Εφαρμογή Επαγγελματικής Κοινωνικής Δικτύωσης

Καρναβάς Μάριος-Τριαντάφυλλος

sdi1900232

Μερκούρης Κωνσταντίνος

sdi1800319

Θεωδόρου Ορέστης

sdi2000058

Εαρινό Εξάμηνο 2023-2024

Contents

1 Εισαγωγή

Σκοπός της εργασίας είναι η ανάπτυξη μίας εφαρμογής κοινωνικής δικτύωσης τύπου Linked In, με νωτιαίο άκρο που προσφέρει RESTful προγραμματιστική διεπαφή και μετωπιαίο άκρο για πιο εύκολη χρήση από τον μέσο χρήστη.

Χρησιμοποιήσαμε τις παρακάτω τεχνολογίες:

- PostgreSQL: Για αποθήκευση των δεδομένων σε ισχυρή, εύρωστη και ευέλικτη βάση δεδομένων.
- ASP .NET: Για ανάπτυξη του νωτιαίου άκρου που προσφέρει RESTful API με authenticated και authorised αιτήματα.
- React : Για την ανάπτυξη του μετωπιαίου άκρου.

2 Οδηγίες εγκατάστασης

Παρακάτω θα βρείτε τις οδηγίες εγκατάστασης της εφαρμογής. Αν υπάρξει οποιοδήποτε πρόβλημα με την εγκατάσταση, στείλτε μήνυμα στον Μάριο Καρναβά (sdi1900232) αν αφορά το νωτιαίο άκρο, η σε κάποιον από τους άλλους 2 συνεργάτες αν αφορά το μετωπιαίο άκρο.

2.1 Εγκατάσταση Νωτιαίου Άκρου

2.1.1 Εγκατάσταση PostgreSQL

Οδηγίες για την εγκατάσταση της PostgreSQL θα βρείτε σε αυτό το βίντεο. Κρατείστε το όνομα χρήστη και τον κωδικό που χρησιμοποιήσατε κατά την εγκατάσταση, θα τον χρειαστούμε και αργότερα. Ανοίξτε το εργαλείο PGAdmin και δημιουργήστε μία βάση με όνομα linked_out, ανεβασμένη στο localhost, με κύριο χρήστη τον postgres (ή αν επιλάξετε κάποιο άλλο όνομα κατά την εγκατάσταση, εκείνο).

2.1.2 Εγκατάσταση .NET Framework

Μπορείτε να κατεβάσετε το .Net8.0 SDK από αυτή την ιστοσελίδα.

2.1.3 Εγκατάσταση Εφαρμογής

Ο πηγαίος κώδικας της εφαρμογής του νωτιαίου άκρου είναι αποθηκευμένος στο zip που αποστάλθηκε στο eclass. Πρωτού ξεκινήσει η εφαρμογή, βεβαιωθείται ότι:

- Τα localhost URL των HTTP και HTTPS αιτημάτων είναι ελεύθερα και δεν τα χρησιμοποιεί κάποια άλλη εφαρμογή. Είναι προεπιλεγμένα στα ports 5048 και 8043 αντίστοιχα.
- Το path για το certificate που χρησιμοποιείται για τα αιτήματα HTTPS και ο κωδικός του αντιστοιχούν στις παραμέτρους, και ταυτόχρονα το certificate είναι δηλωμένο ως έμπιστο από τον υπολογιστή σας.
- Το connection string για την βάση αντιστοιχεί στα δεδομένα που χρησιμοποιήσατε κατά την εγκατάσταση. Η αρχική τιμή του είναι : "Host=localhost; Database=linked_out; Username=postgres; Password=a;". Το όνομα και ο host της βάσης μπορούν να βρεθούν και μέσω του εργαλείου PGAdmin.

Αφότου ετοιμαστούν οι παραπάνω παραμέτροι, πρέπει να χρησιμοποιήσετε το αρχικό migration για την βάση δεδομένων. Όπως αναφέρθηκε ήδη, τα schemas της βάσης παράγονται αυτόματα από το εργαλείο dotnet βάσει της κλάσης ApiContext και των οντοτήτων που χρησιμοποιεί. Πρέπει να τρέξετε τις παρακάτω εντολές:

- `dotnet ef database update —context ApiContext`

Αν δεν υπάρχει κάποιο πρόβλημα με τις ρυθμίσεις, οι εντολές θα δημιουργήσουν και θα εφαρμόσουν στην βάση το Migration, δημιουργώντας έτσι τους βασικούς πίνακες της εφαρμογής.

Σε περιπτωση που το παραπάνω δεν δουλεύει, διαγράψτε όλα τα αρχεία στον φάκελο Migrations και τρέξτε τις 2 παρακάτω εντολές:

- `dotnet ef migrations add InitialMigration —context ApiContext`

  `dotnet ef database update —context ApiContext`

2.1.4 Δημιουργία Dummy Data και Τρέξιμο Νωτιαίου Άκρου

Αφότου ολοκληρωθούν όλες οι παραπάνω διαδικασίες, μπορείται να τρέξετε την εφαρμογή. Στην γραμμή. Η εφαρμογή αρχικά δεν έχει καθόλου δεδομένα. Κάντε uncomment τις γραμμές 168-170 στην **πρώτη και μόνο στην πρώτη** εκτέλεση της εφαρμογής για να έχει ορισμένα δεδομένα.

2.2 Εγκατάσταση Μετωπιαίου Άκρου

2.2.1 Εγκατάσταση React και λοιπών εξαρτήσεων

Προαπαιτούμενα:
Βεβαιωθείτε ότι έχετε εγκαταστήσει τα εξής:

- Node.js (v12 ή νεότερο) – Κατεβάστε το Node.js
- npm (έρχεται με το Node.js)

Μεταβείτε στον φάκελο `my-app` (το frontend) και εγκαταστήστε όλα τα απαραίτητα πακέτα Node.js:

cd FRONTEND_TEDI–1/my–app
npm install

Αυτό θα εγκαταστήσει όλα τα απαραίτητα πακέτα για το React frontend.

2.2.2 Ρύθμιση Περιβαλλοντικών Μεταβλητών (Frontend)

Εάν το frontend χρειάζεται να επικοινωνεί με το API του backend, θα πρέπει να ρυθμίσετε τις περιβαλλοντικές μεταβλητές.

Δημιουργήστε ένα αρχείο `api.js` στον φάκελο `my-app/src/services`. Για παράδειγμα:
Ενημερώστε το αρχείο `api.js` με τη σωστή διεύθυνση URL του API (που δείχνει στο backend σας):

baseURL=http://localhost:5048

2.2.3 Εκκινήστε το Frontend

Σε ένα νέο παράθυρο terminal, μεταβείτε στον φάκελο του frontend και εκκινήστε το frontend:

cd FRONTEND_TEDI–1/my–app
npm start

Από προεπιλογή, η React εφαρμογή θα τρέχει στο `http://localhost:5048`.

3 Ανάπτυξη Νωτιαίο Άκρου

Όπως ήδη αναφέρθηκε, για την ανάπτυξη του Νωτιαίου άκρου χρησιμοποιήθηκε η τεχνολογία ASP .NET πάνω από βάση χτισμένη σε PostgreSQL και πίνακες της βάσης είναι χτισμένοι αυτόματα μέσω του CLI του framework (dotnet), με δεδομένα αυτόματα populated από την εφαρμογή. Η .NET είναι αρκετά παρόμοια με την Springboot, αλλά έχει αρκετές σημαντικές διαφορές, η πρώτη από τις οποίες φαίνεται άμεσα με την δημιουργία μίας καινούριας εφαρμογής. Στο αρχείο Program.cs μπορούμε να δούμε την εφαρμογή μας να κτίζεται βήμα-βήμα, συγκεκριμένα:

- Δημιουργείται ένα στιγμιότυπο της κλάσης WebAppBuilder, πάνω στο οποίο προσαρτώνται οι διάφορες λειτουργίες που χρειάζεται η εφαρμογή μας με ανεξάρτητη σειρά.
- Το στιγμιότυπο αυτό κτίζει την εφαρμογή μας και ενεργοποιεί τα επιμέρους σημεία της. Εδώ, η σειρά ενεργοποίησης έχει σημασία, καθώς είναι η ίδια σειρά την οποία θα ακολουθήσουν τα αιτήματα του κάθε χρήστη.

Η ανάπτυξη RESTFUL API στο ASP .NET γίνεται κατ' εξοχήν με την ανάπτυξη Controllers, κλάσεων που δρομολογούν τα αιτήματα από συγκεκριμένα endpoints σε δεδομένες συναρτήσεις και επιστρέφουν τα αποτελέσματα των συναρτήσεων αυτών σε μορφή JSON. Τα Controllers χρησιμοποιούν Services, τα οποία από την πλευρά τους αλληλεπιδρούν με την βάση μέσω του ειδικού service APiContext, το οποίο παρέχει

- Απεικόνιση της δομής της βάσης εντός του κώδικα απεικονίζοντας τους πίνακες της βάσης σε στιγμιότυπα της κλάσης DbSet.

- Διεπαφή ανάμεσα στον κώδικα και στην βάση μεσω δημιουργίας ερωτημάτων LINQ (θα τα δούμε πιο μετά).

Το framework για την επιτήρηση των Οντοτήτων του Αντικειμενοσχεσιακού Μοντέλου ονομάζεται Entity Framework Core, ή EFCore εν συντομία.

Ακολουθήσαμε αυστηρά modular προσέγγιση στην ανάπτυξη και του μετωπιαίου και του νωτιαίου άκρου, αναθέτοντας στο νωτιαίο άκρο και ταυτόχρονα στην βάση όσο περισσότερο από το υπολογιστικό μέρος της εφαρμογής μπορούσαμε.

Κάθε αίτημα απαιτέι ταυτοποίηση του χρήστη μέσω Json Web Token, και μπορεί να απαγορεύσει την διεκπεραίωση ορισμένων αιτημάτων αν τα δεδομένα που παρέχονται στο JWT δεν την καθιστούν επιτρεπτή. Στην .NET κατά κανόνα κάθε σωστή προσέγγιση χρησιμοποιεί εξουσιοδότηση βάσει πολιτικής, και αυτή χρησιμοποιήσαμε και εμείς, καθώς το σύστημα εξουσιοδότησης βάσει ρόλου της .NET δεν ήταν αρκετά ευέλικτο για να καλύψουμε όλες τις ανάγκες του. Εξουσιοδότηση βάσει πόρων θα μπορούσε επίσης να χρησιμοποιηθεί.

Θα εξετάσουμε πιο λεπτομερώς την ανάπτυξη του νωτιαίου άκρου "Bottom-Up", ξεκινώντας δηλαδή από το επίπεδο της βάσης και των μοντέλων και καταλήγοντας στην ταυτοποίηση και εξουσιοδότηση του χρήστη κατα δτην διεκπεραίωση των αιτημάτων του.

Οι παρακάτω υποενότητες δεν θα μπούν εις βάθως στην λειτουργία κάθε αιτήματος και την δομή κάθε οντότητες, καθώς όλες αυτές οι πληροφορίες είναι ήδη documented από την ίδια την εφαρμογή μέσω του Swagger. Αν ξεκινήσετε την εφαρμογή και πάτε στην σελίδα `http://localhost:5048/swagger/index.html`, θα δείτε αναλυτικά τις παρακάτω πληροφορίες για όλα τα αιτήματα:

- Το endpoint στο οποίο βρίσκονται.
- Την μέθοδο την οποία χρησιμοποιούν (GET, POST, DELETE).
- Τις παραμέτρους που δέχονται, καθώς και το είδος και τον τύπο των παραμέτρων.
- Το σώμα της αιτήματος.
- Τα δεδομένα επιστροφής ενός αιτήματος σε επιτυχή εκτέλεση.
- Ορισμένους από τους κωδικούς σφάλματος ενός αιτήματος.

Παρέχονται επίσης αναλυτικές πληροφορίες για όλα τα μοντέλα παραμέτρων και επιστροφής των αιτημάτων.

3.1 Ανάπτυξη Μοντέλων

Τα μοντέλα στην .NET αποθηκεύουν στην βάση όλες τις ιδιότητες με δημόσιους getters και setters (Public Properties) τους. Υποστηρίζουν συσχετίσεις μονόδρομες και αμφίδρομες συσχετίσεις, One-One, One-Many και Many-Many, προαιρετικές και μη. Το EFCore είναι επίσης πολύ αναλυτικό στην εξερεύνηση του συστήματος, και μπορεί να βγάλει μόνο του πολλά συμπεράσματα για την κατάσταση του. Για παράδειγμα, αν μία ιδιότητα σε μία κλάση ονομάζεται Id, γίνεται αυτόματα αντιστοίχισή ανάμεσα σε αυτήν και στην αντίστοιχη στήλη Id της βάσης, η οποία ταυτόχρονα αναγνωρίζεται ως το unique Id της συσχέτισης.

Για την ανάπτυξη μοντέλων, αποφασίσαμε να χρησιμοποιήσουμε one-way συσχετίσεις, στο κέντρο όλων των οποίων βρίσκεται η κλάση RegularUser, η οποία έχει μόνο μία κατευθυνόμενη συσχέτιση προς τα στοιχεία της που μπορούν να μην είναι δημόσια. Οι υπόλοιπες οντότητες έχουν συνήθως κατευθυνόμενες συσχετίσεις πρός την κλάση του χρήστη (πχ: Η κλάση PostBase έχει κατευθυνόμενη συσχέτιση πρός τον χρήστη στο πεδίο PostedBy, αλλά ο κάθε χρήστης δεν έχει πεδίο που αποτυπώνει τα άρθρα ή τις δουλειές που έχει αναρτήσει). Παρόμοια προσέγγιση είχαμε για άλλες οντότητες που βασίζονται στην υπάρξη κάποιας άλλης. Η προσέγγιση αυτή είχε το μειονέκτημα οτι πολλές πληροφορίες ήταν πιο δυσεύρετες στον κώδικα και ήταν πιο απαιτητική στην ανάπτυξη του APIContext, αλλά:

- Έκανε την δημιουργία απαντήσεων είδους JSON πολύ πιο εύκολη, καθώς εξαφάνισε τις κυκλικές εξαρτήσεις μεταξύ των οντοτήτων.
- Μας επέτρεψε να χρησιμοποιήσουμε Lazy Loading στην εφαρμογή μας μέσω virtual Properties, που διευκόλυνε τα ερωτήματα στην βάση.

3.2 Ανάπτυξη API Context

Η κλάση ApiContext προσδιορίζει την διεπαφή ανάμεσα στον προγραμματιστή και την βάση δεδομένων. Για να χρησιμοποιηθεί αυτή η διεπαφή, πρέπει να οριστεί αρχικά μιά υπόκλαση της κλάσης DbContext (στην δικιά μας περίπτωση, αυτή είναι η ApiContext) και να αρχικοποιηθεί από το αντικείμενο τύπου WebApplicationBuilder στο αρχείο Program.cs:

```
builder.Services.AddDbContext<ApiContext>(
    opt => opt
        .UseLazyLoadingProxies()
        .UseNpgsql(builder.Configuration.GetConnectionString("WebApiDatabase")),
    contextLifetime: ServiceLifetime.Scoped
);
```

Με τις πρόσθετες ρυθμίσεις επιτρέπουμε την χρήση lazy loading για τα πεδία συσχετίσεων, κάτι που θα μας βοηθήσει στην ανάπτυξη ερωτημάτων μέσω services αργότερα, και ορίζουμε τον τρόπο που προσφέρεται σε άλλα κομμάτια του προγράμματος μέσω έγχυσης εξαρτήσεων. Στην συγκεκριμένη περίπτωση, η επιλογή scoped ορίζει ότι κάθε αίτημα HTTP έχει και το δικό του στιγμιότυπο της κλάσης ApiContext.

Η ανάπτυξη της έγκειται στα παρακάτω βήματα:

- Προσδιορισμός των μοντέλων που επιθυμούμε να αποτυπώσουμε στην βάση.
- Χρήση DbSet του κάθε μοντέλου για αναπαράσταση των βασικών πινάκων της βάσης.
- Χρήση της συνάρτησης OnModelCreating για αναπαράσταση των πινάκων που αποτυπώνουν τις συσχετίσεις των οντοτήτων μεταξύ τους.

Οι ρυθμίσεις που ορίζουμε στην OnModelCreating προσδιορίζουν τις σχέσεις εξάρτησης μεταξύ των οντοτήτων, τις ιδιότητες κάθε οντότητας που αντιστοιχούν στην αντίστοιχη συσχετιζόμενη οντότητα, τις στήλες του κάθε πίνακα στις οποίες θα υπάρχουν τα ξένα κλειδιά στην βάση καθώς και λοιπές ρυθμίσεις της βάσης και του EFCore. Για παράδειγμα η παρακάτω εντολή ορίζει μία συσχέτιση 1-1 ανάμεσα στον χρήστη και τα πιθανά κρυφά του δεδομένα μέσω της ιδιότητας HideableInfo του χρήστη και της στήλης "UserId" του πίνακα RegularUserHideableInfo:

```
modelBuilder.Entity<RegularUser>()
    .HasOne( u => u.HideableInfo )
    .WithOne()
    .HasForeignKey<RegularUserHideableInfo>("UserId")
    .IsRequired();
```

Με παρόμοιο τρόπο ορίζονται και οι υπόλοιπες συσχετίσεις.

Οι ιδιότητες τύπου DbSet μπορούν είτε να είναι private, και τα ερωτήματα να ορίζονται εσωτερικά σε μεθόδους του, είτε public και να δίνεται η ελευθερία στα Services να κάνουν τα ερωτήματα που χρειάζονται με τις παραμέτρους που διαθέτουν. Επιλέγχθηκε η δεύτερη προσέγγιση, καθώς αυτή είναι η πιο ευραίως χρησιμοποιούμενη.

3.3 Ανάπτυξη Services

Τα Services αποτελούν την υλοποίηση μίας διεπαφής ανάμεσα στις μεθόδους του κάθε Controller και στα στιγμιότυπα της κλάσης ApiContext. Στην πράξη, Τα Controllers δέχονται μέσω έγχυσης εξαρτήσεων τα στιγμιότυπα των Services που χρειάζονται και τα χρησιμοποιούν για να εκτελέσουν τις λειτουργίες τους. Τα Services από την πλευρά τους δέχονται το ανάλογο στιγμιότυπο ApiContext καθώς και άλλα πιθανά services με τον ίδιο τρόπο.

Στην ανάπτυξη των services, προσπαθήσαμε να ακολουθήσουμε καθαρά ιεραρχική δομή, χωρίς κυκλικές εξαρτήσεις μεταξύ τους. Για κάθε Service ορίζεται ένα Interface από τις μεθόδους που υποχρεώνεται να προσφέρει. Ύστερα αναπτύσεται μία κλάση που υλοποιεί το Interface. Τέλος, κατοχυρώνεται κατά την διαδικασία κτισίματος της εφαρμογής ότι θέλουμε να δημιουργηθεί υπηρεσία τέτοιου είδους. Επιλέξαμε Scoped Lifetime και για τα Services, καθώς έχουν σε γενικότερο βαθμό καλύτερες επιδόσεις χρόνου και χώρου από τα Transient Services αλλά ταυτόχρονα τα Singleton Services δεν μπορούν να χρησιμοποιήσουν Scoped APiContext.

Για να επιλέξουμε τι Services θα κατασκευάσουμε, φτιάξαμε αρχικά 1 Service για κάθε οντότητα με βασικές CRUD λειτουργίες της οντότητας αυτής, και ύστερα χτίσαμε Services "δεύτερου επιπέδου" που διαπράττουν ενέργειες που αφορούν πολύ συγκεκριμένες λειτουργίες που χρησιμοποιούν πολλαπλές οντότητες (όπως πχ η παραγωγή προτάσεων αγγελιών ή παραγωγή και επικύρωση JWT), και πιθανό να χρησιμοποιούν και Services του χαμηλότερου επιπέδου.

3.3.1 LINQ

Σχεδόν κάθε μέθοδος της υλοποίησης λειτουργεί ελέγχοντας την ορθότητα των παραμέτρων της και ύστερα χτίζοντας και εκτελώντας ένα ερώτημα LINQ στην βάση.

Τα ερωτήματα LINQ είναι η μέθοδος με την οποία το σύστημα της .NET πραγματοποιεί ερωτήματα στην βάση. Ουσιαστικά, κλάσεις συλλογών που υλοποιούν το Interface IEnumerable ή το IQueryable υποστηρίζουν ορισμένες

συναρτήσεις πάνω στα στιγμιότυπα τους. Όταν αυτές οι μεθόδοι εφαρμόζονται πάνω σε IEnumerable, παράγουν σαν αποτέλεσμα μία παραλλαγή της συλλογής δεδομένων. Αλλά όταν εφαρμόζονται σε IQueryable, παράγουν ένα ερώτημα το οποίο μπορεί να εκτελεστεί για να παραγχθεί μία συλλογή δεδομένων εκ νέου. Η κλάση DbSet υλοποιεί την κλάση IQueryable με τρόπο τέτοιο ώστε η εκτέλεση ερωτημάτων LINQ πάνω σε στιγμιότυπα της να επιστρέφει πίσω συλλογές οντοτήτων που έχουν κατασκευαστεί με δεδομένα της βάσης. Η προσέγγιση αυτή προσφέρει Type Safety στα ερωτήματα ακόμα και στο στάδιο της μεταγλώττισης του προγράμματος.

Τα ερωτήματα LINQ μπορούν να εκφραστούν με 2 συντάξεις. Αν για παράδειγμα θέλουμε να δουμε μία συλλογή από τους :

- Σύνταξη ερωτήματος:

```
var query = from connection in context.Connections
            where connection.Timestamp.Day == (int)DayOfWeek.Thursday
            select connection.SentTo;
```

- Συναρτησιακή Σύνταξη:

```
var query = context.Connections
            .Where( con => con.Timestamp.Day == (int) DayOfWeek.Thursday)
            .Select(con => con.SentTo);
```

Για εκτέλεση του ερωτήματος μπορούμε να ζητήσουμε το είδος συλλογής που θέλουμε (λίστα, πίνακα) με κληση στην αντίστοιχη μέθοδο .To...() ή τοποθετώντας το μέσα σε collection expression ([.. query]).

Για τα περισσότερα ερωτήματα επιλέξαμε την δεύτερη σύνταξη καθώς είναι πιο συμβιβάσιμη και κατανοητή βάσει του συντακτικού της γλώσσας.

Στην πράξη, το EFCore αναλύει τα ερωτήματα που παράγονται και τα μετατρέπει σε Expression Trees, μία ενδιάμεση αναπαράσταση του ερωτήματος η οποία μπορεί πιο εύκολα να μεταφραστεί σε ερώτημα SQL. Για αυτόν τον λόγο η κάθε συνάρτηση που δέχεται σαν όρισμα μία μέθοδος LINQ σε DbSet πρέπει να είναι ακριβώς 1 έκφραση χωρίς σύμβολα και συναρτήσεις που δεν μπορούν να αποτιμηθούν από την βάση δεδομένων. Σε κανονικές συνθήκες, τα πεδία συσχετίσεων σε ερωτήματα βάσης πρέπει να ορίζονται ρητά μέσω της μεθόδου .Include (πχ, για να εμφανιστεί στο αποτέλεσμα ερωτήματος ο χρήστης ο οποίος έστειλε κάποιο αίτημα διασύνδεσης, θα πρέπει να κάναμε context.Connections.Include(con =¿ con.SentBy)...). Χρησιμοποιήσαμε το πακέτο **Microsoft.EntityFrameworkCore.Proxies**, το οποίο επιτρέπει LazyLoading όλων των virtual μελών των οντοτήτων για να αποφύγουμε αυτή την κατάσταση.

3.4 Ανάπτυξη Controllers

Τα Controllers στην .NET αποτελούν το σημείο έναρξης των επικυρωμένων και εξουσιοδοτημένων αιτημάτων HTTP/HTTPS που φτάνουν στο νωτιαίο άκρο. Τα Controllers χρησιμοποιούν τα Services για να παράγουν τα αποτελέσματα των HTTP αιτημάτων. Η ανάπτυξη τους γίνεται κυρίως μέσω Attributes. Όλα τα Controller έχουν ως υπέρκλαση την κλάση ControllerBase και είναι Annotated ως με το Attribute [Controller].

Αποφασίσαμε τα Controller να λειτουργούν ανεξάρτητα το ένα από το άλλο, και να μπορούν να στείλουν δεδομένα από ένα Service σε άλλο για να παράγουν τα αποτελέσματα τους. Επίσης αποφασίσαμε να χρησιμοποιήσουμε κυρίως Routing parameters και Request Bodies για την μεταφορά δεδομένων προς τα Controllers. Δημιουργήσαμε ένα Controller για κάθε βασική οντότητα, και ένα ακόμα για διαχείριση Login και Register. Τα endpoints που δημιουργούν τα Controllers παράγουν απαντήσεις και δέχονται Request Bodies μόνο σε μορφή JSON. Μπορούν όμως να παράγουν δεδομένα μορφής XML αν περαστεί σε εκείνα η παράμετρος ερωτήματος **format=xml**.

3.4.1 Ταυτοποίηση

Για ταυτοποίηση του χρήστη χρησιμοποιήσαμε Json Web Token με τα παρακάτω Claims (εκτός Issuer και Audience):

- emailaddress: το email του χρήστη που συνδέεται
- sid: το Id του χρήστη στην βάση
- role: ο ρόλος του χρήστη (admin ή user)

Ως αλγόριθμος hashing χρησιμοποιήθηκε ο HmacSha256 και για secret χρησιμοποιήσαμε μία παράμετρο που ορίζεται στο settings αρχείο της εφαρμογής. Τα JWT έχουν διάρκεια 4 ώρες από την στιγμή που εκδόθηκαν. Όλα τα αιτήματα εκτός από το Login και το Register απαιτούν χρήση έγκυρου JWT στο Authorisation Header του αιτήματος, με την μορφή "Bearer token", ειδάλλως επιστρέφουν

3.4.2 Εξουσιοδότηση

Όπως προαναφέρθηκε, χρησιμοποιήσαμε εξουσιοδότηση βασισμένη σε πολιτική. Ο τρόπος με τον οποίο εφαρμόζεται η κάθε πολιτική βασίζεται στις 2 κλάσεις που ορίζουν τις παραλλαγές της, η καθεμία απο αυτές πρέπει υλοποιεί να την αντίστοιχη διεπαφή:

- IAuthorizationRequirement: Ορίζει τις παραμέτρους της πολιτικής, δηλαδή τα δεδομένα που θα χρησιμοποιήσει για να ορίσει αν ο χρήστης είναι εξουσιοδοτημένος ή όχι.
- IAuthorizationHandler: Ορίζει τον τρόπο με τον οποίο η πολιτική κρίνει την εξουσιοδότηση του χρήστη χρησιμοποιώντας τα δεδομένα του αντίστοιχου IAuthorizationRequirement.

Τέλος, κάθε πολιτική χρειάζεται και το δικό της μοναδικό όνομα. Να σημειωθεί ότι κάθε στιγμιότυπο της κλάσης IAuthorizationRequirement ανήκει σε διαφορετική πολιτική, πάντα. Για παράδειγμα, αν θέλουμε ενα αίτημα να κάνει ακριβώς τον ίδιο έλεγχο σε κάποια παράμετρο route με ένα άλλο, με διαφορά μόνο το όνομα της παραμέτρου, κάτι που χρειάστηκε στην εφαρμογή μας, στην εξουσιοδότηση καθαρά βασισμένη σε πολιτική πρέπει να φτιαχτεί μία καινούρια πολιτική που χρησιμοποιεί τις ίδιες κλάσης Requirement και Handler αλλά διαφορετικό στιγμιότυπο του Requirement.

Σε γενικές γραμμές, οι πολιτικές που χρησιμοποιήσαμε ήταν:

- HasUserIdParamEqualTo: Ο χρήστης πρέπει να έχει Id ίσο με την παράμετρο userId του route του αιτήματος
- HasNotificationPolicy: ο χρήστης πρέπει να έχει την ειδοποίηση στην οποία αναφέρεται το αίτημα.
- SentMessagePolicy: ο χρήστης πρέπει να έχει στείλει το μήνυμα στο οποίο αναφέρεται το αίτημα.
- SentConnectionRequestPolicy: ο χρήστης πρέπει να έχει στείλει το αίτημα για την σύνδεση στην οποία αναφέρεται το αίτημα.
- ReceivedConnectionRequestPolicy: ο χρήστης πρέπει να έχει σταλθεί το αίτημα για την σύνδεση στην οποία αναφέρεται το αίτημα.
- CreatedJobPolicy: Ο χρήστης πρέπει να έχει αναρτήσει την αγγελία πάνω στην οποία αναφέρεται το αίτημα.
- CreatedPostPolicy: Ο χρήστης πρέπει να έχει αναρτήσει το άρθρο πάνω στο οποίο αναφέρεται το αίτημα.
- IsAdminPolicy: Ο χρήστης πρέπει να είναι διαχειριστής.
- IsMemberOfConversationPolicy: Ο χρήστης πρέπει να είναι έχει Id είτε ίσο με μία από 2 παραμέτρους.
- IsMemberOfConnectionPolicy: Ο χρήστης πρέπει να είναι ένας απο τους 2 χρήστες τους οποίους αφόρά η διασύνδεση.

Σε όλες τις παραπάνω πολιτικές, αν ο χρήστης είναι διαχειριστής, θεωρείται εξουσιοδοτημένος ανεξαρτήτως οποιουδήποτε άλλου περιορισμού. Παραλείπονται πολιτικές που ήταν απλές παραλλαγές της πρώτης (αλλάζει μόνο το όνομα της παραμέτρου που ελέγχεται).

3.5 Ανάπτυξη Αλγορίθμου Προτάσεων

Αναπτύξαμε τους αλγορίθμους προτάσεων βάση του αλγορίθμου Matrix Factorisation που έχετε προσφέρει στις διαφάνειες του μαθήματος. Ο αλγόριθμος υλοποιήθηκε εκ του μηδενός χωρίς την χρήση έτοιμης βιβλιοθήκης, και θεωρείται ότι συγκλίνει όταν το RMSE μείνει ίδιο ή όταν φτάσει τα 1000 iterations.

4 Ανάπτυξη Μετωπιαίου Άκρου

4.1 Επισκόπηση Δομής Έργου

- **src/components**: Περιέχει όλα τα στοιχεία του React όπως το `Navbar`, το `Chat` και το `Timeline`.
- **src/pages**: Κάθε σελίδα αντιπροσωπεύει ένα διαφορετικό μέρος της εφαρμογής (π.χ. Αρχική σελίδα, Σελίδα συνομιλιών, Σελίδα ρυθμίσεων).
- **src/context**: Αποθηκεύει το `AuthContext` για το χειρισμό του ελέγχου ταυτότητας παγκοσμίως.
- **src/services**: Περιλαμβάνει `api.js` για την υποβολή αιτημάτων HTTP στο API υποστήριξης.
- **src/utils**: Περιέχει βοηθητικά αρχεία όπως `auth.js` και `validations.js`.

4.2 Εκτέλεση της Εφαρμογής

4.2.1 Navbar Component

Το `Navbar` προσαρμόζει δυναμικά τα περιεχόμενά του με βάση το αν ο χρήστης είναι συνδεδεμένος ή εκτός σύνδεσης, καθώς και με βάση τον ρόλο του χρήστη (διαχειριστής ή κανονικός χρήστης). Οι λειτουργίες σύνδεσης και αποσύνδεσης αντιμετωπίζονται απευθείας σε αυτό το στοιχείο.

4.2.2 Network Page

Αυτή η σελίδα δίνει τη δυνατότητα στους συνδεδεμένους χρήστες να διαχειρίζονται τις επαγγελματικές τους συνδέσεις και να αναζητούν νέους επαγγελματίες.

Σημαντικές σημειώσεις:

- Οι χρήστες μπορούν να δουν προφίλ και να ξεκινήσουν συνομιλίες με άλλους επαγγελματίες.
- Αυτή η σελίδα ενσωματώνεται απευθείας με το backend για την ανάκτηση και εμφάνιση του δικτύου ενός χρήστη.

4.2.3 Admin Dashboard

Η διεπαφή διαχειριστή επιτρέπει την εξαγωγή δεδομένων χρήστη είτε σε μορφή JSON είτε σε μορφή XML. Παρέχει μια λεπτομερή επισκόπηση των δεδομένων χρήστη, συμπεριλαμβανομένων των αναρτήσεων, των ενδιαφερόμενων αναρτήσεων, των θέσεων εργασίας που έχουν δημοσιευτεί και των ενδιαφερόμενων θέσεων εργασίας.

Σημαντικές σημειώσεις:

- Μόνο χρήστες με δικαιώματα διαχειριστή έχουν πρόσβαση σε αυτήν τη σελίδα.
- Οι διαχειριστές μπορούν να προβάλλουν τα στοιχεία χρήστη και να εξάγουν επιλεγμένα δεδομένα χρήστη για διαχειριστικούς σκοπούς.

4.2.4 Chats Page

Το `ChatsPage` είναι η κύρια διεπαφή ανταλλαγής μηνυμάτων για την επικοινωνία των χρηστών σε πραγματικό χρόνο.

Λειτουργικότητα:

- Εμφανίζει όλες τις συνομιλίες στην αριστερή πλαϊνή γραμμή.
- Τα μηνύματα λαμβάνονται κάθε 2 δευτερόλεπτα για να διατηρείται η συνομιλία ενημερωμένη.
- Η συνομιλία μεταβαίνει αυτόματα στο πιο πρόσφατο μήνυμα, αλλά ανανεώνεται μόνο εάν εντοπιστούν νέα μηνύματα συγκρίνοντας αναγνωριστικά μηνυμάτων.
- Τα μηνύματα που αποστέλλονται από τον συνδεδεμένο χρήστη ευθυγραμμίζονται προς τα δεξιά και τα ληφθέντα μηνύματα ευθυγραμμίζονται προς τα αριστερά.

-Κλήσεις Backend:

- Οι συνομιλίες λαμβάνονται από το `/api/Message/chat/members/{userId}`.
- Τα μηνύματα συνομιλίας λαμβάνονται από το `/api/Message/chat/{loggedInUserId}/{recipientId}`.
- Τα νέα μηνύματα αποστέλλονται μέσω του `/api/Message/send/{loggedInUserId}/{recipientId}`.

4.2.5 Notifications Page

Η Σελίδα Ειδοποιήσεων επιτρέπει στους χρήστες να προβάλλουν και να διαχειρίζονται τόσο αιτήματα σύνδεσης όσο και αναρτήσεις ειδοποιήσεων.

Λειτουργικότητα:

- Τα αιτήματα σύνδεσης μπορούν να γίνουν δεκτά ή να απορριφθούν από τη διεπαφή.
- Οι ειδοποιήσεις σχετικά με τις αναρτήσεις μπορούν να επισημανθούν ως αναγνωσμένες.
- Οι ειδοποιήσεις που διαβάζονται εμφανίζονται ξεθωριασμένες για να υποδείξουν ότι έχουν υποβληθεί σε επεξεργασία.

-Κλήσεις Backend:

- Τα αιτήματα σύνδεσης λαμβάνονται από το `/api/Connection/received/{userId}`.
- Οι ειδοποιήσεις λαμβάνονται από το `/api/Notification/my/{userId}`.
- Οι ενέργειες σύνδεσης (αποδοχή/απόρριψη) αντιμετωπίζονται μέσω `/api/Connection/accept/{requestId}` και `/api/Connection/decline/{requestId}`.
- Οι ειδοποιήσεις επισημαίνονται ως αναγνωσμένες μέσω του `/api/Notification/read/{notificationId}`.

5 Επίλογος

5.1 Δυσκολίες και τρόποι αντιμετώπισης

5.1.1 Δυσκολία 1: Η C# δεν έχει native υποστήριξη για διαχείριση πινάκων μαθηματικών

Λόγω της απουσίας τέτοιας υποστήριξης, έπρεπε να αναπτύξουμε όλες τις πράξεις των πινάκων από την αρχή.

5.1.2 Δυσκολία 2: Η ανάγκη για Joins σε ερωτήματα LINQ της EFCore

Η ανάγκη για συνεχή χρήση των μεθόδων του στυλ .Include στα ερωτήματα ήταν κουραστική και έκανε τον κώδικο των ερωτημάτων πολύ δυσανάγνωστο για περίπλοκο ερωτήματα, ακόμα και αν προσφέρει καλύτερες επιδόσεις. Για να λύσουμε το πρόβλημα αυτό, χρησιμοποιήσαμε το Module των Proxies που επιτρέπει οι ιδιότητες συσχετίσεων που αφορούν άλλες οντότητες να φορτώνονται μέσω Lazy Loading.

5.1.3 Δυσκολία 3: Το σύστημα των Policies στην .NET δεν παρέχει φυσικό τρόπο παραμετροποίησης βάσει του αιτήματος

Οι παράμετροι μίας πολιτικής θέτονται κατά το κτίσιμο της εφαρμογής και, χωρίς overhaul όλων των προεπιλογών του συστήματος εξουσιοδότησης βάσει πολιτικής, δεν μπορούν να παραμετροποιηθούν ανάλογα με τις ανάγκες του κάθε αιτήματος, που μας οδήγησε στο να φτιάχνουμε διάφορες πολύ παρόμοιες πολιτικές για το κάθε αίτημα.

5.2 Σημεία Σύγκρουσης με την εκφώνηση

Παρκάτω θα βρείτε όλα τα σημεία της εφαρμογής τα οποία ήρθαν σε σύγκρουση με την εκφώνηση επειδή αποτύχαμε να τα υλοποιήσουμε.

- Το μετωπιαίο άκρο κάνει HTTP και όχι HTTPS αιτήματα στο νωτιαίο.
- Το νωτιαίο άκρο δεν υποστηρίζει αποθήκευση απολύτως ακουστικών πολυμέσων (Υποστηρίζει δηλαδή μόνο βίντεο και εικόνες).
- Το μετωπιαίο άκρο δεν υποστηρίζει ανέβασμα πολυμέσων στο νωτιαίο άκρο.