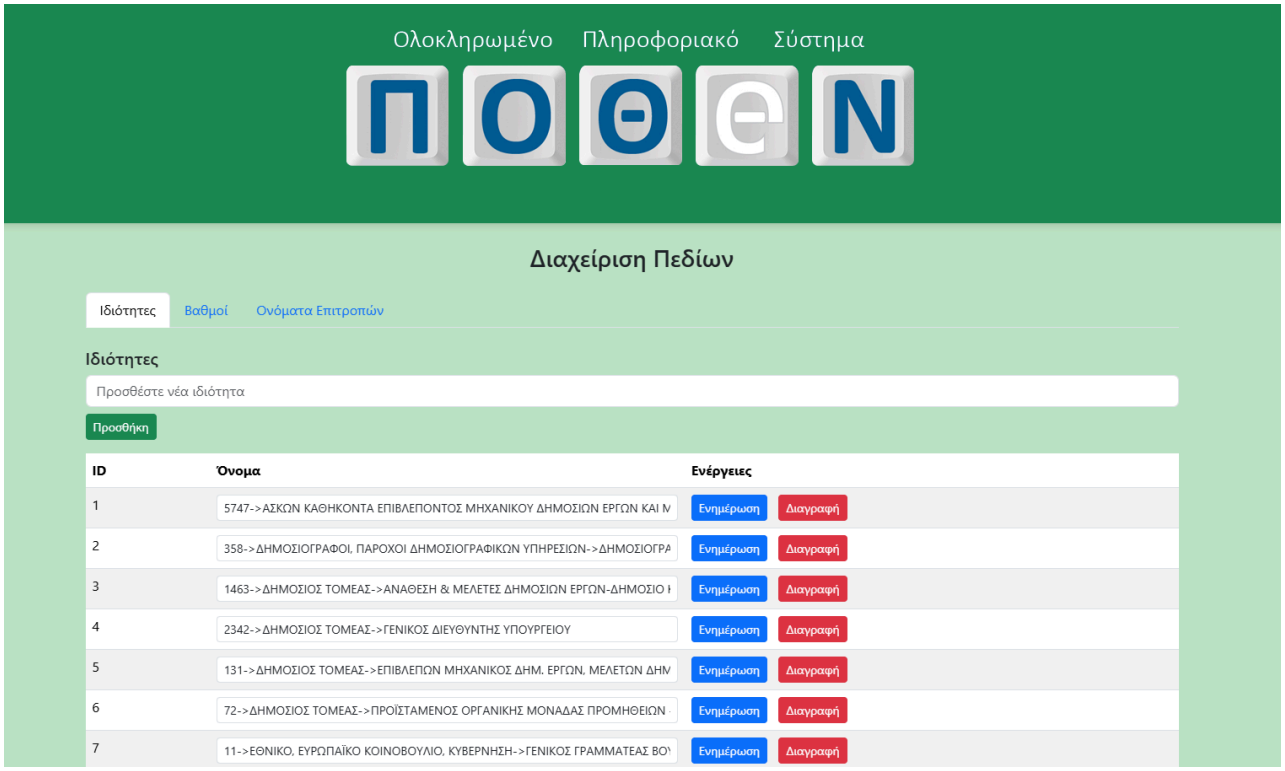
```
public SubmissionDTO update(SubmissionDTO submissionDTO) {
    LOG.debug("Request to update Submission : {}", submissionDTO);

    // Fetch the existing Submission from the database
    Optional<Submission> existingSubmission = submissionRepository.findById(submissionDTO.getId());

    Submission submission = submissionMapper.toEntity(submissionDTO);
    submission = submissionRepository.save(submission);

    // If an existing record was found, create an audit entry
    existingSubmission.ifPresent(oldSubmission -> {
        SubmissionAudit audit = new SubmissionAudit();
        audit.setAfm(oldSubmission.getAfm());
        audit.setAdt(oldSubmission.getAdt());
        audit.setLastName(oldSubmission.getLastName());
        audit.setFirstName(oldSubmission.getFirstName());
        audit.setFatherName(oldSubmission.getFatherName());
        audit.setAcquisitionDate(oldSubmission.getAcquisitionDate());
        audit.setLossDate(oldSubmission.getLossDate());
        audit.setOrganizationUnit(oldSubmission.getOrganizationUnit());
        audit.setNewOrganizationUnit(oldSubmission.getNewOrganizationUnit());
        audit.setProtocolNumber(oldSubmission.getProtocolNumber());
        audit.setDecisionDate(oldSubmission.getDecisionDate());
        audit.setPreviousSubmission(oldSubmission.getPreviousSubmission());
        audit.setModifiedDate(Instant.now());
        audit.setModifiedBy(SecurityUtils.getCurrentUserLogin().orElse("unknown"));
        audit.setChangeType("UPDATE");
        audit.setOriginalSubmission(oldSubmission);

        submissionAuditRepository.save(audit);
    });

    return submissionMapper.toDto(submission);
```

Κώδικας για την τροποποίηση δήλωσης από πλευράς backend.

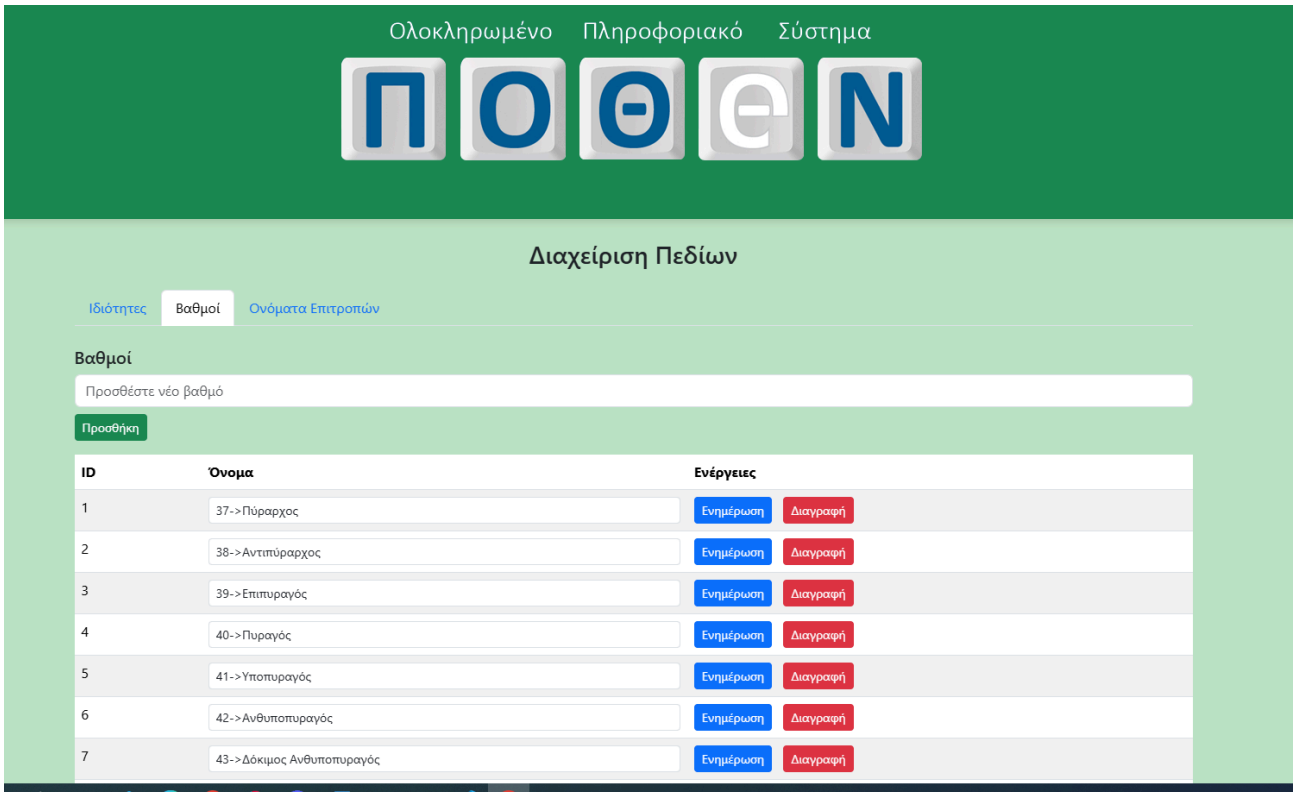
5.3 Ρόλος Admin – Εποπτεία, τροποποίηση πεδίων και έλεγχος δηλώσεων

Η ευθύνη του διαχειριστή (admin) περιλαμβάνει όχι μόνο την επισκόπηση των δηλώσεων, αλλά και τη διατήρηση της ομοιογένειας, της ακρίβειας και της τυποποίησης των δεδομένων που εισάγονται. Μέσα από το διαχειριστικό dashboard, ο admin έχει πρόσβαση σε όλες τις δηλώσεις χρηστών. Ένα από τα σημαντικότερα use cases είναι η παρέμβαση του admin στα μετα-δεδομένα μιας δήλωσης – κυρίως για τη διόρθωση τιμών σε πεδία που συχνά καταχωρούνται με λάθη ή ποικιλομορφία (π.χ. διαφορετική ορολογία για την ίδια επιτροπή).

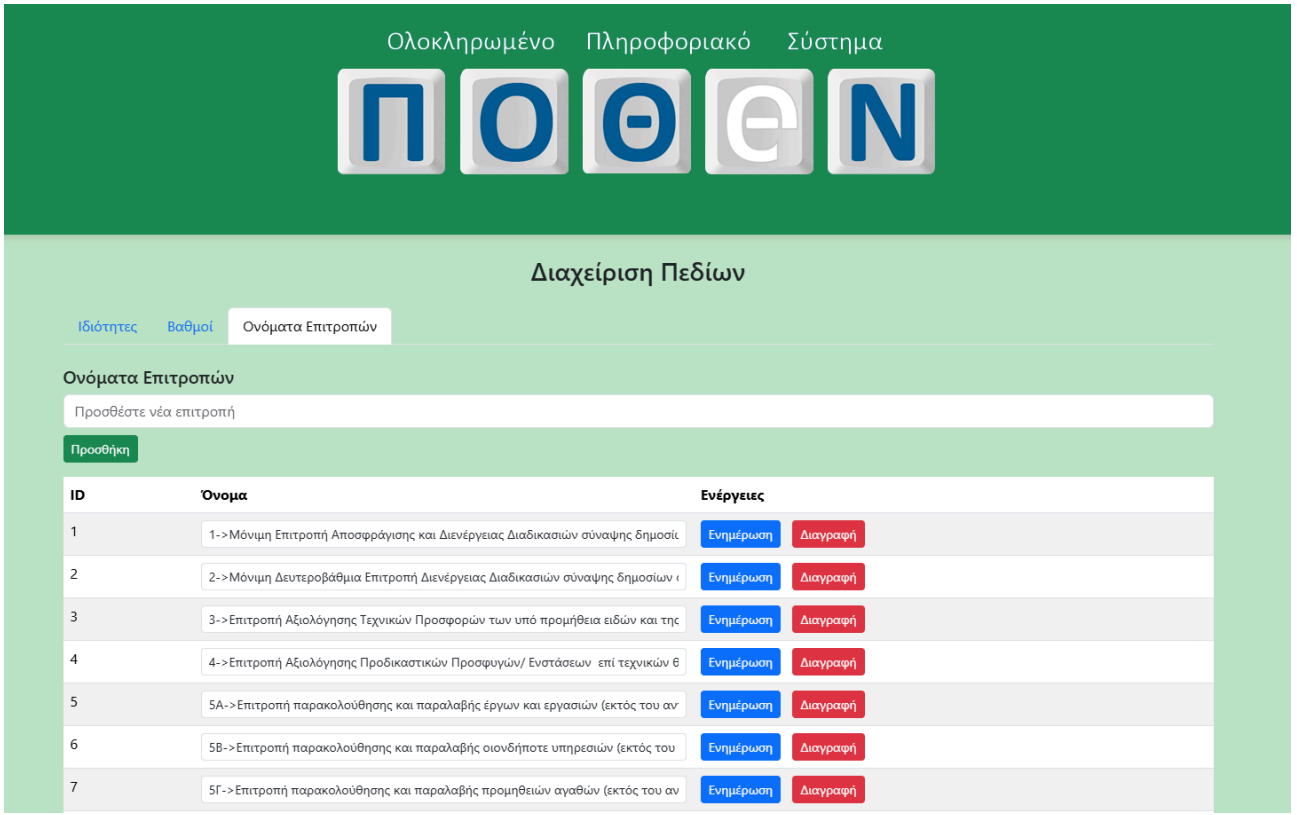
Για παράδειγμα, ο admin ενδέχεται να παρατηρήσει ότι πολλοί χρήστες έχουν καταχωρήσει με παραλλαγές τον τίτλο «Μέλος Επιτροπής Διαφάνειας». Μέσα από τη διεπαφή επεξεργασίας, έχει τη δυνατότητα να ενοποιήσει τις τιμές σε μία προκαθορισμένη επιλογή, χωρίς να χρειάζεται να τροποποιήσει όλη τη δήλωση. Η αλλαγή αυτή καταγράφεται επίσης στο audit log, μαζί με την πληροφορία ότι έγινε από admin, με διαχωρισμό από τις ενέργειες του τελικού χρήστη. Επιπλέον, ο admin έχει πρόσβαση σε dashboards στατιστικής απεικόνισης, ώστε να ελέγχει τη ροή των δηλώσεων ανά χρονική περίοδο, να εντοπίζει ανωμαλίες ή να ετοιμάζει reports για εποπτικούς φορείς. Παρακάτω δίνονται οι εικόνες της εφαρμογής όπου σε κάθε περίπτωση ο admin μπορεί είτε να προσθέσει είτε να τροποποιήσει είτε να διαγράψει το αντίστοιχο metadata.



Διαχείριση Metadata – Ιδιότητες



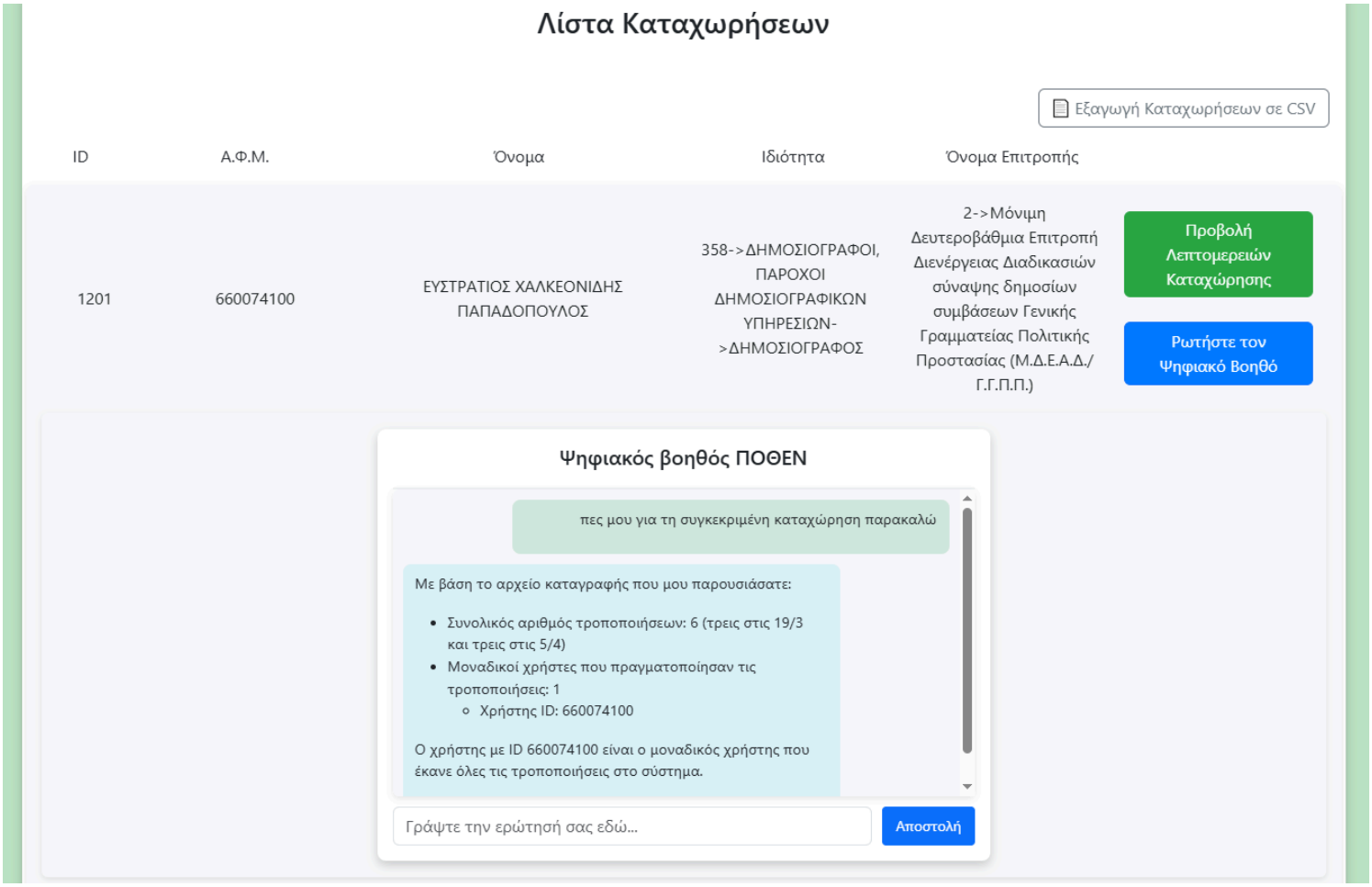
Διαχείριση Metadata – Βαθμοί



Διαχείριση Metadata - Επιτροπές

5.4 Ρόλος Admin – Αλληλεπίδραση με το Chatbot για ιστορικό αλλαγών

Ένα από τα πιο καινοτόμα use cases της πλατφόρμας αφορά την αξιοποίηση του ενσωματωμένου AI Chatbot από τον διαχειριστή. Το chatbot λειτουργεί ως εργαλείο φυσικής γλώσσας για την ανάκτηση πληροφορίας από τα submission-audits, επιτρέποντας στον admin να εκτελεί σύνθετες αναζητήσεις χωρίς να χρειάζεται να γνωρίζει τη δομή της βάσης ή να χρησιμοποιεί SQL. Παρακάτω παρατίθεται ένα παράδειγμα αλληλεπίδρασης με το συγκεκριμένο bot.



Παράδειγμα αλληλεπίδρασης με το AI Chatbot

Όπως παρατηρούμε στο παραπάνω παράδειγμα ο χρήστης (admin) κλικάροντας στο button ΄΄Ρωτήστε τον ψηφιακό βοηθό του ανοίγει ένα παράθυρο τύπο chat στο οποίο μπορεί να αλληλεπιδράσει με τον ΄΄Ψηφιακό Βοηθό ΠΟΘΕΝ΄΄. Γράφοντας λοιπόν ο

χρήστης ΄΄Πες μου για τη συγκεκριμένη καταχώρηση΄΄ και πατώντας αποστολή το chatbot του απαντά το παρακάτω κείμενο.

΄΄Με βάση το αρχείο καταγραφής που μου παρουσιάσατε:

- Συνολικός αριθμός τροποποιήσεων: 6 (τρεις στις 19/3 και τρεις στις 5/4)
- Μοναδικοί χρήστες που πραγματοποίησαν τις τροποποιήσεις: 1
    - Χρήστης ID: 660074100

Ο χρήστης με ID 660074100 είναι ο μοναδικός χρήστης που έκανε όλες τις τροποποιήσεις στο σύστημα.΄΄

Το chatbot προγραμματιστεί κατάλληλα ώστε να απαντάει με συγκεκριμένο τρόπο όπως περιγράφηκε στο κεφάλαιο 4. Όταν ο χρήστης κλικάρει αποστολή ουσιαστικά συμβαίνει το εξής: Στέλνουμε στο server του chatbot (ο οποίος τρέχει ένα KriKri LLM) το μήνυμα του χρήστη μαζί με ένα αρχείο καταγραφής του ιστορικού των καταχωρήσεων (του entity submission-audits) το οποίο έχουμε πάρει από προηγούμενο http request στο jhipster backend.

Οπότε το τελικό μήνυμα που φτάνει στο chatbot για το παράδειγμά μας είναι το παρακάτω:

΄΄πες μου για τη συγκεκριμένη καταχώρηση παρακαλώ

Αρχείο Καταγραφής:

ID: 1251, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19

ID: 1252, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19

ID: 1253, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-03-19

ID: 1601, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-04-05

ID: 1602, Τροποποιήθηκε από: 660074100, Ημερομηνία Τροποποίησης: 2025-04-05

΄΄

Άρα ο ΄΄Ψηφιακός Βοηθός ΠΟΘΕΝ΄΄ είναι προσανατολισμένος να μας απαντάει για το ιστορικό της εκάστοτε δήλωσης.

ΚΕΦΑΛΑΙΟ 6. ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Η παρούσα πλατφόρμα αποτελεί ένα σταθερό και επεκτάσιμο θεμέλιο για τον εκσυγχρονισμό της διαδικασίας υποβολής και εποπτείας των δηλώσεων Πόθεν

Έσχες. Παρότι η υλοποίηση καλύπτει τις βασικές ανάγκες διαχείρισης και ελέγχου δηλώσεων, ενσωματώνει σύγχρονες τεχνολογίες και λειτουργεί πλήρως σε ρεαλιστικό περιβάλλον, υπάρχουν σημαντικά περιθώρια εξέλιξης, βελτιστοποίησης και εμπλουτισμού με νέες δυνατότητες. Η ευελιξία της αρχιτεκτονικής και η modular φιλοσοφία τόσο του frontend (Angular) όσο και του backend (JHipster/Spring Boot), επιτρέπουν την ομαλή προσαρμογή της εφαρμογής στις διαρκώς μεταβαλλόμενες ανάγκες του θεσμικού και τεχνολογικού περιβάλλοντος.

6.1 Προτάσεις για βελτιώσεις και επέκταση λειτουργιών,

Μια από τις πιο άμεσες δυνατότητες επέκτασης της πλατφόρμας αφορά την υλοποίηση μηχανισμού αξιολόγησης κινδύνου (risk scoring) σε υποβαλλόμενες δηλώσεις. Βάσει του ιστορικού τροποποιήσεων, του πλήθους των αλλαγών, του τύπου των πεδίων που αλλάζουν, αλλά και της χρονικής συγκέντρωσης επεμβάσεων, μπορούν να υπολογίζονται δυναμικά “δείκτες ανωμαλίας”. Αυτοί οι δείκτες θα μπορούσαν να εμφανίζονται ως rating για κάθε δήλωση, βοηθώντας τον διαχειριστή να δώσει προτεραιότητα σε ελέγχους. Το σύστημα θα μπορούσε να αξιοποιήσει machine learning τεχνικές, ώστε το μοντέλο κινδύνου να εκπαιδεύεται σταδιακά με βάση τα πραγματικά δεδομένα του συστήματος.

Επιπλέον, σημαντική βελτίωση θα μπορούσε να αποτελέσει η εισαγωγή υπογραφής δηλώσεων με ψηφιακό πιστοποιητικό, ώστε η τελική υποβολή να αποκτά και νομική υπόσταση. Η πλατφόρμα θα μπορούσε να ενσωματώσει μηχανισμούς υπογραφής εγγράφων (π.χ. με eIDAS συμβατές υποδομές ή cloud-based signing services), συνδέοντας με ασφάλεια τον τελικό χρήστη με τη δήλωσή του μέσω ενός αναγνωρισμένου αποδεικτικού.

Μια άλλη χρήσιμη επέκταση θα ήταν η εισαγωγή role-based notifications και ειδοποιήσεων: ο χρήστης θα μπορούσε να λαμβάνει ειδοποίηση κάθε φορά που ο admin πραγματοποιεί επέμβαση στη δήλωσή του, ενώ ο admin θα ενημερώνεται αυτόματα για υποβολές που εκκρεμούν για παραπάνω από X ημέρες, ή δηλώσεις που παρουσιάζουν pattern που ταιριάζει με “ύποπτη συμπεριφορά”. Παρόμοια notifications θα μπορούσαν να αξιοποιήσουν τεχνικές real-time event streaming με χρήση Kafka ή WebSockets για ζωντανή ανανέωση του UI.

Τέλος, μια ουσιαστική βελτίωση αφορά την αυτοματοποιημένη εξαγωγή αναφορών προς εξωτερικούς φορείς. Η πλατφόρμα θα μπορούσε να ενσωματώσει export modules για CSV, PDF ή XML μορφοποιήσεις, σύμφωνα με πρότυπα που ορίζονται από εποπτευόμενες αρχές (π.χ. Αρχή Διαφάνειας). Αυτό θα επέτρεπε την εύκολη διαβίβαση συγκεντρωτικών δεδομένων για έλεγχο, διασταύρωση ή αποθήκευση σε συστήματα τρίτων.

6.2 Προσαρμογή της εφαρμογής σε νέες τεχνολογίες και ανάγκες

Η ραγδαία εξέλιξη των τεχνολογιών Τεχνητής Νοημοσύνης και ιδιαίτερα των Large Language Models ανοίγει νέες δυνατότητες για την περαιτέρω "νοημοσύνη" της πλατφόρμας. Ήδη, μέσω του υπάρχοντος chatbot, έχει επιτευχθεί ένας πρώτος βαθμός αλληλεπίδρασης με φυσική γλώσσα. Στο μέλλον, η λειτουργία αυτή μπορεί να εμπλουτιστεί με multilingual υποστήριξη, ώστε η πλατφόρμα να εξυπηρετεί χρήστες άλλων γλωσσικών προφίλ (π.χ. ομογενείς ή Ευρωπαίους αξιωματούχους που υπόκεινται σε έλεγχο), καθιστώντας την εφαρμογή διεθνώς αξιοποιήσιμη.

Επιπλέον, το chatbot μπορεί να επεκταθεί σε παραγωγική υποβοήθηση, παρέχοντας αυτόματες προτάσεις για συμπλήρωση πεδίων βάσει ιστορικού του χρήστη ή ακόμα και με σύνδεση σε δημόσια δεδομένα (π.χ. υπηρεσιακή κατάσταση από Εργάνη ή Μητρώα προσωπικού). Ένα τέτοιο σύστημα θα μπορούσε να προτείνει: «Βάσει της ιδιότητάς σας, ενδέχεται να ανήκετε στην Επιτροπή Χ», ή να εντοπίζει ελλείψεις πριν την υποβολή (π.χ. μη καταχωρισμένο IBAN).

Από πλευράς τεχνικής αρχιτεκτονικής, θα μπορούσε να ενσωματωθεί υποστήριξη για mobile εφαρμογή ή PWA (Progressive Web App), ώστε οι χρήστες να έχουν άμεση πρόσβαση στην πλατφόρμα και από κινητές συσκευές. Επίσης, σε backend επίπεδο, η μετάβαση από monolith σε microservices (με Kubernetes και Service Mesh) θα επέτρεπε την καλύτερη διαχείριση φόρτου και την αποσύνδεση κρίσιμων modules (όπως authentication, audit, chatbot) για ανεξάρτητη ανάπτυξη και scaling.

Τέλος, υπάρχει προοπτική η ίδια πλατφόρμα να επαναχρησιμοποιηθεί ως βάση για άλλου τύπου δηλώσεις ή θεσμικά workflows, όπως δηλώσεις συμφερόντων σε επαγγελματικούς συλλόγους, ετήσια reviews για εργαζόμενους του δημοσίου ή ακόμα και αιτήσεις επιχορηγήσεων. Η ευελιξία του μοντέλου της πλατφόρμας την καθιστά κατάλληλη για οποιαδήποτε διαδικασία με φόρμες, audits, ρόλους και εποπτεία.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Stanford Institute for Human-Centered AI. (2024). AI Index Report 2024. https://aiindex.stanford.edu/report/

[2] McKinsey & Company. (2023). The state of AI in 2023: Generative AI’s breakout year. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year

[3] OECD.(2023) Artificial Intelligence in Society. https://www.oecd.org/publications/artificial-intelligence-in-society-9789264723678-en.htm

[4] M. Raible, "Building Modern Web Applications with Spring Boot and Angular Using JHipster," *SpringOne Conference*, 2019. https://www.jhipster.tech/

[5] Spring.io, "Spring Roo Reference Guide," *Spring Framework Docs*, 2021. https://docs.spring.io/spring-roo/docs/current/reference/html/

[6] Yeoman.io, "Yeoman – The web's scaffolding tool for modern webapps," 2024. https://yeoman.io/

[7] Nx.dev (Nrwl), "Nx: Smart Monorepo Build System," *Official Documentation*, 2024. https://nx.dev/

[8]LoopBack.io, "LoopBack 4 Framework," *IBM API Connect*, 2024: https://loopback.io/

[9] Volosoft, "ABP Framework Documentation," *ABP.IO Platform*, 2024. https://docs.abp.io/en/abp/latest/

[10] LMSys, "Vicuna: An Open-Source Chatbot Impressing GPT-4," 2023.: https://vicuna.lmsys.org/

[11] DeepSeek AI, "DeepSeek LLM Overview," 2024.: https://deepseek.com/

[12] Mistral AI, "Mistral 7B," 2023.: https://mistral.ai/news/announcing-mistral-7b/

[13]Flask: https://flask.palletsprojects.com/.

[14]Flask-CORS Documentation: https://flask-cors.readthedocs.io/en/latest/

[15] Gunicorn, "Green Unicorn: Python WSGI HTTP Server for UNIX": https://gunicorn.org/

[16] A. Bousios, "LLAMA & Krikri: A revolution in Greek AI technology," *Kathimerini English Edition*, Feb. 6, 2024.: https://www.ekathimerini.com/economy/1261724/llama-krikri-a-revolution-in-greek-ai-technology/

[17] Red Hat, "Keycloak Documentation," *Keycloak.org*, 2024.: https://www.keycloak.org/documentation.html

APPENDIX A

API

Μέθοδος	Endpoint	Περιγραφή
GET	/api/submissions	Λίστα όλων των δηλώσεων
POST	/api/submissions	Δημιουργία νέας δήλωσης
GET	/api/submissions/{id}	Ανάκτηση δήλωσης με βάση το ID
PUT	/api/submissions/{id}	Ολική ενημέρωση υπάρχουσας δήλωσης
PATCH	/api/submissions/{id}	Μερική ενημέρωση υπάρχουσας δήλωσης
DELETE	/api/submissions/{id}	Διαγραφή δήλωσης
GET	/api/submission-audits	Λίστα όλων των audit entries
POST	/api/submission-audits	Καταγραφή νέου audit entry
GET	/api/submission-audits/{id}	Ανάκτηση audit entry
PUT	/api/submission-audits/{id}	Ενημέρωση audit entry
PATCH	/api/submission-audits/{id}	Μερική ενημέρωση audit entry
DELETE	/api/submission-audits/{id}	Διαγραφή audit entry
GET	/api/submission-audits/by-submission/{submissionId}	Λήψη audit entries για συγκεκριμένη δήλωση
GET	/api/positions	Λίστα θέσεων
POST	/api/positions	Δημιουργία θέσης
GET	/api/positions/{id}	Ανάκτηση θέσης
PUT	/api/positions/{id}	Ενημέρωση θέσης
PATCH	/api/positions/{id}	Μερική ενημέρωση θέσης
DELETE	/api/positions/{id}	Διαγραφή θέσης
GET	/api/grades	Λίστα βαθμών
POST	/api/grades	Δημιουργία βαθμού
GET	/api/grades/{id}	Ανάκτηση βαθμού

PUT	/api/grades/{id}	Ενημέρωση βαθμού		
PATCH	/api/grades/{id}	Μερική ενημέρωση βαθμού		
DELETE	/api/grades/{id}	Διαγραφή βαθμού		
GET	/api/committees	Λίστα επιτροπών		
POST	/api/committees	Δημιουργία επιτροπής		
GET	/api/committees/{id}	Ανάκτηση επιτροπής		
PUT	/api/committees/{id}	Ενημέρωση επιτροπής		
PATCH	/api/committees/{id}	Μερική ενημέρωση επιτροπής		
DELETE	/api/committees/{id}	Διαγραφή επιτροπής		
GET	/api/admin/users	Λίστα χρηστών (admin)		
POST	/api/admin/users	Δημιουργία χρήστη (admin)		
GET	/api/admin/users/{login}	Ανάκτηση χρήστη μέσω login		
PUT	/api/admin/users	Μαζική ενημέρωση χρηστών		
PUT	/api/admin/users/{login}	Ενημέρωση χρήστη μέσω login		
DELETE	/api/admin/users/{login}	Διαγραφή χρήστη		
POST	/api/register	Εγγραφή χρήστη		
GET	/api/account	Λήψη στοιχείων λογαριασμού		
POST	/api/account	Ενημέρωση στοιχείων λογαριασμού		
POST	/api/account/change-password	Αλλαγή κωδικού		
POST	/api/account/reset-password/init	Έναρξη διαδικασίας ανάκτησης κωδικού		
POST	/api/account/reset-password/finish	Ολοκλήρωση ανάκτησης κωδικού		
GET	/api/activate	Ενεργοποίηση λογαριασμού		
GET	/api/authorities	Λίστα ρόλων		
POST	/api/authorities	Δημιουργία ρόλου		
GET	/api/authorities/{id}	Ανάκτηση ρόλου		
DELETE	/api/authorities/{id}	Διαγραφή ρόλου		
POST	/api/authenticate	Σύνδεση με τοπικά διαπιστευτήρια		
GET	/api/authenticate-oauth2	OAuth2 σύνδεση (π.χ. TaxisNet)		

GET	/api/authenticate-keycloak	Σύνδεση μέσω Keycloak	
GET	/api/users	Λίστα χρηστών (δημόσια προβολή)	

APPENDIX B

JHipster Basic Code

AuthenticateController.java

```
1 | package com.mycompany.myapp.web.rest;
2 |
3 | import static com.mycompany.myapp.security.SecurityUtils.AUTHORITIES_KEY;
4 | import static com.mycompany.myapp.security.SecurityUtils.JWT_ALGORITHM;
5 |
6 | import com.fasterxml.jackson.annotation.JsonProperty;
7 | import com.mycompany.myapp.domain.User;
8 | import com.mycompany.myapp.service.UserService;
9 | import com.mycompany.myapp.web.rest.vm.LoginVM;
10| import jakarta.validation.Valid;
11| import java.net.URI;
12| import java.net.http.HttpClient;
13| import java.net.http.HttpRequest;
14| import java.net.http.HttpResponse;
15| import java.nio.charset.StandardCharsets;
16| import java.security.Principal;
17| import java.time.Instant;
18| import java.time.temporal.ChronoUnit;
19| import java.util.Base64;
20| import java.util.Map;
21| import java.util.stream.Collectors;
22| import org.json.JSONObject;
23| import org.json.JSONObject;
24| import org.slf4j.Logger;
25| import org.slf4j.LoggerFactory;
26| import org.springframework.beans.factory.annotation.Value;
27| import org.springframework.http.HttpHeaders;
28| import org.springframework.http.HttpStatus;
29| import org.springframework.http.MediaType;
30| import org.springframework.http.ResponseEntity;
31| import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
32| import
org.springframework.security.config.annotation.authentication.builders.Authenticati
onManagerBuilder;
33| import org.springframework.security.core.Authentication;
```

```
34| import org.springframework.security.core.GrantedAuthority;
35| import org.springframework.security.core.context.SecurityContextHolder;
36| import org.springframework.security.oauth2.jwt.JwsHeader;
37| import org.springframework.security.oauth2.jwt.JwtClaimsSet;
38| import org.springframework.security.oauth2.jwt.JwtEncoder;
39| import org.springframework.security.oauth2.jwt.JwtEncoderParameters;
40| import org.springframework.web.bind.annotation.*;
41| import org.springframework.web.reactive.function.client.WebClient;
42| import reactor.core.publisher.Mono;
43|
44| /**
45|  * Controller to authenticate users with both JWT and Taxisnet OAuth2.
46|  */
47| @RestController
48| @RequestMapping("/api")
49| public class AuthenticateController {
50|
51|     private static final Logger LOG =
LoggerFactory.getLogger(AuthenticateController.class);
52|
53|     private final JwtEncoder jwtEncoder;
54|     private final AuthenticationManagerBuilder authenticationManagerBuilder;
55|     private final WebClient webClient;
56|     private final UserService userService;
57|
58|
 @Value("${jhipster.security.authentication.jwt.token-validity-in-seconds:0}")
59|     private long tokenValidityInSeconds;
60|
61|
 @Value("${jhipster.security.authentication.jwt.token-validity-in-seconds-for-remem
ber-me:0}")
62|     private long tokenValidityInSecondsForRememberMe;
63|
64|     @Value("${taxisnet.client-id}")
65|     private String taxisnetClientId;
66|
67|     @Value("${taxisnet.client-secret}")
68|     private String taxisnetClientSecret;
69|
70|     @Value("${taxisnet.token-uri}")
71|     private String taxisnetTokenUri;
72|
73|     @Value("${taxisnet.user-info-uri}")
74|     private String taxisnetUserInfoUri;
75|
76|     public AuthenticateController(
77|         JwtEncoder jwtEncoder,
78|         AuthenticationManagerBuilder authenticationManagerBuilder,
79|         WebClient.Builder webClientBuilder,
80|         UserService userService
81|     ) {
82|         this.jwtEncoder = jwtEncoder;
83|         this.authenticationManagerBuilder = authenticationManagerBuilder;
84|         this.webClient = webClientBuilder.build();
85|         this.userService = userService;
86|     }
87|
88|     @PostMapping("/authenticate")
89|     public ResponseEntity<JWTToken> authorize(@Valid @RequestBody LoginVM
loginVM) {
90|         UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(
91|             loginVM.getUsername(),
92|             loginVM.getPassword()
93|         );
94|
95|         Authentication authentication =
authenticationManagerBuilder.getObject().authenticate(authenticationToken);
96|         SecurityContextHolder.getContext().setAuthentication(authentication);
97|         String jwt = this.createToken(authentication, loginVM.isRememberMe());
98|         HttpHeaders httpHeaders = new HttpHeaders();
99|         httpHeaders.setBearerAuth(jwt);
100|          return new ResponseEntity<>(new JWTToken(jwt), httpHeaders,
HttpStatus.OK);
```

```
101|     }
102|
103|     /**
104|      * OAuth2 Authentication via Taxisnet.
105|      *
106|      * @param code Authorization code from Taxisnet.
107|      * @return JWT token.
108|      */
109|     @GetMapping("/authenticate-oauth2")
110|     public ResponseEntity<JWTToken> authorizeWithOAuth2(@RequestParam String
code) {
111|         try {
112|             // Step 1: Exchange authorization code for access token
113|             String accessToken = getAccessTokenFromTaxisnet(code);
114|             LOG.info("Received OAuth2 Access Token: {}", accessToken);
115|
116|             // Step 2: Retrieve user info (XML Response) from Taxisnet
117|             String userInfoXml = getUserInfoFromTaxisnet(accessToken);
118|             LOG.info("User Info from Taxisnet: {}", userInfoXml);
119|
120|             // Step 3: Register user in the system if they don't exist
121|             User registeredUser =
userService.registerUserFromXml(userInfoXml);
122|             if (registeredUser == null) {
123|                 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
124|             }
125|
126|             // Step 4: Generate JWT token
127|             String jwt = createToken(registeredUser.getLogin(), "ROLE_USER");
128|
129|             HttpHeaders httpHeaders = new HttpHeaders();
130|             httpHeaders.setBearerAuth(jwt);
131|
132|             return new ResponseEntity<>(new JWTToken(jwt), httpHeaders,
HttpStatus.OK);
133|         } catch (Exception e) {
134|             LOG.error("OAuth2 authentication failed", e);
135|             return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
136|         }
137|     }
138|
139|     @GetMapping("/authenticate-keycloak")
140|     public ResponseEntity<JWTToken> authorizeWithKeycloak(@RequestParam String
code) {
141|         try {
142|             System.out.println("▶ Received code: " + code);
143|
144|             String idToken = getIdTokenFromKeycloak(code);
145|             System.out.println("Received id_token");
146|
147|             String[] parts = idToken.split("\\.");
148|             if (parts.length != 3) {
149|                 System.err.println("Invalid JWT format");
150|                 return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
151|             }
152|
153|             String payloadJson = new
String(Base64.getUrlDecoder().decode(parts[1]), StandardCharsets.UTF_8);
154|             System.out.println("🔍 Decoded JWT payload: " + payloadJson);
155|
156|             JSONObject payload = new JSONObject(payloadJson);
157|             String username = payload.optString("preferred_username", null);
158|             String email = payload.optString("email", null);
159|             String name = payload.optString("name", null);
160|
161|             System.out.println("👤 Extracted: username=" + username + ",
email=" + email + ", name=" + name);
162|
163|             if (username == null) {
164|                 System.err.println("Missing 'preferred_username' in token");
165|                 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
166|             }
167|
```

```
168|             User user = userService.registerUserFromKeycloak(username, email,
name);
169|             if (user == null) {
170|                 System.err.println("Failed to register user: " + username);
171|                 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
172|             }
173|
174|             String jwt = createToken(user.getLogin(), "ROLE_USER");
175|             HttpHeaders headers = new HttpHeaders();
176|             headers.setBearerAuth(jwt);
177|             return new ResponseEntity<>(new JWTToken(jwt), headers,
HttpStatus.OK);
178|         } catch (Exception e) {
179|             System.err.println("Exception in /authenticate-keycloak");
180|             e.printStackTrace();
181|             return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
182|         }
183|     }
184|
185|     private String getIdTokenFromKeycloak(String code) {
186|         try {
187|             String requestBody =
188|                 "client_id=public-client" + "&grant_type=authorization_code" +
"&redirect_uri=https://147.102.74.34" + "&code=" + code;
189|
190|             System.out.println("Sending token request to Keycloak...");
191|             System.out.println("client_id=public-client");
192|             System.out.println("grant_type=authorization_code");
193|             System.out.println("redirect_uri=https://147.102.74.34/");
194|             System.out.println("code=" + code);
195|
196|             String tokenResponse = webClient
197|                 .post()
198|
.uri("https://gpu.dslab.ece.ntua.gr:8443/realms/billys/protocol/openid-connect/toke
n")
199|                 .header("Content-Type", "application/x-www-form-urlencoded")
200|                 .bodyValue(requestBody)
201|                 .retrieve()
202|                 .onStatus(
203|                     status -> status.isError(),
204|                     response ->
205|                         response
206|                             .bodyToMono(String.class)
207|                             .flatMap(body -> {
208|                                 System.err.println("Keycloak token endpoint
returned error: " + body);
209|                                 return Mono.error(new
RuntimeException("Keycloak error: " + body));
210|                             })
211|                 )
212|                 .bodyToMono(String.class)
213|                 .block();
214|
215|             System.out.println("Raw token response: " + tokenResponse);
216|
217|             JSONObject json = new JSONObject(tokenResponse);
218|             return json.getString("id_token");
219|         } catch (Exception e) {
220|             System.err.println("Exception while calling Keycloak token
endpoint:");
221|             e.printStackTrace();
222|             throw new RuntimeException("Failed to get id_token from Keycloak",
e);
223|         }
224|     }
225|
226|     /**
227|      * Calls Taxisnet to exchange authorization code for access token.
228|      */
229|     private String getAccessTokenFromTaxisnet(String code) {
230|         String requestBody =
231|             "client_id=" +
232|             taxisnetClientId +
233|             "&client_secret=" +
234|             taxisnetClientSecret +
```

```
235|              "&scope=read" +
236|              "&code=" +
237|              code +
238|              "&grant_type=authorization_code";
239|
240|          return webClient
241|              .post()
242|              .uri(taxisnetTokenUri)
243|              .header("Content-Type", "application/x-www-form-urlencoded")
244|              .bodyValue(requestBody)
245|              .retrieve()
246|              .bodyToMono(String.class)
247|              .map(response -> new
JSONObject(response).getString("access_token"))
248|              .block(); // Blocking for simplicity
249|      }
250|
251|      /**
252|       * Calls Taxisnet to get user info.
253|       */
254|      private String getUserInfoFromTaxisnet(String accessToken) {
255|          return webClient
256|              .get()
257|              .uri(taxisnetUserInfoUri)
258|              .header("Authorization", "Bearer " + accessToken)
259|              .header("Accept", "application/json")
260|              .retrieve()
261|              .bodyToMono(String.class)
262|              .block();
263|      }
264|
265|      /**
266|       * Creates JWT token from authentication.
267|       */
268|      public String createToken(Authentication authentication, boolean
rememberMe) {
269|          String authorities =
authentication.getAuthorities().stream().map(GrantedAuthority::getAuthority).collec
t(Collectors.joining(" "));
270|
271|          Instant now = Instant.now();
272|          Instant validity = rememberMe
273|              ? now.plus(this.tokenValidityInSecondsForRememberMe,
ChronoUnit.SECONDS)
274|              : now.plus(this.tokenValidityInSeconds, ChronoUnit.SECONDS);
275|
276|          JwtClaimsSet claims = JwtClaimsSet.builder()
277|              .issuedAt(now)
278|              .expiresAt(validity)
279|              .subject(authentication.getName())
280|              .claim(AUTHORITIES_KEY, authorities)
281|              .build();
282|
283|          JwsHeader jwsHeader = JwsHeader.with(JWT_ALGORITHM).build();
284|          return this.jwtEncoder.encode(JwtEncoderParameters.from(jwsHeader,
claims)).getTokenValue();
285|      }
286|
287|      /**
288|       * Creates JWT token directly from username and authorities.
289|       */
290|      public String createToken(String username, String authorities) {
291|          Instant now = Instant.now();
292|          Instant validity = now.plus(this.tokenValidityInSeconds,
ChronoUnit.SECONDS);
293|
294|          JwtClaimsSet claims = JwtClaimsSet.builder()
295|              .issuedAt(now)
296|              .expiresAt(validity)
297|              .subject(username)
298|              .claim(AUTHORITIES_KEY, authorities)
299|              .build();
300|
301|          JwsHeader jwsHeader = JwsHeader.with(JWT_ALGORITHM).build();
302|          return this.jwtEncoder.encode(JwtEncoderParameters.from(jwsHeader,
claims)).getTokenValue();
```

```
303|     }
304|
305|     /**
306|      * Object to return as body in JWT Authentication.
307|      */
308|     static class JWTToken {
309|
310|         private String idToken;
311|
312|         JWTToken(String idToken) {
313|             this.idToken = idToken;
314|         }
315|
316|         @JsonProperty("id_token")
317|         String getIdToken() {
318|             return idToken;
319|         }
320|
321|         void setIdToken(String idToken) {
322|             this.idToken = idToken;
323|         }
324|     }
325| }
```

SecurityConfiguration.Java

```
1 | package com.mycompany.myapp.config;
2 |
3 | import static org.springframework.security.config.Customizer.withDefaults;
4 | import static
org.springframework.security.web.util.matcher.AntPathRequestMatcher.antMatcher;
5 |
6 | import com.mycompany.myapp.security.*;
7 | import com.mycompany.myapp.web.filter.SpaWebFilter;
8 | import java.util.List;
9 | import org.springframework.context.annotation.Bean;
10| import org.springframework.context.annotation.Configuration;
11| import org.springframework.core.env.Environment;
12| import org.springframework.core.env.Profiles;
13| import org.springframework.http.HttpMethod;
14| import
org.springframework.security.config.annotation.method.configuration.EnableMethodSec
urity;
15| import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
16| import
org.springframework.security.config.annotation.web.configurers.HeadersConfigurer.Fr
ameOptionsConfig;
17| import org.springframework.security.config.http.SessionCreationPolicy;
18| import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
19| import org.springframework.security.crypto.password.PasswordEncoder;
20| import
org.springframework.security.oauth2.server.resource.web.BearerTokenAuthenticationEn
tryPoint;
21| import
org.springframework.security.oauth2.server.resource.web.access.BearerTokenAccessDen
iedHandler;
22| import org.springframework.security.web.SecurityFilterChain;
23| import
org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
```

```
24| import
org.springframework.security.web.header.writers.ReferrerPolicyHeaderWriter;
25| import org.springframework.security.web.servlet.util.matcher.MvcRequestMatcher;
26| import org.springframework.web.cors.CorsConfiguration;
27| import org.springframework.web.cors.CorsConfigurationSource;
28| import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
29| import org.springframework.web.servlet.handler.HandlerMappingIntrospector;
30| import tech.jhipster.config.JHipsterConstants;
31| import tech.jhipster.config.JHipsterProperties;
32|
33| @Configuration
34| @EnableMethodSecurity(securedEnabled = true)
35| public class SecurityConfiguration {
36|
37|     private final Environment env;
38|
39|     private final JHipsterProperties jHipsterProperties;
40|
41|     public SecurityConfiguration(Environment env, JHipsterProperties
jHipsterProperties) {
42|         this.env = env;
43|         this.jHipsterProperties = jHipsterProperties;
44|     }
45|
46|     @Bean
47|     public CorsConfigurationSource corsConfigurationSource() {
48|         CorsConfiguration configuration = new CorsConfiguration();
49|         configuration.setAllowedOrigins(List.of("*")); // Allow specific
origins
50|         configuration.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE",
"PATCH", "OPTIONS"));
51|         configuration.setAllowedHeaders(List.of("*"));
52|         //configuration.setAllowCredentials(true);
53|
54|         UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
55|         source.registerCorsConfiguration("/**", configuration);
56|         return source;
57|     }
58|
59|     @Bean
60|     public PasswordEncoder passwordEncoder() {
61|         return new BCryptPasswordEncoder();
62|     }
63|
64|     @Bean
65|     public SecurityFilterChain filterChain(HttpSecurity http,
MvcRequestMatcher.Builder mvc) throws Exception {
66|         http
67|             .cors(cors -> cors.configurationSource(corsConfigurationSource()))
68|             .csrf(csrf -> csrf.disable())
69|             .addFilterAfter(new SpaWebFilter(),
BasicAuthenticationFilter.class)
70|             .headers(headers ->
71|                 headers
72|                     .contentSecurityPolicy(csp ->
csp.policyDirectives(jHipsterProperties.getSecurity().getContentSecurityPolicy()))
73|                     .frameOptions(FrameOptionsConfig::sameOrigin)
74|                     .referrerPolicy(referrer ->
referrer.policy(ReferrerPolicyHeaderWriter.ReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_
ORIGIN))
75|                     .permissionsPolicy(permissions ->
76|                         permissions.policy(
77|                             "camera=(), fullscreen=(self), geolocation=(),
gyroscope=(), magnetometer=(), microphone=(), midi=(), payment=(), sync-xhr=()"
78|                         )
79|                     )
80|             )
81|             .authorizeHttpRequests(
82|                 authz ->
83|                     // prettier-ignore
84|                 authz
85|                     .requestMatchers(mvc.pattern("/index.html"),
mvc.pattern("/*.js"), mvc.pattern("/*.txt"), mvc.pattern("/*.json"),
mvc.pattern("/*.map"), mvc.pattern("/*.css")).permitAll()
```

```
                    .requestMatchers(mvc.pattern("/*.ico"),
mvc.pattern("/*.png"), mvc.pattern("/*.svg"), mvc.pattern("/*.webapp")).permitAll()
                    .requestMatchers(mvc.pattern("/app/**")).permitAll()
                    .requestMatchers(mvc.pattern("/i18n/**")).permitAll()
                    .requestMatchers(mvc.pattern("/content/**")).permitAll()
                    .requestMatchers(mvc.pattern("/swagger-ui/**")).permitAll()
                    .requestMatchers(mvc.pattern(HttpMethod.POST,
"/api/authenticate")).permitAll()
                    .requestMatchers(mvc.pattern(HttpMethod.GET,
"/api/authenticate")).permitAll()
                    .requestMatchers(HttpMethod.GET,
"/api/authenticate-oauth2").permitAll()
                    .requestMatchers(mvc.pattern(HttpMethod.GET,
"/api/authenticate-keycloak")).permitAll()
                    .requestMatchers(mvc.pattern("/api/register")).permitAll()
                    .requestMatchers(mvc.pattern("/api/activate")).permitAll()

.requestMatchers(mvc.pattern("/api/account/reset-password/init")).permitAll()

.requestMatchers(mvc.pattern("/api/account/reset-password/finish")).permitAll()

.requestMatchers(mvc.pattern("/api/admin/**")).hasAuthority(AuthoritiesConstants.AD
MIN)
                     .requestMatchers(mvc.pattern("/api/**")).authenticated()

.requestMatchers(mvc.pattern("/v3/api-docs/**")).hasAuthority(AuthoritiesConstants.
ADMIN)

.requestMatchers(mvc.pattern("/management/health")).permitAll()

.requestMatchers(mvc.pattern("/management/health/**")).permitAll()

.requestMatchers(mvc.pattern("/management/info")).permitAll()

.requestMatchers(mvc.pattern("/management/prometheus")).permitAll()

.requestMatchers(mvc.pattern("/management/**")).hasAuthority(AuthoritiesConstants.A
DMIN)
                    .requestMatchers(mvc.pattern(HttpMethod.GET,
"/api/positions")).authenticated() // Allow GET /api/positions for users

.requestMatchers(mvc.pattern("/api/positions/**")).hasAuthority(AuthoritiesConstant
s.ADMIN) // Restrict all other positions endpoints to admins
                    .requestMatchers(mvc.pattern(HttpMethod.GET,
"/api/grades")).authenticated() // Allow GET /api/grades for users

.requestMatchers(mvc.pattern("/api/grades/**")).hasAuthority(AuthoritiesConstants.A
DMIN) // Restrict all other grades endpoints to admins
                    .requestMatchers(mvc.pattern(HttpMethod.GET,
"/api/committees")).authenticated() // Allow GET /api/committees for users

.requestMatchers(mvc.pattern("/api/committees/**")).hasAuthority(AuthoritiesConstan
ts.ADMIN) // Restrict all other committees endpoints to admins
                )
                .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .exceptionHandling(exceptions ->
                    exceptions
                        .authenticationEntryPoint(new
BearerTokenAuthenticationEntryPoint())
                        .accessDeniedHandler(new BearerTokenAccessDeniedHandler())
                )
                .oauth2ResourceServer(oauth2 -> oauth2.jwt(withDefaults()));
            if
(env.acceptsProfiles(Profiles.of(JHipsterConstants.SPRING_PROFILE_DEVELOPMENT))) {
                http.authorizeHttpRequests(authz ->
authz.requestMatchers(antMatcher("/h2-console/**")).permitAll());
            }
            return http.build();
        }

        @Bean
        MvcRequestMatcher.Builder mvc(HandlerMappingIntrospector introspector) {
            return new MvcRequestMatcher.Builder(introspector);
        }
    }
```

UserService.Java

```
1 | package com.mycompany.myapp.service;
2 |
3 | import com.mycompany.myapp.config.Constants;
4 | import com.mycompany.myapp.domain.Authority;
5 | import com.mycompany.myapp.domain.User;
6 | import com.mycompany.myapp.repository.AuthorityRepository;
7 | import com.mycompany.myapp.repository.UserRepository;
8 | import com.mycompany.myapp.security.AuthoritiesConstants;
9 | import com.mycompany.myapp.security.SecurityUtils;
10| import com.mycompany.myapp.service.dto.AdminUserDTO;
11| import com.mycompany.myapp.service.dto.UserDTO;
12| import java.io.StringReader;
13| import java.time.Instant;
14| import java.time.temporal.ChronoUnit;
15| import java.util.*;
16| import java.util.HashSet;
17| import java.util.Optional;
18| import java.util.Set;
19| import java.util.stream.Collectors;
20| import javax.xml.parsers.DocumentBuilder;
21| import javax.xml.parsers.DocumentBuilderFactory;
22| import org.slf4j.Logger;
23| import org.slf4j.LoggerFactory;
24| import org.springframework.cache.CacheManager;
25| import org.springframework.data.domain.Page;
26| import org.springframework.data.domain.Pageable;
27| import org.springframework.scheduling.annotation.Scheduled;
28| import org.springframework.security.crypto.password.PasswordEncoder;
29| import org.springframework.stereotype.Service;
30| import org.springframework.transaction.annotation.Transactional;
31| import org.w3c.dom.Document;
32| import org.w3c.dom.Element;
33| import org.xml.sax.InputSource;
34| import tech.jhipster.security.RandomUtil;
35|
36| /**
37|  * Service class for managing users.
38|  */
39| @Service
40| @Transactional
41| public class UserService {
42|
43|     private static final Logger LOG =
LoggerFactory.getLogger(UserService.class);
44|
45|     private final UserRepository userRepository;
46|
47|     private final PasswordEncoder passwordEncoder;
48|
49|     private final AuthorityRepository authorityRepository;
50|
51|     private final CacheManager cacheManager;
52|
53|     public UserService(
54|         UserRepository userRepository,
55|         PasswordEncoder passwordEncoder,
56|         AuthorityRepository authorityRepository,
57|         CacheManager cacheManager
58|     ) {
59|         this.userRepository = userRepository;
60|         this.passwordEncoder = passwordEncoder;
```

```
61|          this.authorityRepository = authorityRepository;
62|          this.cacheManager = cacheManager;
63|      }
64|
65|      /**
66|       * Registers a user from XML response if they don't already exist.
67|       *
68|       * @param xmlResponse The XML response containing user details.
69|       * @return The registered user or existing user details.
70|       */
71|      public User registerUserFromXml(String xmlResponse) {
72|          try {
73|              // Parse XML
74|              DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
75|              DocumentBuilder builder = factory.newDocumentBuilder();
76|              Document document = builder.parse(new InputSource(new
StringReader(xmlResponse)));
77|
78|              Element userInfo = (Element)
document.getElementsByTagName("userinfo").item(0);
79|              if (userInfo == null) {
80|                  LOG.error("XML Parsing Error: <userinfo> tag not found");
81|                  return null;
82|              }
83|
84|              // Extract attributes from XML
85|              String taxId = userInfo.getAttribute("taxid").trim();
86|              String firstName = userInfo.getAttribute("firstname").trim();
87|              String lastName = userInfo.getAttribute("lastname").trim();
88|              String email = taxId + "@taxisnet.gr"; // Fixed email as per
request
89|              String username = taxId; // Use taxId as the username
90|              String password = RandomUtil.generatePassword(); // Generate a
random password
91|
92|              LOG.info("Parsed XML - TaxID: {}, FirstName: {}, LastName: {},
Email: {}", taxId, firstName, lastName, email);
93|
94|              // Check if user already exists
95|              Optional<User> existingUser =
userRepository.findOneByLogin(username);
96|              if (existingUser.isPresent()) {
97|                  LOG.info("User already exists with TaxID: {}", taxId);
98|                  return existingUser.get();
99|              }
100|
101|              // Create new user
102|              User newUser = new User();
103|              newUser.setLogin(username);
104|              newUser.setPassword(passwordEncoder.encode(password));
105|              newUser.setFirstName(firstName);
106|              newUser.setLastName(lastName);
107|              newUser.setEmail(email);
108|              newUser.setLangKey(Constants.DEFAULT_LANGUAGE); // Default
language
109|              newUser.setActivated(true); // Auto-activate
110|              newUser.setActivationKey(null); // No activation needed
111|
112|              // Assign default USER role
113|              Set<Authority> authorities = new HashSet<>();
114|
 authorityRepository.findById(AuthoritiesConstants.USER).ifPresent(authorities::add
);
115|              newUser.setAuthorities(authorities);
116|
117|              // Save user
118|              userRepository.save(newUser);
119|              this.clearUserCaches(newUser);
120|
121|              LOG.info("User registered successfully with TaxID: {}", taxId);
122|              return newUser;
123|          } catch (Exception e) {
124|              LOG.error("Error parsing XML and registering user: {}",
e.getMessage(), e);
125|              return null;
```

```
126|         }
127|     }
128|
129|     public User registerUserFromKeycloak(String username, String email, String
fullName) {
130|         try {
131|             // Parse full name into first/last (basic split)
132|             String[] nameParts = fullName != null ? fullName.trim().split(" ",
2) : new String[] { username, "" };
133|             String firstName = nameParts[0];
134|             String lastName = nameParts.length > 1 ? nameParts[1] : "";
135|
136|             LOG.info("Keycloak - Parsed User: username={}, firstName={},
lastName={}, email={}", username, firstName, lastName, email);
137|
138|             // Check if user already exists
139|             Optional<User> existingUser =
userRepository.findOneByLogin(username);
140|             if (existingUser.isPresent()) {
141|                 LOG.info("User already exists with username: {}", username);
142|                 return existingUser.get();
143|             }
144|
145|             // Create new user
146|             User newUser = new User();
147|             newUser.setLogin(username);
148|
 newUser.setPassword(passwordEncoder.encode(RandomUtil.generatePassword()));
149|             newUser.setFirstName(firstName);
150|             newUser.setLastName(lastName);
151|             newUser.setEmail(email != null ? email : username +
"@keycloak.local");
152|             newUser.setLangKey(Constants.DEFAULT_LANGUAGE);
153|             newUser.setActivated(true);
154|             newUser.setActivationKey(null);
155|
156|             // Assign ROLE_USER authority
157|             Set<Authority> authorities = new HashSet<>();
158|
 authorityRepository.findById(AuthoritiesConstants.USER).ifPresent(authorities::add
);
159|             newUser.setAuthorities(authorities);
160|
161|             // Save and clear caches
162|             userRepository.save(newUser);
163|             this.clearUserCaches(newUser);
164|
165|             LOG.info("User registered successfully from Keycloak: {}",
username);
166|             return newUser;
167|         } catch (Exception e) {
168|             LOG.error("Error registering user from Keycloak: {}",
e.getMessage(), e);
169|             return null;
170|         }
171|     }
172|
173|     public Optional<User> activateRegistration(String key) {
174|         LOG.debug("Activating user for activation key {}", key);
175|         return userRepository
176|             .findOneByActivationKey(key)
177|             .map(user -> {
178|                 // activate given user for the registration key.
179|                 user.setActivated(true);
180|                 user.setActivationKey(null);
181|                 this.clearUserCaches(user);
182|                 LOG.debug("Activated user: {}", user);
183|                 return user;
184|             });
185|     }
186|
187|     public Optional<User> completePasswordReset(String newPassword, String
key) {
188|         LOG.debug("Reset user password for reset key {}", key);
189|         return userRepository
190|             .findOneByResetKey(key)
```

```
            .filter(user -> user.getResetDate().isAfter(Instant.now().minus(1,
ChronoUnit.DAYS)))
            .map(user -> {
                user.setPassword(passwordEncoder.encode(newPassword));
                user.setResetKey(null);
                user.setResetDate(null);
                this.clearUserCaches(user);
                return user;
            });
    }

    public Optional<User> requestPasswordReset(String mail) {
        return userRepository
            .findOneByEmailIgnoreCase(mail)
            .filter(User::isActivated)
            .map(user -> {
                user.setResetKey(RandomUtil.generateResetKey());
                user.setResetDate(Instant.now());
                this.clearUserCaches(user);
                return user;
            });
    }

    public User registerUser(AdminUserDTO userDTO, String password) {
        userRepository
            .findOneByLogin(userDTO.getLogin().toLowerCase())
            .ifPresent(existingUser -> {
                boolean removed = removeNonActivatedUser(existingUser);
                if (!removed) {
                    throw new UsernameAlreadyUsedException();
                }
            });
        userRepository
            .findOneByEmailIgnoreCase(userDTO.getEmail())
            .ifPresent(existingUser -> {
                boolean removed = removeNonActivatedUser(existingUser);
                if (!removed) {
                    throw new EmailAlreadyUsedException();
                }
            });
        User newUser = new User();
        String encryptedPassword = passwordEncoder.encode(password);
        newUser.setLogin(userDTO.getLogin().toLowerCase());
        // new user gets initially a generated password
        newUser.setPassword(encryptedPassword);
        newUser.setFirstName(userDTO.getFirstName());
        newUser.setLastName(userDTO.getLastName());
        if (userDTO.getEmail() != null) {
            newUser.setEmail(userDTO.getEmail().toLowerCase());
        }
        newUser.setImageUrl(userDTO.getImageUrl());
        newUser.setLangKey(userDTO.getLangKey());
        // new user is not active
        newUser.setActivated(false);
        // new user gets registration key
        newUser.setActivationKey(RandomUtil.generateActivationKey());
        Set<Authority> authorities = new HashSet<>();

 authorityRepository.findById(AuthoritiesConstants.USER).ifPresent(authorities::add
);
        newUser.setAuthorities(authorities);
        userRepository.save(newUser);
        this.clearUserCaches(newUser);
        LOG.debug("Created Information for User: {}", newUser);
        return newUser;
    }

    private boolean removeNonActivatedUser(User existingUser) {
        if (existingUser.isActivated()) {
            return false;
        }
        userRepository.delete(existingUser);
        userRepository.flush();
        this.clearUserCaches(existingUser);
        return true;
    }
```

```
264|
265|     public User createUser(AdminUserDTO userDTO) {
266|         User user = new User();
267|         user.setLogin(userDTO.getLogin().toLowerCase());
268|         user.setFirstName(userDTO.getFirstName());
269|         user.setLastName(userDTO.getLastName());
270|         if (userDTO.getEmail() != null) {
271|             user.setEmail(userDTO.getEmail().toLowerCase());
272|         }
273|         user.setImageUrl(userDTO.getImageUrl());
274|         if (userDTO.getLangKey() == null) {
275|             user.setLangKey(Constants.DEFAULT_LANGUAGE); // default language
276|         } else {
277|             user.setLangKey(userDTO.getLangKey());
278|         }
279|         String encryptedPassword =
passwordEncoder.encode(RandomUtil.generatePassword());
280|         user.setPassword(encryptedPassword);
281|         user.setResetKey(RandomUtil.generateResetKey());
282|         user.setResetDate(Instant.now());
283|         user.setActivated(true);
284|         if (userDTO.getAuthorities() != null) {
285|             Set<Authority> authorities = userDTO
286|                 .getAuthorities()
287|                 .stream()
288|                 .map(authorityRepository::findById)
289|                 .filter(Optional::isPresent)
290|                 .map(Optional::get)
291|                 .collect(Collectors.toSet());
292|             user.setAuthorities(authorities);
293|         }
294|         userRepository.save(user);
295|         this.clearUserCaches(user);
296|         LOG.debug("Created Information for User: {}", user);
297|         return user;
298|     }
299|
300|     /**
301|      * Update all information for a specific user, and return the modified
user.
302|      *
303|      * @param userDTO user to update.
304|      * @return updated user.
305|      */
306|     public Optional<AdminUserDTO> updateUser(AdminUserDTO userDTO) {
307|         return Optional.of(userRepository.findById(userDTO.getId()))
308|             .filter(Optional::isPresent)
309|             .map(Optional::get)
310|             .map(user -> {
311|                 this.clearUserCaches(user);
312|                 user.setLogin(userDTO.getLogin().toLowerCase());
313|                 user.setFirstName(userDTO.getFirstName());
314|                 user.setLastName(userDTO.getLastName());
315|                 if (userDTO.getEmail() != null) {
316|                     user.setEmail(userDTO.getEmail().toLowerCase());
317|                 }
318|                 user.setImageUrl(userDTO.getImageUrl());
319|                 user.setActivated(userDTO.isActivated());
320|                 user.setLangKey(userDTO.getLangKey());
321|                 Set<Authority> managedAuthorities = user.getAuthorities();
322|                 managedAuthorities.clear();
323|                 userDTO
324|                     .getAuthorities()
325|                     .stream()
326|                     .map(authorityRepository::findById)
327|                     .filter(Optional::isPresent)
328|                     .map(Optional::get)
329|                     .forEach(managedAuthorities::add);
330|                 userRepository.save(user);
331|                 this.clearUserCaches(user);
332|                 LOG.debug("Changed Information for User: {}", user);
333|                 return user;
334|             })
335|             .map(AdminUserDTO::new);
336|     }
337|
```

```
338|     public void deleteUser(String login) {
339|         userRepository
340|             .findOneByLogin(login)
341|             .ifPresent(user -> {
342|                 userRepository.delete(user);
343|                 this.clearUserCaches(user);
344|                 LOG.debug("Deleted User: {}", user);
345|             });
346|     }
347|
348|     /**
349|      * Update basic information (first name, last name, email, language) for
the current user.
350|      *
351|      * @param firstName first name of user.
352|      * @param lastName  last name of user.
353|      * @param email     email id of user.
354|      * @param langKey   language key.
355|      * @param imageUrl  image URL of user.
356|      */
357|     public void updateUser(String firstName, String lastName, String email,
String langKey, String imageUrl) {
358|         SecurityUtils.getCurrentUserLogin()
359|             .flatMap(userRepository::findOneByLogin)
360|             .ifPresent(user -> {
361|                 user.setFirstName(firstName);
362|                 user.setLastName(lastName);
363|                 if (email != null) {
364|                     user.setEmail(email.toLowerCase());
365|                 }
366|                 user.setLangKey(langKey);
367|                 user.setImageUrl(imageUrl);
368|                 userRepository.save(user);
369|                 this.clearUserCaches(user);
370|                 LOG.debug("Changed Information for User: {}", user);
371|             });
372|     }
373|
374|     @Transactional
375|     public void changePassword(String currentClearTextPassword, String
newPassword) {
376|         SecurityUtils.getCurrentUserLogin()
377|             .flatMap(userRepository::findOneByLogin)
378|             .ifPresent(user -> {
379|                 String currentEncryptedPassword = user.getPassword();
380|                 if (!passwordEncoder.matches(currentClearTextPassword,
currentEncryptedPassword)) {
381|                     throw new InvalidPasswordException();
382|                 }
383|                 String encryptedPassword =
passwordEncoder.encode(newPassword);
384|                 user.setPassword(encryptedPassword);
385|                 this.clearUserCaches(user);
386|                 LOG.debug("Changed password for User: {}", user);
387|             });
388|     }
389|
390|     @Transactional(readOnly = true)
391|     public Page<AdminUserDTO> getAllManagedUsers(Pageable pageable) {
392|         return userRepository.findAll(pageable).map(AdminUserDTO::new);
393|     }
394|
395|     @Transactional(readOnly = true)
396|     public Page<UserDTO> getAllPublicUsers(Pageable pageable) {
397|         return
userRepository.findAllByIdNotNullAndActivatedIsTrue(pageable).map(UserDTO::new);
398|     }
399|
400|     @Transactional(readOnly = true)
401|     public Optional<User> getUserWithAuthoritiesByLogin(String login) {
402|         return userRepository.findOneWithAuthoritiesByLogin(login);
403|     }
404|
405|     @Transactional(readOnly = true)
406|     public Optional<User> getUserWithAuthorities() {
```

```
407|         return
SecurityUtils.getCurrentUserLogin().flatMap(userRepository::findOneWithAuthoritiesB
yLogin);
408|     }
409|
410|     /**
411|      * Not activated users should be automatically deleted after 3 days.
412|      * <p>
413|      * This is scheduled to get fired everyday, at 01:00 (am).
414|      */
415|     @Scheduled(cron = "0 0 1 * * ?")
416|     public void removeNotActivatedUsers() {
417|         userRepository
418|
.findAllByActivatedIsFalseAndActivationKeyIsNotNullAndCreatedDateBefore(Instant.now
().minus(3, ChronoUnit.DAYS))
419|             .forEach(user -> {
420|                 LOG.debug("Deleting not activated user {}", user.getLogin());
421|                 userRepository.delete(user);
422|                 this.clearUserCaches(user);
423|             });
424|     }
425|
426|     /**
427|      * Gets a list of all the authorities.
428|      * @return a list of all the authorities.
429|      */
430|     @Transactional(readOnly = true)
431|     public List<String> getAuthorities() {
432|         return
authorityRepository.findAll().stream().map(Authority::getName).toList();
433|     }
434|
435|     private void clearUserCaches(User user) {
436|
 Objects.requireNonNull(cacheManager.getCache(UserRepository.USERS_BY_LOGIN_CACHE))
.evict(user.getLogin());
437|         if (user.getEmail() != null) {
438|
 Objects.requireNonNull(cacheManager.getCache(UserRepository.USERS_BY_EMAIL_CACHE))
.evict(user.getEmail());
439|         }
440|     }
441| }
```

SubmissionService.java

```
1 | package com.mycompany.myapp.service;
2 |
3 | import com.mycompany.myapp.domain.Submission;
4 | import com.mycompany.myapp.domain.SubmissionAudit;
5 | import com.mycompany.myapp.repository.SubmissionAuditRepository;
6 | import com.mycompany.myapp.repository.SubmissionRepository;
7 | import com.mycompany.myapp.security.SecurityUtils;
8 | import com.mycompany.myapp.service.dto.SubmissionDTO;
9 | import com.mycompany.myapp.service.mapper.SubmissionMapper;
10| import java.time.Instant;
11| import java.util.Optional;
12| import org.slf4j.Logger;
13| import org.slf4j.LoggerFactory;
14| import org.springframework.data.domain.Page;
15| import org.springframework.data.domain.Pageable;
16| import org.springframework.stereotype.Service;
17| import org.springframework.transaction.annotation.Transactional;
18|
19| /**
```

```
 * Service Implementation for managing {@link
com.mycompany.myapp.domain.Submission}.
 */
@Service
@Transactional
public class SubmissionService {

    private static final Logger LOG =
LoggerFactory.getLogger(SubmissionService.class);

    private final SubmissionRepository submissionRepository;

    private final SubmissionMapper submissionMapper;

    private final SubmissionAuditRepository submissionAuditRepository;

    public SubmissionService(
        SubmissionRepository submissionRepository,
        SubmissionMapper submissionMapper,
        SubmissionAuditRepository submissionAuditRepository
    ) {
        this.submissionRepository = submissionRepository;
        this.submissionMapper = submissionMapper;
        this.submissionAuditRepository = submissionAuditRepository;
    }

    /**
     * Save a submission.
     *
     * @param submissionDTO the entity to save.
     * @return the persisted entity.
     */
    public SubmissionDTO save(SubmissionDTO submissionDTO) {
        LOG.debug("Request to save Submission : {}", submissionDTO);
        Submission submission = submissionMapper.toEntity(submissionDTO);
        submission = submissionRepository.save(submission);
        return submissionMapper.toDto(submission);
    }

    /**
     * Update a submission.
     *
     * @param submissionDTO the entity to save.
     * @return the persisted entity.
     */
    public SubmissionDTO update(SubmissionDTO submissionDTO) {
        LOG.debug("Request to update Submission : {}", submissionDTO);

        // Fetch the existing Submission from the database
        Optional<Submission> existingSubmission =
submissionRepository.findById(submissionDTO.getId());

        Submission submission = submissionMapper.toEntity(submissionDTO);
        submission = submissionRepository.save(submission);

        // If an existing record was found, create an audit entry
        existingSubmission.ifPresent(oldSubmission -> {
            SubmissionAudit audit = new SubmissionAudit();
            audit.setAfm(oldSubmission.getAfm());
            audit.setAdt(oldSubmission.getAdt());
            audit.setLastName(oldSubmission.getLastName());
            audit.setFirstName(oldSubmission.getFirstName());
            audit.setFatherName(oldSubmission.getFatherName());
            audit.setAcquisitionDate(oldSubmission.getAcquisitionDate());
            audit.setLossDate(oldSubmission.getLossDate());
            audit.setOrganizationUnit(oldSubmission.getOrganizationUnit());

audit.setNewOrganizationUnit(oldSubmission.getNewOrganizationUnit());
            audit.setProtocolNumber(oldSubmission.getProtocolNumber());
            audit.setDecisionDate(oldSubmission.getDecisionDate());
            audit.setPreviousSubmission(oldSubmission.getPreviousSubmission());
            audit.setModifiedDate(Instant.now());

audit.setModifiedBy(SecurityUtils.getCurrentUserLogin().orElse("unknown"));
            audit.setChangeType("UPDATE");
```

```
90|             audit.setOriginalSubmission(oldSubmission);
91|
92|             submissionAuditRepository.save(audit);
93|         });
94|
95|         return submissionMapper.toDto(submission);
96|     }
97|
98|     /**
99|      * Partially update a submission.
100|      *
101|      * @param submissionDTO the entity to update partially.
102|      * @return the persisted entity.
103|      */
104|     public Optional<SubmissionDTO> partialUpdate(SubmissionDTO submissionDTO)
{
105|         LOG.debug("Request to partially update Submission : {}",
submissionDTO);
106|
107|         return submissionRepository
108|             .findById(submissionDTO.getId())
109|             .map(existingSubmission -> {
110|                 // Store the previous version in SubmissionAudit
111|                 SubmissionAudit audit = new SubmissionAudit();
112|                 audit.setAfm(existingSubmission.getAfm());
113|                 audit.setAdt(existingSubmission.getAdt());
114|                 audit.setLastName(existingSubmission.getLastName());
115|                 audit.setFirstName(existingSubmission.getFirstName());
116|                 audit.setFatherName(existingSubmission.getFatherName());
117|
 audit.setAcquisitionDate(existingSubmission.getAcquisitionDate());
118|                 audit.setLossDate(existingSubmission.getLossDate());
119|
 audit.setOrganizationUnit(existingSubmission.getOrganizationUnit());
120|
 audit.setNewOrganizationUnit(existingSubmission.getNewOrganizationUnit());
121|
 audit.setProtocolNumber(existingSubmission.getProtocolNumber());
122|                 audit.setDecisionDate(existingSubmission.getDecisionDate());
123|
 audit.setPreviousSubmission(existingSubmission.getPreviousSubmission());
124|                 audit.setModifiedDate(Instant.now());
125|
 audit.setModifiedBy(SecurityUtils.getCurrentUserLogin().orElse("unknown"));
126|                 audit.setChangeType("PARTIAL_UPDATE");
127|                 audit.setOriginalSubmission(existingSubmission);
128|
129|                 submissionAuditRepository.save(audit);
130|
131|                 // Apply the partial update
132|                 submissionMapper.partialUpdate(existingSubmission,
submissionDTO);
133|                 return existingSubmission;
134|             })
135|             .map(submissionRepository::save)
136|             .map(submissionMapper::toDto);
137|     }
138|
139|     // /**
140|     //  * Get all the submissions.
141|     //  *
142|     //  * @param pageable the pagination information.
143|     //  * @return the list of entities.
144|     //  */
145|     // @Transactional(readOnly = true)
146|     // public Page<SubmissionDTO> findAll(Pageable pageable) {
147|     //     LOG.debug("Request to get all Submissions");
148|     //     return
submissionRepository.findAll(pageable).map(submissionMapper::toDto);
149|     // }
150|
151|     /**
152|      * Get all the submissions with eager load of many-to-many relationships.
153|      *
154|      * @return the list of entities.
155|      */
```

```
156|     public Page<SubmissionDTO> findAllWithEagerRelationships(Pageable
pageable) {
157|         return
submissionRepository.findAllWithEagerRelationships(pageable).map(submissionMapper::
toDto);
158|     }
159|
160|     /**
161|      * Get one submission by id.
162|      *
163|      * @param id the id of the entity.
164|      * @return the entity.
165|      */
166|     @Transactional(readOnly = true)
167|     public Optional<SubmissionDTO> findOne(Long id) {
168|         LOG.debug("Request to get Submission : {}", id);
169|         return
submissionRepository.findOneWithEagerRelationships(id).map(submissionMapper::toDto)
;
170|     }
171|
172|     /**
173|      * Delete the submission by id.
174|      *
175|      * @param id the id of the entity.
176|      */
177|     public void delete(Long id) {
178|         LOG.debug("Request to delete Submission : {}", id);
179|         submissionRepository.deleteById(id);
180|     }
181|
182|     @Transactional(readOnly = true)
183|     public Page<SubmissionDTO> findAll(Pageable pageable) {
184|         LOG.debug("Request to get all Submissions");
185|         return
submissionRepository.findAll(pageable).map(submissionMapper::toDto);
186|     }
187|
188|     @Transactional(readOnly = true)
189|     public Page<SubmissionDTO> findAllByUser(Pageable pageable, String login)
{
190|         LOG.debug("Request to get all Submissions for user: {}", login);
191|         return submissionRepository.findAllByUserLogin(pageable,
login).map(submissionMapper::toDto);
192|     }
193| }
```

Application.yml

```
1 | # ===================================================================
2 | # Spring Boot configuration.
3 | #
4 | # This configuration will be overridden by the Spring profile you use,
5 | # for example application-dev.yml if you use the "dev" profile.
6 | #
7 | # More information on profiles: https://www.jhipster.tech/profiles/
8 | # More information on configuration properties:
https://www.jhipster.tech/common-application-properties/
9 | # ===================================================================
10|
11| # ===================================================================
12| # Standard Spring Boot properties.
13| # Full reference is available at:
14| #
http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-pr
operties.html
15| # ===================================================================
16|
```

```yaml
---
# Conditionally disable springdoc on missing api-docs profile
taxisnet:
  client-id: 'TJP48Y31612'
  client-secret: 'pm!VC67#gn4'
  token-uri: 'https://test.gsis.gr/oauth2servergov/oauth/token'
  user-info-uri: 'https://test.gsis.gr/oauth2servergov/userinfo?format=json'
  redirect-uri: 'https://147.102.74.34/'

spring:
  config:
    activate:
      on-profile: '!api-docs'
springdoc:
  api-docs:
    enabled: false
---
management:
  endpoints:
    web:
      base-path: /management
      exposure:
        include:
          - configprops
          - env
          - health
          - info
          - jhimetrics
          - jhiopenapigroups
          - logfile
          - loggers
          - prometheus
          - threaddump
          - caches
          - liquibase
  endpoint:
    health:
      show-details: when_authorized
      roles: 'ROLE_ADMIN'
      probes:
        enabled: true
      group:
        liveness:
          include: livenessState
        readiness:
          include: readinessState,db
    jhimetrics:
      enabled: true
  info:
    git:
      mode: full
    env:
      enabled: true
  health:
    mail:
      enabled: false # When using the MailService, configure an SMTP server and
set this to true
  prometheus:
    metrics:
      export:
        enabled: true
        step: 60
  observations:
    key-values:
      application: ${spring.application.name}
  metrics:
    enable:
      http: true
      jvm: true
      logback: true
      process: true
      system: true
    distribution:
      percentiles-histogram:
        all: true
      percentiles:
```

```
        all: 0, 0.5, 0.75, 0.95, 0.99, 1.0
    data:
      repository:
        autotime:
          enabled: true
    tags:
      application: ${spring.application.name}

spring:
  application:
    name: myApp
  docker:
    compose:
      enabled: true
      lifecycle-management: start-only
      file: src/main/docker/services.yml
  profiles:
    # The commented value for `active` can be replaced with valid Spring profiles to load.
    # Otherwise, it will be filled in by maven when building the JAR file
    # Either way, it can be overridden by `--spring.profiles.active` value passed in the commandline or `-Dspring.profiles.active` set in `JAVA_OPTS`
    active: '@spring.profiles.active@'
    group:
      dev:
        - dev
        - api-docs
        # Uncomment to activate TLS for the dev profile
        #- tls
  jmx:
    enabled: false
  data:
    jpa:
      repositories:
        bootstrap-mode: deferred
  jpa:
    open-in-view: false
    properties:
      hibernate.jdbc.time_zone: UTC
      hibernate.timezone.default_storage: NORMALIZE
      hibernate.type.preferred_instant_jdbc_type: TIMESTAMP
      hibernate.id.new_generator_mappings: true
      hibernate.connection.provider_disables_autocommit: true
      hibernate.cache.use_second_level_cache: true
      hibernate.cache.use_query_cache: false
      hibernate.generate_statistics: false
      # modify batch size as necessary
      hibernate.jdbc.batch_size: 25
      hibernate.order_inserts: true
      hibernate.order_updates: true
      hibernate.query.fail_on_pagination_over_collection_fetch: true
      hibernate.query.in_clause_parameter_padding: true
    hibernate:
      ddl-auto: none
      naming:
        physical-strategy: org.hibernate.boot.model.naming.CamelCaseToUnderscoresNamingStrategy
        implicit-strategy: org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
  messages:
    basename: i18n/messages
  main:
    allow-bean-definition-overriding: true
  mvc:
    problemdetails:
      enabled: true
  security:
    oauth2:
      resourceserver:
        jwt:
          authority-prefix: ''
          authorities-claim-name: auth
  task:
    execution:
      thread-name-prefix: my-app-task-
      pool:
```

```
        core-size: 2
        max-size: 50
        queue-capacity: 10000
    scheduling:
      thread-name-prefix: my-app-scheduling-
      pool:
        size: 2
  thymeleaf:
    mode: HTML
  output:
    ansi:
      console-available: true

server:
  port: 8080  # Keep JHipster running on 8080
  ssl:
    enabled: true
    key-store-type: PKCS12
    key-store: classpath:keystore.p12
    key-store-password: papalas  # Use your actual password
    key-alias: selfsigned
  servlet:
    session:
      cookie:
        http-only: true

springdoc:
  show-actuator: true

# Properties to be exposed on the /info management endpoint
info:
  # Comma separated list of profiles that will trigger the ribbon to show
  display-ribbon-on-profiles: 'dev'

# ===================================================================
# JHipster specific properties
#
# Full reference is available at:
https://www.jhipster.tech/common-application-properties/
# ===================================================================

jhipster:
  clientApp:
    name: 'myApp'
  # By default CORS is disabled. Uncomment to enable.
  cors:
    allowed-origins: "*"
    allowed-methods: "GET, POST, PUT, DELETE, PATCH, OPTIONS"
    allowed-headers: "*"
    exposed-headers: "Authorization,Link,X-Total-Count"
    #allow-credentials: true
    max-age: 1800
  mail:
    from: myApp@localhost
  api-docs:
    default-include-pattern: /api/**
    management-include-pattern: /management/**
    title: My App API
    description: My App API documentation
    version: 0.0.1
    terms-of-service-url:
    contact-name:
    contact-url:
    contact-email:
    license: unlicensed
    license-url:
  security:
    content-security-policy: "default-src 'self'; frame-src 'self' data:;
script-src 'self' 'unsafe-inline' 'unsafe-eval' https://storage.googleapis.com;
style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self' data:"
# ===================================================================
# Application specific properties
# Add your own application properties here, see the ApplicationProperties
class
# to have type-safe configuration, like in the JHipsterProperties above
#
```

```
# More documentation is available at:
# https://www.jhipster.tech/common-application-properties/
# ===================================================================

# application:
```

APPENDIX C

Angular Basic Code

Middleware.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../../../services/auth.service';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-middleware',
  standalone: false,

  templateUrl: './middleware.component.html',
  styleUrl: './middleware.component.css'
})
export class MiddlewareComponent implements OnInit {
  constructor(private authService: AuthService, private router: Router, private route: ActivatedRoute){}

  ngOnInit(): void {
    this.route.queryParams.subscribe((params) => {
      const code = params['code']; // Get 'code' from the URL
      const iss = params['iss'];
      const sessionState = params['session_state'];
      const paramKeys = Object.keys(params);

      const isKeycloakLogin =
        code && iss && sessionState &&
        paramKeys.length === 3;

      if(isKeycloakLogin){
        console.log('Keycloak login detected');
        console.log('Code found in URL:', code);
        this.authService.loginKeycloak(code).subscribe(response => {
          this.authService.saveToken(response.id_token);
          this.authService.saveOAuthMethod('keycloak');
          const roles = ['ROLE_USER']
          this.authService.saveRoles(['ROLE_USER']);

          // Redirect based on role
          if (roles.includes('ROLE_ADMIN')) {
            this.router.navigate(['/admin/home']);
          } else if (roles.includes('ROLE_USER')) {
            this.router.navigate(['/user/home']);
          }
        }, error => {
          console.error('Oauth2 Login failed', error);
          alert('Απέτυχε η αυθεντικοποίηση μέσω keycloak')
          this.router.navigate(['/login'])
        })
      }
      else if (code) {
        console.log('Code found in URL:', code);
```

```
        this.authService.loginOauth2(code).subscribe(response => {
          //console.log('Login successful!', response);
          this.authService.saveToken(response.id_token);
          this.authService.saveOAuthMethod('oauth2');
          const roles = this.authService.getRoles();
          this.authService.saveRoles(roles);

          // Redirect based on role
          if (roles.includes('ROLE_ADMIN')) {
            this.router.navigate(['/admin/home']);
          } else if (roles.includes('ROLE_USER')) {
            this.router.navigate(['/user/home']);
          }
        }, error => {
          console.error('Oauth2 Login failed', error);
          alert('Απέτυχε η αυθεντικοποίηση μέσω taxisnet')
          this.router.navigate(['/login'])
        })
      }
      else if (this.authService.isAuthenticated()) {
        const roles = this.authService.getRoles();

        // Redirect based on authenticated role
        if (roles.includes('ROLE_ADMIN')) {
          this.router.navigate(['/admin/home']);
        } else if (roles.includes('ROLE_USER')) {
          this.router.navigate(['/user/home']);
        }
      }
      else {
        console.log('No code found and not authenticated, redirecting to
login');
        this.router.navigate(['/login']);
      }
    });
  }
}
```

Login.component.ts

```
import { CommonModule } from '@angular/common';
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from
'@angular/forms';
import { AuthService } from '../../../services/auth.service';
import { Router } from '@angular/router';
import { environment } from '../../../config';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'], // Assuming you're using plain CSS
  imports: [ReactiveFormsModule, CommonModule]
})
export class LoginComponent implements OnInit{
  loginForm: FormGroup;
  selectedTab: string = 'taxisnet'; // Default tab
  private client_id = environment.client_id;

  constructor(private fb: FormBuilder, private authService: AuthService,
private router: Router) {
    this.loginForm = this.fb.group({
      username: ['', [Validators.required]],
      password: ['', [Validators.required]]
```

```
23|      });
24|    }
25|
26|    ngOnInit(): void {
27|      if (this.authService.isAuthenticated()) {
28|        const roles = this.authService.getRoles();
29|
30|        // Redirect based on role
31|        if (roles.includes('ROLE_ADMIN')) {
32|          this.router.navigate(['/admin/home']);
33|        } else if (roles.includes('ROLE_USER')) {
34|          this.router.navigate(['/user/home']);
35|        }
36|      }
37|    }
38|
39|    onSubmit(): void {
40|      if (this.loginForm.valid) {
41|        const username = this.loginForm.get('username')?.value;
42|        const password = this.loginForm.get('password')?.value;
43|        //console.log('Login form submitted!', { username, password });
44|
45|        const credentials = {
46|          username: this.loginForm.get('username')?.value,
47|          password: this.loginForm.get('password')?.value,
48|          rememberMe: true // Include rememberMe flag
49|        };
50|
51|        this.authService.login(credentials).subscribe(response => {
52|          //console.log('Login successful!', response);
53|          this.authService.saveToken(response.id_token);
54|          this.authService.saveOAuthMethod('jwt');
55|          const roles = this.authService.getRoles()
56|          this.authService.saveRoles(roles);
57|
58|          // Redirect based on role
59|          if (roles.includes('ROLE_ADMIN')) {
60|            this.router.navigate(['/admin/home']);
61|          } else if (roles.includes('ROLE_USER')) {
62|            this.router.navigate(['/user/home']);
63|          }
64|        }, error => {
65|          console.error('Login failed', error);
66|          alert('Λάθος κωδικός ή όνομα χρήστη')
67|        });
68|      } else {
69|        console.log('Form is invalid');
70|      }
71|    }
72|
73|    selectTab(tab: string) {
74|      this.selectedTab = tab;
75|    }
76|
77|    redirectToTaxisnet() {
78|      window.location.href =
`https://test.gsis.gr/oauth2servergov/oauth/authorize?response_type=code&client_id=
${this.client_id}`;
79|    }
80|
81|    redirectToKeycloak() {
82|      const clientId = 'public-client'; // your client ID in Keycloak
83|      const redirectUri = encodeURIComponent('https://147.102.74.34'); // or your
deployed frontend
84|      const realm = 'billys'; // your realm name
85|      const responseType = 'code';
86|      const scope = 'openid';
87|
88|      window.location.href =
`https://gpu.dslab.ece.ntua.gr:8443/realms/${realm}/protocol/openid-connect/auth` +
89|        `?client_id=${clientId}` +
90|        `&redirect_uri=${redirectUri}` +
91|        `&response_type=${responseType}` +
92|        `&scope=${scope}`;
93|    }
94|
```

```
95|
96| }
```

Auth.service.ts

```
1 | import { Injectable } from '@angular/core';
2 | import { HttpClient, HttpHeaders } from '@angular/common/http';
3 | import { Router } from '@angular/router';
4 | import { JwtHelperService } from '@auth0/angular-jwt';
5 | import { Observable } from 'rxjs';
6 | import { environment } from '../config';
7 |
8 | @Injectable({
9 |   providedIn: 'root',
10| })
11| export class AuthService {
12|   private apiUrl = environment.apiUrl;
13|   private jwtHelper = new JwtHelperService();
14|   private tokenKey = 'auth-token';
15|   private rolesKey = 'user-roles';
16|   private oauthMethod = 'jwt';
17|
18|   constructor(private http: HttpClient, private router: Router) {}
19|
20|   login(credentials: { username: string; password: string }): Observable<any> {
21|     return this.http.post(`${this.apiUrl}/authenticate`, credentials);
22|   }
23|
24|   loginOauth2(code: string): Observable<any> {
25|     return this.http.get(`${this.apiUrl}/authenticate-oauth2?code=${code}`);
26|   }
27|
28|   loginKeycloak(code: string): Observable<any> {
29|     return this.http.get(`${this.apiUrl}/authenticate-keycloak?code=${code}`);
30|   }
31|
32|
33|   saveToken(token: string) {
34|     localStorage.setItem(this.tokenKey, token);
35|   }
36|
37|   getToken(): string | null {
38|     return localStorage.getItem(this.tokenKey);
39|   }
40|
41|   saveOAuthMethod(method: string){
42|     localStorage.setItem(this.oauthMethod, method);
43|   }
44|
45|   getOAuthMethod(): string | null {
46|     return localStorage.getItem(this.oauthMethod);
47|   }
48|
49|
50|   isAuthenticated(): boolean {
51|     const token = this.getToken();
52|     return token ? !this.jwtHelper.isTokenExpired(token) : false;
53|   }
54|
55|   getRoles(): string[] {
56|     const token = this.getToken();
57|     if (token) {
58|       const decodedToken = this.jwtHelper.decodeToken(token);
59|       return decodedToken.auth || [];
60|     }
```

```
61|      return [];
62|    }
63|
64|    saveRoles(roles: string[]) {
65|      localStorage.setItem(this.rolesKey, JSON.stringify(roles));
66|    }
67|
68|    logout() {
69|      localStorage.removeItem(this.tokenKey);
70|      this.router.navigate(['/login']);
71|    }
72|
73|    logout_oauth2(){
74|      localStorage.removeItem(this.tokenKey);
75|      window.location.href = environment.logoutURL;
76|    }
77|
78|    logout_keycloak(){
79|      window.location.href = environment.logoutKeycloakURL;
80|      localStorage.removeItem(this.tokenKey);
81|    }
82|
83|
84|    getAccount(): Observable<any> {
85|      const token = localStorage.getItem('auth-token'); // Retrieve the token
from localStorage
86|      const headers = new HttpHeaders({
87|            Authorization: `Bearer ${token}`
88|          });
89|
90|      return this.http.get(`${this.apiUrl}/account`, { headers });
91|    }
92| }
```

Submission.service.ts

```
1 | // Import necessary modules
2 | import { Injectable } from '@angular/core';
3 | import { HttpClient, HttpHeaders } from '@angular/common/http';
4 | import { Observable } from 'rxjs';
5 | import { environment } from '../config';
6 |
7 | @Injectable({
8 |   providedIn: 'root'
9 | })
10| export class SubmissionService {
11|   private apiUrl = `${environment.apiUrl}/submissions`; // API endpoint
12|
13|   constructor(private http: HttpClient) {}
14|
15|   // Method to post a new submission
16|   createSubmission(submission: any): Observable<any> {
17|     const headers = this.getAuthHeaders();
18|     return this.http.post<any>(this.apiUrl, submission, { headers });
19|   }
20|
21|   // Method to update an existing submission
22|   updateSubmission(submissionId: number, updatedSubmission: any):
Observable<any> {
23|     const headers = this.getAuthHeaders();
24|     return this.http.put<any>(`${this.apiUrl}/${submissionId}`,
updatedSubmission, { headers });
25|   }
26|
27|   // Fetch paginated submissions
```

```
  getSubmissions(page: number = 0, size: number = 20, eagerload: boolean =
true): Observable<any> {
    const headers = this.getAuthHeaders();
    const params = {
      page: page.toString(),
      size: size.toString(),
      eagerload: eagerload.toString()
    };
    return this.http.get<any>(this.apiUrl, { headers, params });
  }

  // Utility method to generate headers with authentication token
  private getAuthHeaders(): HttpHeaders {
    const token = localStorage.getItem('auth-token'); // Retrieve the token
from localStorage
    return new HttpHeaders({
      'Content-Type': 'application/json',
      Authorization: `Bearer ${token}`
    });
  }
}
```

Chat.component.ts

```
import { Component, Input } from '@angular/core';
import { ChatService } from '../../../services/chat.service';

@Component({
  selector: 'app-chat',
  templateUrl: './chat.component.html',
  styleUrls: ['./chat.component.css'],
  standalone: false
})
export class ChatComponent {
  messages: { text: string; sender: string }[] = [];
  messageText = '';
  isLoading = false; // Spinner state

  @Input() auditResponse: any = []; // Receiving audit data

  constructor(private chatService: ChatService) {}

  sendMessage() {
    if (!this.messageText.trim()) return;

    // Add user message
    this.messages.push({ text: this.messageText, sender: 'user' });

    // Show loading spinner
    this.isLoading = true;

    // Extract `modifiedBy` and `modifiedDate`, format as string
    const formattedData = this.auditResponse.map((item: { id: any; modifiedBy:
any; modifiedDate: any; }) =>
      `ID: ${item.id}, Τροποποιήθηκε από: ${item.modifiedBy}, Ημερομηνία
Τροποποίησης: ${item.modifiedDate.split('T')[0]}`
    ).join("\n");

    // Prepare full prompt
    const fullPrompt = `${this.messageText}\n\nΑρχείο
Καταγραφής:\n${formattedData}`;

    console.log(fullPrompt)
    // Call backend (JHipster API)
    this.chatService.sendMessage(fullPrompt).subscribe(response => {
        this.messages.push({ text: response.response, sender: 'bot' });

        // Hide loading spinner
```

```
42|          this.isLoading = false;
43|       }, () => {
44|        // In case of an error, hide spinner
45|        this.isLoading = false;
46|       });
47|
48|      this.messageText = '';
49|    }
50| }
```