ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Art Portal API Service

Του φοιτητή
Νικόλαου Βουλγαρίδη
Αρ. Μητρώου: 164850

Επιβλέπων
Μιχαήλ Σαλαμπάσης
Βαθμίδα: Καθηγητής

Ημερομηνία …07-01-2024…

Τίτλος Δ.Ε. Backend: Ανάκτηση και Επιμέλεια (Curation) Δεδομένων για Καλλιτεχνικό Portal
Κωδικός Δ.Ε. #23154
Ονοματεπώνυμο φοιτητή/των: Νικόλαος Βουλγαρίδης
Ονοματεπώνυμο εισηγητή: Μιχαήλ Σαλαμπάσης
Ημερομηνία ανάληψης Δ.Ε. 17-03-2023
Ημερομηνία περάτωσης Δ.Ε. 07-01-2024

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

To my dearest friend in the vast expanse of Canada, whose friendship has transcended miles and time zones. Your unwavering support has been a guiding light, illuminating even the darkest corners of this academic journey. In moments of triumph and challenges, your virtual presence has been a constant reminder that true friendship knows no boundaries.

To my mother, my unwavering pillar of strength and the embodiment of selfless love. Your sacrifices, guidance, and boundless affection has been the driving force behind my endeavors. This achievement is a reflection of your enduring influence on my life.

With heartfelt gratitude,

Nikolaos

Πρόλογος

Η επιλογή μου προς τη συγκεκριμένη εργασία πηγάζει από τον ενθουσιασμό μου για την ανάπτυξη εφαρμογών, κυρίως στον τομέα του backend, καθώς και από την επιθυμία μου να εξελιχθώ σε έναν ικανό backend developer στην επαγγελματική μου πορεία.

Καθώς προχωρούσα στην πανεπιστημιακή μου πορεία, και μέσω του προγράμματος Erasmus παρακολούθησα μαθήματα για τα web services και τον προγραμματισμό τους. Ήταν ακριβώς αυτό που με καθήλωσε ολοκληρωτικά και αποτέλεσε το κίνητρο για το επόμενο βήμα στην εξέλιξη της καριέρας μου.

Μετά το πέρας του προγράμματος, αντιλήφθηκα πόσο κρίσιμος είναι ο ρόλος του backend στη δημιουργία σύγχρονων και αποδοτικών εφαρμογών. Συνειδητοποίησα ότι το backend είναι αυτό που δημιουργεί τη σύνδεση μεταξύ των χρηστών και δυνατοτήτων μιας εφαρμογής, είναι η αόρατη υποδομή που κρύβεται πίσω από κάθε απλή λειτουργία που λαμβάνουμε ως δεδομένη.

Κάθε πρόκληση στον κόσμο του backend αντιμετωπίζεται με πάθος και αφοσίωση. Καθώς εξελίσσομαι, διαμορφώνοντας και προσαρμόζοντας τις ικανότητές μου, προσδοκώ να δημιουργήσω πρωτοποριακές λύσεις που θα συμβάλουν στη συνεχή εξέλιξη του τομέα. Ο δρόμος μου είναι μια διαρκής αναζήτηση για αποδοτική και καινοτόμο προγραμματιστική δημιουργία.

Περίληψη

Η παρούσα διπλωματική εργασία εμβαθύνει στη ανάπτυξη ενός μοντέρνου backend, χρησιμοποιώντας το ευέλικτο framework ASP.NET. Το έργο παρέχει μια ισχυρή υποδομή σχεδιασμένο για την αποτελεσματική διαχείριση διαφόρων λειτουργιών στον τομέα του θεάτρου.

Οι βασικές λειτουργίες που περιέχονται στο API είναι οι ακόλουθες:

- Διαχείριση ηθοποιών και τοποθεσιών: Το σύστημα επιτρέπει την προσθήκη και την προβολή νέων ηθοποιών, εμπλουτίζοντας της πλατφόρμα με νέα ανερχόμενα ταλέντα. Επιπλέον, οι τοποθεσίες όπου διεξάγονται οι παραστάσεις/εκδηλώσεις, ενσωματώνονται απρόσκοπτα για την ολοκληρωμένη διαχείριση των εκδηλώσεων.

- Διαχείριση εκδηλώσεων και παραγωγής: Η υπηρεσία υποστηρίζει την απρόσκοπτη προσθήκη και ανάκτηση δεδομένων που αφορούν εκδηλώσεις και παραγωγές, επιτρέποντας ταυτόχρονα τη λεπτομερή καταγραφή της συμβολής των ηθοποιών σε πολιτιστικά δρώμενα.

- Διαχείριση διοργανωτών: Ένα ειδικό σύστημα διαχείρισης διοργανωτών, το οποίο επιτρέπει την προσθήκη νέων διοργανωτών και την προβολή των ήδη εγγεγραμμένων. Το εν λόγω χαρακτηριστικό ενισχύει την ικανότητα της πλατφόρμας να συσχετίζει τις εκδηλώσεις με τους αντίστοιχους διοργανωτές τους, προωθώντας τον αποτελεσματικό συντονισμό των εκδηλώσεων.

- Αποθήκευση δεδομένων: Η υπηρεσία διαχειρίζεται ένα ευρύ φάσμα δεδομένων, συμπεριλαμβανομένων των λεπτομερειών των εκδηλώσεων, των πληροφοριών παραγωγών, των συνεισφορών των ηθοποιών, των στοιχείων των χώρων διεξαγωγή εκδηλώσεων, των δεδομένων των διοργανωτών και των προφίλ των χρηστών.

- Αυθεντικοποίηση και εξουσιοδότηση χρηστών: Η υπηρεσία υποστηρίζει βασικές λειτουργίες χρηστών, όπως η εγγραφή, η σύνδεση και άλλες, εκδίδοντας JSON Web Tokens (JWT) για ασφαλή έλεγχο αυθεντικοποίησης.

- Administrative λειτουργίες: Οι συντονιστές έχουν τη δυνατότητα να εισάγουν νέα δεδομένα, με αυτόματο έλεγχο που παρέχεται από μια υπηρεσία καθαρισμού δεδομένων για τη διασφάλιση της ακεραιότητας των δεδομένων.

Η υλοποίηση αναδεικνύει τις αρχές της τμηματικότητας, της συντηρησιμότητας και της επεκτασιμότητας, καταλήγοντας σε μια συνεκτική και προσαρμόσιμη αρχιτεκτονική του έργου και του API.

Η εργασία αυτή υπογραμμίζει την καταλληλότητα και την ευελιξία του πλαισίου ASP.NET για την κατασκευή ενός προηγμένου backend service, προσαρμοσμένο στις περίπλοκες απαιτήσεις που σχετίζονται με ένα σύστημα διαχείρισης θεατρικών παραστάσεων.

Backend: Data Retrieval and Curation for Artistic Portal

Nikolaos Voulgaridis

Abstract

The focus of this thesis is to create a modern backend service using the versatile ASP.NET framework. The project aims to establish a comprehensive infrastructure that can efficiently manage various operations in the theater sector.

The main functions contained in the API are the following:

- Management of Actors and Locations: The service allows for the addition and viewing of new actors, enriching the platform with either new up-and-coming talents or not yet added talents.

- Event and Production Management: The service allows for seamless addition and retrieval of data related to events and productions while also recording detailed actor contributions to cultural events.

- Organizer Management: A management system for organizers that allows for adding and viewing registered organizers. This feature promotes efficient event coordination by associating events with their respective organizers.

- Data Handling: The service manages a wide range of data, including event details, production information, actor contributions, venue details, organizer data, and user profiles.

- User Authentication and Authorization: The service supports basic, but not only, user operations such as registration and login using JSON Web Tokens (JWT) for secure authentication.

- Administrative functions: Moderators can import new data. Imported data pass by a data cleansing service to ensure data integrity.

The implementation highlights the principles of modularity, maintainability, and scalability, resulting in a consistent and adaptable project and API architecture.

This project emphasizes the suitability and flexibility of the ASP.NET framework for building an advanced backend service tailored to the complex requirements associated with a theatrical management system.

Ευχαριστίες

Θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη προς τη μητέρα μου, της οποίας η αμέριστη υποστήριξη με συνόδευε σε όλη την διάρκεια της ακαδημαϊκής μου πορείας. Η ενθάρρυνση και η αγάπη της ήταν η κινητήρια δύναμη πίσω από τα επιτεύγματά μου.

Ένα ιδιαίτερο ευχαριστώ στην ARX.NET για τις γνώσεις που αποκόμισα κατά τη διάρκεια της πρακτικής μου άσκησης. Οι γνώσεις και οι δεξιότητες που αποκτήθηκαν μέσω της συνεργασίας μας ήταν καθοριστικές για τη διαμόρφωση των αποτελεσμάτων αυτής της εργασίας. Η δέσμευσή τους στην εκπαίδευση και την καινοτομία έπαιξε καθοριστικό ρόλο στην ακαδημαϊκή μου εξέλιξη.

Περιεχόμενα

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
JWT	JSON Web Token
API	Application Programming Interface
DTO	Data Transfer Object
EF	Entity Framework
ORM	Object-Relational Mapping

Κεφάλαιο 1ο: Δομή της Εφαρμογής

1.1 Εισαγωγή

Η εφαρμογή αποτελείται από τέσσερα ξεχωριστά projects, εκ των οποίων τρία είναι βιβλιοθήκες κλάσεων (class libraries), και ένα είναι το κεντρικό project εκκίνησης, το οποίο περιλαμβάνει τους controllers, οι οποίοι είναι τα API endpoints μέσω των οποίων οι χρήστες μπορούν να στείλουν αιτήματα και να λάβουν απαντήσεις από την εφαρμογή. Το όνομα του έργου/εφαρμογής είναι "TheatricalPlays".

1.2 Αρχιτεκτονική του Έργου

Η αρχιτεκτονική του TheatricalPlays σχεδιάστηκε με γνώμονα της επίτευξη μιας σταθερής, ευέλικτης και επεκτάσιμης εφαρμογής. Το έργο ακολουθεί μια πολυεπίπεδη αρχιτεκτονική, στηριζόμενη σε διάφορα class libraries, κάθε ένα αναλαμβάνοντας ένα συγκεκριμένο ρόλο προκειμένου να διευκολύνει την ανάπτυξη και τη συντήρηση του κώδικα.

Τα βασικά στρώματα περιλαμβάνουν:

Theatrical.Api: Στο στρώμα αυτό βρίσκεται το Presentation Layers, που περιλαμβάνει τους controllers και τα API Endpoints για την αλληλεπίδραση με τους χρήστες μέσω της διεπαφής του swagger.

Theatrical.Services: Στο Business Logic Layer, περιέχονται όλες οι υπηρεσίες και η επιχειριματική λογική που χρησιμοποιούνται για τη διαχείριση των λειτουργιών της εφαρμογής.

Theatrical.Data: Αυτή η βιβλιοθήκη κλάσεων περιέχει τα κύρια μοντέλα κλάσεων, και την αλληλεπίδραση με την βάση δεδομένων.

Theatrical.Dto: Αυτή η βιβλιοθήκη κλάσεων περιέχει όλα τα μοντέλα μεταφοράς δεδομένων.

1.3 Δομή

Η δομή της εφαρμογής ορίζεται από τα παρακάτω projects:

1.3.1 Theatrical.Api

Το Theatrical.Api αναδεικνύεται ως το βασικό project εκκίνησης. Περιλαμβάνει τους controllers που αποτελούν το κύριο σημείο επαφής για τα αιτήματα των χρηστών και τη διαχείριση των λειτουργιών της εφαρμογής. Επίσης εκεί υπάρχει το πρόγραμμα σημείο εκκίνησης (Program.cs) όπου πραγματοποιείται η εγγραφή όλων των υπηρεσιών που έχουν δημιουργηθεί και που χρησιμοποιούνται στην εφαρμογή.

1.3.2 Theatrical.Data

Το Theatrical.Data αποτελεί μια βιβλιοθήκη κλάσεων που φιλοξενεί το Database Context της εφαρμογής, συμπεριλαμβανομένου του Fluent API για την καθοδήγηση της διαμόρφωσης της βάσης δεδομένων. Επιπλέον, περιέχει τα κύρια μοντέλα κλάσεις που αντιπροσωπεύουν τη δομή των

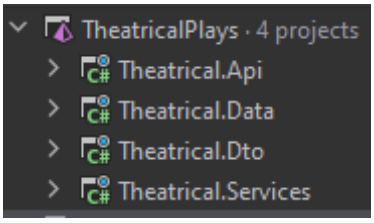
δεδομένων στην εφαρμογή μας. Μέσω αυτής της βιβλιοθήκης, η εφαρμογή αλληλεπιδρά με τη βάση δεδομένων, εφαρμόζοντας τις αναγκαίες διαδικασίες για τη διαχείριση και ανάκτηση των δεδομένων.

1.3.3 Theatrical.Dto

Το Theatrical.Dto αποτελεί μια βιβλιοθήκη κλάσεων το οποίο προσφέρει τα Data Transfer Objects (DTOs) που χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ των διάφορων στρωμάτων της εφαρμογής.

1.3.4 Theatrical.Services

Το Theatrical.Services αποτελεί μια βιβλιοθήκη κλάσεων που φιλοξενεί όλες τις υπηρεσίες και την επιχειρηματική λογική που χρησιμοποιείται στην εφαρμογή μας. Εδώ συγκεντρώνεται η κύρια λειτουργικότητα της εφαρμογής, καθορίζοντας πώς διαχειρίζονται τα δεδομένα, πώς επιτελούνται οι διεργασίες, και πώς αλληλεπιδρούν οι χρήστες με το σύστημα.



Σχήμα 1.1: Απεικόνιση των έργων μέσα από το προγραμματιστικό περιβάλλον.

1.4 Συνεργατικότητα με άλλες Εφαρμογές

Η εφαρμογή ως μία REST API backend εφαρμογή, έχει σχεδιαστεί με προοπτική στενής ενσωμάτωσης και αλληλεπίδρασης με άλλες εφαρμογές, συνδυάζοντας λειτουργικότητα και ευελιξία. Οι εφαρμογές που συνεργάζεται, περιγράφονται παρακάτω και περιέχουν εφαρμογές τελικών χρηστών και εφαρμογές διαχείρισης περιεχομένου.

1.4.1 Συμβατότητα με Web App, iOS, Android

Η εφαρμογή ενσωματώνεται χρησιμοποιείται από μία ολοκληρωμένη web app εφαρμογή, καθώς και από δύο εφαρμογές για κινητά, μία android και μία iOS εφαρμογή, οι οποίες βρίσκονται σε στάδιο εξέλιξης. Οι οποίες καταναλώνουν το API της εφαρμογής για να αποστέλλουν αιτήματα και να λαμβάνουν απαντήσεις, ενισχύοντας έτσι την ολοκληρωμένη λειτουργικότητα και τη συνεργασία μεταξύ των διαφορετικών πλατφορμών.

Οι χρήστες έχουν την δυνατότητα να πραγματοποιούν εγγραφή, σύνδεση, να ενημερώσουν το προφίλ τους, και να εκτελούν αναζητήσεις για ηθοποιούς, παραστάσεις και άλλες λειτουργίες.

1.4.2 Συμβατότητα με μία WebScraping εφαρμογή

Η εφαρμογή συνεργάζεται επίσης με μία υπό εξέλιξη εφαρμογή WebScraping, συνδυάζοντας τη δυνατότητα ανάκτησης δεδομένων από διάφορες πηγές στο διαδίκτυο. Η υπό εξέλιξη εφαρμογή WebScraping επιτρέπει την αυτοματοποιημένη συλλογή πληροφοριών που μπορούν να εμπλουτίζουν το περιεχόμενο της εφαρμογής backend, με νέους ηθοποιούς, παραστάσεις και άλλα δεδομένα.

Μέσω αυτής της συνεργασίας, η εφαρμογή μας διασφαλίζει ότι οι πληροφορίες που παρέχει είναι πάντα ενημερωμένες. Η συνεχής αυτή συλλογή δεδομένων επιτρέπει την προσθήκη νέου περιεχομένου, ενισχύοντας έτσι την εμπειρία των χρηστών και εμπλουτίζοντας την πλατφόρμα της εφαρμογής με ποικιλία πληροφοριών.

1.4.3 Συμβατότητα με μία Εφαρμογή Data Cleaning

Η εφαρμογή συνεργάζεται επίσης με μία υπό εξέλιξη εφαρμογή διαχείρισης περιεχομένου, η οποία έχει ως στόχο τον καθαρισμό και την απλοποίηση των δεδομένων της εφαρμογής μας.

Αυτή η συνεργασία θα επιτρέπει στην εφαρμογή διαχείρισης περιεχομένου διαχειρίζεται αποτελεσματικά τα δεδομένα που προέρχονται από τη εφαρμογή μας, προσφέροντας μια ομαλή διαδικασία καθαρισμού, ανανέωσης και οργάνωσης του περιεχομένου.

Η υπό εξέλιξη εφαρμογή διαχείρισης περιεχομένου συμβάλλει στη βελτίωση της ποιότητας των δεδομένων, ενώ ταυτόχρονα διευκολύνει την αποτελεσματική ανάκτηση και ενημέρωση των πληροφοριών της εφαρμογής μας. Με αυτόν τον τρόπο, επιδιώκεται η δημιουργία ενός ολοκληρωμένου συστήματος διαχείρισης δεδομένων, εξασφαλίζοντας τη συνέπεια και την ακρίβεια του περιεχομένου που παρέχει η εφαρμογή “TheatricalPlays”.

1.5 Επίλογος

Πέραν της ανάλυσης της αρχιτεκτονικής του έργου μας, είναι σημαντικό να εξετάσουμε επίσης τη σημασία και τον ρόλο της πολυεπίπεδης αρχιτεκτονικής στον ευρύτερο κόσμο της ανάπτυξης λογισμικού. Μέσα από την εφαρμογή αυτής της μεθοδολογίας, πετυχαίνεται όχι μόνο η βελτιστοποίηση του κώδικά μας, αλλά και η διευκόλυνση της συντηρησιμότητας και η ενίσχυση της ευελιξίας του συστήματός μας.

Καθώς εμβαθύνουμε στα επόμενα κεφάλαια, θα επισημάνουμε τις λεπτομέριες κάθε class library, αναδεικνύοντας τον ρόλο και τη συνεισφορά του καθενός στο συνολικό πλαίσιο. Θα εξετάσουμε τις βέλτιστες πρακτικές που ακολουθήθηκαν κατά την ανάπτυξη των υπηρεσιών και θα αναλύσουμε τις αποφάσεις που λάβαμε για την επίτευξη συγκεκριμένων στόχων στον τομέα της απόδοσης, της ασφάλειας και της επεκτασιμότητας.

Συνεχίζοντας, θα εξετάσουμε πιο λεπτομερώς την εφαρμογή της πολυεπίπεδης αρχιτεκτονικής στον καθορισμό της διασυνδετικότητας μεταξύ των βιβλιοθηκών κλάσεων και το πώς αυτή η δομή βοηθά στη αποφυγή σφαλμάτων και στη διατήρηση ενός καλά οργανωμένου κώδικα.

Τέλος, θα εξετάσουμε πιθανές βελτιώσεις και μελλοντικές επεκτάσεις στο σύστημα μας, εντοπίζοντας περαιτέρω πεδία ανάπτυξης.

Κεφάλαιο 2ο: Data Transfer Objects

2.1 Εισαγωγή

Στο δεύτερο κεφάλαιο της εργασίας, θα συζητηθεί η σημαντικότητα των DTOs και η σημασία τους για τη σωστή λειτουργία της εφαρμογής μας. Σε αυτό το κεφάλαιο επίσης θα εστιάσουμε στο λόγο που τα DTOs που αποτελούν θεμελιώδη στοιχεία για τη μεταφορά δεδομένων στο πλαίσιο της εφαρμογής. Επίσης θα παρουσιάσουμε αναλυτικά τα DTOs που χρησιμοποιούνται στην εφαρμογή.

2.2 Σκοπός των Data Transfer Objects (DTOs)

Τα DTOs διαδραματίζουν έναν κρίσιμο ρόλο στην επιτυχημένη λειτουργία της εφαρμογής, επιτρέποντας την ασφαλή, αποτελεσματική και συνεκτική μεταφορά δεδομένων σε όλη την αρχιτεκτονική του συστήματος.

2.2.1 Ρόλος των Data Transfer Objects (DTOs)

Ο σκοπός των DTOs είναι να διευκολύνουν τη μεταφορά δεδομένων μεταξύ διαφορετικών στρωμάτων της εφαρμογής. Αυτό γίνεται με το να παρέχουν μια κοινή γλώσσα μεταξύ των στρωμάτων, εξασφαλίζοντας ότι οι πληροφορίες μεταφέρονται ομαλά και χωρίς προβλήματα.

2.2.2 Διατήρηση Συνοχής στα Δεδομένα

Η χρήση DTOs συμβάλλει στη διατήρηση της συνοχής στα δεδομένα. Ανεξάρτητα από το πού αποθηκεύονται ή πώς χρησιμοποιούνται τα δεδομένα, τα DTOs εξασφαλίζουν ότι η μορφή και η δομή των δεδομένων παραμένουν συνεπείς.

2.2.3 Αποφυγή Υπερβολικής Πληροφορίας

Ένα από τα κύρια πλεονεκτήματα των DTOs είναι η δυνατότητα να αποφεύγουν τη μεταφορά υπερβολικής πληροφορίας. Ανάλογα με το πεδίο εφαρμογής, μπορούμε να προσαρμόσουμε τα DTOs ώστε να μεταφέρουν μόνο τα απαραίτητα δεδομένα.

2.2.4 Ασφάλεια Δεδομένων στη Βάση Δεδομένων

Τα DTOs παίζουν ένα καθοριστικό ρόλο στη διασφάλιση της ασφάλειας των δεδομένων που αποθηκεύονται στη βάση δεδομένων. Μέσω του κατάλληλα σχεδιασμένου μοντέλου DTO, μπορούμε να ελέγξουμε την ποσότητα και τον τύπο των πληροφοριών που μεταβιβάζονται προς και από τη βάση δεδομένων.

Περιορισμός Ευαίσθητων Δεδομένων: Χρησιμοποιώντας DTOs, μπορούμε να ελέγξουμε την ποσότητα ευαίσθητων πληροφοριών που μεταφέρονται ανάμεσα στο backend και τη βάση δεδομένων, επιτρέποντας μας να περιορίσουμε την πρόσβαση σε ευαίσθητα δεδομένα μόνο σε εκείνους που έχουν τα κατάλληλα δικαιώματα.

Έλεγχος Τύπου Δεδομένων: Με τη σωστή δομή DTO, εξασφαλίζουμε ότι τα δεδομένα που αποστέλλονται και λαμβάνονται είναι συνεπή και αντιστοιχούν στους καθορισμένους τύπους δεδομένων, προστατεύοντας έτσι από πιθανούς κινδύνους όπως οι επιθέσεις με βάση τον τύπο.

Προστασία από SQL Injection: Με τη χρήση DTOs, μπορούμε να εφαρμόσουμε κατάλληλα φίλτρα και επικυρώσεις, βοηθώντας στην αποτροπή επιθέσεων SQL injection και εξασφαλίζοντας ότι τα δεδομένα που εισάγονται στη βάση δεδομένων είναι ασφαλή.

2.3 Σχεδιασμός DTOs

Κατά το σχεδιασμό των Data Transfer Objects στην εφαρμογή, τηρήθηκαν ορισμένες βασικές αρχές και πρακτικές που συνέβαλαν στην συνολική απόδοση και δομή του συστήματος μας. Ακολουθήθηκαν τα παρακάτω κριτήρια:

Σαφής Αναπαράσταση Δεδομένων:

Κάθε DTO σχεδιάστηκε με στόχο τη σαφή και κατανοητή αναπαράσταση των δεδομένων που προορίζεται να μεταφέρει. Ο καθαρός ορισμός των πεδίων και η ομαδοποίησή τους με βάση τη λειτουργικότητα και την ονομασία τους επιτρέπουν στους προγραμματιστές να κατανοούν γρήγορα τον σκοπό του κάθε DTO.

Πρότυπα Ονομασίας στην Εφαρμογή:

Εφαρμόστηκαν συγκεκριμένα πρότυπα ονομασίας για τα DTOs προκειμένου να πετύχουμε αναγνωσιμότητα και συντήρηση του κώδικα. Όλα τα DTOs ακολουθούν το κεφαλαίο-πεζό πρότυπο (camel case), και προστίθεται πρόθεμα “Dto” για να είναι σαφές ότι πρόκειται για ένα Object Transfer.

Χρήση Ιεραρχίας:

Εκμεταλλευτήκαμε την ιεραρχική δομή των DTOs για να αντιμετωπίσουμε την ανάγκη για διάφορες επιπλέον πληροφορίες σε συγκεκριμένα σημεία της εφαρμογής. Για παράδειγμα, ένα “UserDto”, που επιστέφεται στην απάντηση ενός API request, μπορεί να έχει μια σχέση “UserProfilePictureDto” ώστε να παρέχει επιπλέον λεπτομέρειες σχετικά με την κύρια φωτογραφία ενός χρήστη.

Περιορισμός Όγκου Δεδομένων:

Σε κάθε περίπτωση μελετήθηκε διεξοδικά το ποια δεδομένα είναι απαραίτητα για κάθε σενάριο μεταφοράς. Αυτό επιτρέπει την μείωση του όγκου των δεδομένων που μεταφέρονται με κάθε αίτηση χρήστη, βελτιώνοντας έτσι την απόδοση του συστήματος.

2.4 DTOs στην Εφαρμογή

Σε αυτήν την ενότητα, θα εξετάσουμε τα μερικά από τα πιο σημαντικά Data Transfer Objects που χρησιμοποιούνται στην εφαρμογή. Κάθε ένα από αυτά τα DTOs έχει σχεδιαστεί με συγκεκριμένο τρόπο και συμβάλει στην αποτελεσματική επικοινωνία μεταξύ διαφορετικών στρωμάτων του συστήματος. Ας εξετάσουμε λεπτομερώς τη δομή, τον σκοπό και τον ρόλο κάθε DTO στην εφαρμογή.

Τα DTOs είναι οργανωμένα σε φακέλους μέσα στο class library “**Theatrical.Dto**” ανάλογα με το όνομά τους και τα μέρη, στην εφαρμογή που χρησιμοποιούνται.

2.4.1 LoginUserDto

Το “LoginUserDto” είναι σχεδιασμένο στο να χειρίζεται ευαίσθητες πληροφορίες χρήστη και συμβάλλει στη διαδικασία σύνδεσης των χρηστών. Συμπεριλαμβάνει τα ακόλουθα δεδομένα:

- Email: Συμβολοσειρά που αναπαριστά το email του χρήστη.
- Password: Συμβολοσειρά που αναπαριστά τον κωδικό πρόσβασης του χρήστη.

Σκοπός:

- Επιτρέπει την μεταφορά δεδομένων σύνδεσης χρήστη μεταξύ του πελάτη και του διακομιστή.
- Χρησιμοποιείται κατά τη διάρκεια της διαδικασίας πιστοποίησης για τον έλεγχο των διαπιστευτηρίων του χρήστη.

Ρόλος στην Επικοινωνία:

- Λειτουργεί ως ελαφρύς φορέας δεδομένων που προορίζεται ειδικά για τη λειτουργικότητα της σύνδεσης και το hashing του κωδικού.

```
public class LoginUserDto
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

Κώδικας 2.1: Το αντικείμενο DTO που χρησιμοποιείται κατά την σύνδεση.

2.4.2 JwtDto

Το “JwtDto” αναπαριστά τα δεδομένα που σχετίζονται με τα JWTs και την αυθεντικοποίηση. Με την επιτυχημένη αυθεντικοποίηση ο χρήστης λαμβάνει το παρακάτω αντικείμενο DTO. Η αυθεντικοποίηση και εξουσιοδότηση θα εξεταστεί στα επόμενα κεφάλαια. Παρέχει τα εξής στοιχεία:

- access_token: Το JWT χορηγείται για τον εξουσιοδοτημένο χρήστη.
- token_type: Ο τύπος του token, ο οποίος στη περίπτωση της εφαρμογής μας είναι “Bearer”.
- expires_in: Ο χρόνος ισχύος του token σε δευτερόλεπτα.

Αυτό το DTO είναι κρίσιμο για τη μεταφορά των πληροφοριών αυθεντικοποίησης ανάμεσα σε πελάτη και διακομιστή, ενισχύοντας την ασφάλεια και τη σωστή λειτουργία του συστήματος.

Το access token χρησιμοποιείται για την πρόσβαση σε προστατευόμενα API endpoints.

```
public class JwtDto
{
    public string access_token { get; set; }
    public string token_type { get; set; }
    public int expires_in { get; set; }
}
```

Κώδικας 2.2: Το αντικείμενο DTO απάντησης του Login Action.

2.4.3 UserDto

Στη συνέχεια ας μιλήσουμε για ένα ακόμη σημαντικό DTO που παρουσιάζει τα δεδομένα ενός χρήστη στο σύστημα. Το παρακάτω DTO χρησιμοποιείται σαν απάντηση μόνο όταν κάποιος χρήστης θέλει τα στοιχεία του προφίλ του. Παρέχει τα εξής στοιχεία:

- Id: Το μοναδικό αναγνωριστικό του χρήστη.
- Username: Το όνομα χρήστη του χρήστη (προαιρετικό).
- Email: Το email του χρήστη.
- EmailVerified: Ένδειξη εάν το email έχει επαληθευτεί.
- _2FA_enabled: Κατάσταση ενεργοποίησης διπαραμετρικής αυθεντικοποίησης (Two-Factor Authentication).
- Role: Ο ρόλος του χρήστη στο σύστημα.
- Transactions: Λίστα με τις συναλλαγές του χρήστη (προαιρετικό).
- Facebook, Youtube, Instagram: Συνδέσμους προς τα προφίλ του χρήστη σε κοινωνικά δίκτυα (προαιρετικοί).
- Balance: Το υπόλοιπο του χρήστη στο σύστημα.
- PerformerRoles: Λίστα με τους ρόλους καλλιτέχνη του χρήστη (προαιρετικό).
- UserImages: Λίστα με εικόνες που έχει ανεβάσει ο χρήστης (προαιρετικό).
- ProfilePhoto: Πληροφορίες σχετικά με την φωτογραφία προφίλ του χρήστη (προαιρετικό).
- BioPdfLocation: Η τοποθεσία αποθήκευσης του αρχείου Βιογραφίας του χρήστη.
- ClaimedPerson: Πληροφορίες σχετικά με το πρόσωπο που έχει δηλωθεί ο χρήστης μετά από την διαδικασία επαλήθευσης.
- ClaimedVenues: Λίστα με τα μέρη που ο χρήστης έχει δηλώσει ως δικά του.
- ClaimedEvents: Λίστα με τα events που ο χρήστης έχει δηλώσει ως δικά του.
- PhoneNumber: Ο τηλεφωνικός αριθμός του χρήστη.
- PhoneVerified: Ένδειξη αν ο τηλεφωνικός αριθμός έχει επαληθευτεί..

```
public class UserDto
{
    public int Id { get; set; }
    public string? Username { get; set; }
    public string Email { get; set; } = null!;
    public bool? EmailVerified { get; set; }
    public bool _2FA_enabled { get; set; }
    public string Role { get; set; }
    public List<TransactionDtoFetch>? Transactions { get; set; }
    public string? Facebook { get; set; }
    public string? Youtube { get; set; }
    public string? Instagram { get; set; }
    public decimal? Balance { get; set; }
    public List<string>? PerformerRoles { get; set; }
    public List<UserImagesDto>? UserImages { get; set; }
    public UserProfilePictureResponseDto? ProfilePhoto { get; set; }
    public string? BioPdfLocation { get; set; }
    public PersonDto? ClaimedPerson { get; set; }
    public List<VenueDto>? ClaimedVenues { get; set; }
    public List<EventDto>? ClaimedEvents { get; set; }
    public string? PhoneNumber { get; set; }
    public bool? PhoneVerified { get; set; }
}
```

Κώδικας 2.3: Το αντικείμενο DTO απάντησης με τις πληροφορίες του χρήστη.

2.4.4 JwtOptions

Το “JwtOptions” αποτελεί μια σημαντική κλάση στο πλαίσιο της υλοποίησης του συστήματος μας, εξυπηρετώντας τη διαδικασία δημιουργίας και επαλήθευσης των JWT (JSON Web Tokens). Αυτή η κλάση αναλαμβάνει την παραμετροποίηση του token service μας, επιτρέποντάς μας να διαχειριζόμαστε σημαντικές πτυχές της ασφάλειας των δεδομένων μας.

Με τις παραμετροποιήσεις που παρέχει, το “JwtOptions” συνεισφέρει στη σωστή λειτουργία των υπηρεσιών μας που σχετίζονται με τα JWT. Ακόμη, εξασφαλίζει την ομαλή και ασφαλή ροή πληροφοριών ανάμεσα στο σύστημα και τους χρήστες.

Αυτή η κλάση διαδραματίζει έναν κρίσιμο ρόλο κατά τα αιτήματα που υποβάλλονται από τους χρήστες σε προστατευμένα σημεία πρόσβασης (endpoints). Μέσω του JwtOptions, διασφαλίζουμε ότι τα αιτήματα αυτά είναι ασφαλή και αξιόπιστα.

Η διαδικασία αυθεντικοποίησης βασίζεται στα JWT που παράγονται και επαληθεύονται χρησιμοποιώντας τις παραμετροποιήσεις αυτής της κλάσης. Οι παράμετροι **Issuer**, **Audience**, και **SigningKey** παίζουν έναν καθοριστικό ρόλο στην επικύρωση και δημιουργία των JWT. Καθορίζοντας τον αποστολέα (Issuer), τον παραλήπτη (Audience), και το κλειδί υπογραφής (SigningKey), εξασφαλίζουμε ότι τα JWT που παράγονται είναι έγκυρα, αξιόπιστα και προστατευμένα από ανεπιθύμητες επεμβάσεις.

Συνολικά, το JwtOptions είναι αναπόσπαστο κομμάτι της ασφαλούς αρχιτεκτονικής μας, παρέχοντας την απαραίτητη υποστήριξη για την προστασία των προστατευμένων σημείων πρόσβασης από ανεπιθύμητες προσπάθειες πρόσβασης και διασφαλίζοντας τη σωστή εκτέλεση των αιτημάτων των χρηστών.

Λόγο της απλότητας χρήσης του, το παρόν αντικείμενο είναι υλοποιημένο σαν record στον κώδικα.

```
public record JwtOptions(string Issuer,
    string Audience,
    string SigningKey);
```

Κώδικας 2.4: Το record που χρησιμοποιείται στην αυθεντικοποίηση.

2.4.5 UpdateUserPhotoDto

Η κλάση αυτή χρησιμοποιείται από έναν χρήστη όταν θέλει να ανεβάσει μία φωτογραφία. Τα δεδομένα που περιέχει αυτή η κλάση επιτρέπουν στον χρήστη να θέσει μια ετικέτα/περιγραφή για την φωτογραφία που ανεβάζει και επίσης να την κάνει κύρια εικόνα προφίλ κατευθείαν.

Ιδιότητες:

Photo: Αναπαριστά τη φωτογραφία που ο χρήστης επιθυμεί να ανεβάσει και είναι υποχρεωτική.

Label: Παρέχει τη δυνατότητα προσθήκης περιγραφής για την συγκεκριμένη φωτογραφία και είναι προαιρετική.

IsProfile: Καθορίζει αν η φωτογραφία θα είναι η κύρια εικόνα προφίλ του χρήστη.

```
public class UpdateUserPhotoDto
{
    public string Photo { get; set; }
    public string? Label { get; set; }
    public bool IsProfile { get; set; }
}
```

Κώδικας 2.5: Το αντικείμενο DTO που χρησιμοποιείται για το ανέβασμα μιας φωτογραφίας χρήστη.

2.4.6 SetProfilePhotoDto

Η κλάση “SetProfilePhotoDto” χρησιμοποιείται για τον καθορισμό της φωτογραφίας προφίλ ενός χρήστη. Τα στοιχεία που παρέχει αυτή η κλάση επιτρέπουν στον χρήστη να ορίσει ποια εικόνα θα χρησιμοποιηθεί ως η εικόνα προφίλ του.

Ιδιότητες:

ImageId: Αναπαριστά το αναγνωριστικό της εικόνας που ο χρήστης επιθυμεί να ορίσει ως προφίλ και είναι υποχρεωτικό.

Label: Παρέχει τη δυνατότητα προσθήκης περιγραφής στην επιλεγμένη εικόνα.

```
public class SetProfilePhotoDto
{
    public int ImageId { get; set; }
    public string? Label { get; set; }
}
```

Κώδικας 2.6: Το αντικείμενο DTO που χρησιμοποιείται για την ενημέρωση φωτογραφίας προφίλ.

Με τον τρόπο αυτό, ο χρήστης μπορεί να επιλέξει μια εικόνα από τις διαθέσιμες, που έχει ήδη ανεβάσει, και να την ορίσει ως εικόνα προφίλ του. Επίσης παρέχεται δυνατότητα προσθήκης περιγραφής για την συγκεκριμένη φωτογραφία.

2.4.7 UpdateUsernameDto

Η κλάση “UpdateUsernameDto” χρησιμοποιείται για την ενημέρωση του ονόματος χρήστη (username) ενός χρήστη. Τα δεδομένα που περιέχει αυτή η κλάση επιτρέπουν στον χρήστη να ενημερώνει το όνομα χρήστη του στο σύστημα.

Ιδιότητες:

User: Αναπαριστά τον χρήστη για τον οποίο πρόκειται να ενημερωθεί το όνομα χρήστη. Είναι ένα αντικείμενο το οποίο παρέχει τις πληροφορίες χρήστη. Οι χρήστες δεν έχουν αλληλεπίδραση με το συγκεκριμένο πεδίο.

Username: Αναπαριστά το νέο όνομα χρήστη που θα ενημερωθεί για τον συγκεκριμένο χρήστη.

```
public class UpdateUsernameDto
{
    public User User { get; set; }
    public string Username { get; set; }
}
```

Κώδικας 2.7: Το αντικείμενο DTO που χρησιμοποιείται για του ονόματος χρήστη.

2.4.8 UpdatePasswordDto

Η κλάση “UpdatePasswordDto” χρησιμοποιείται για την αλλαγή του κωδικού πρόσβασης (password) ενός χρήστη. Τα δεδομένα που περιέχει αυτή η κλάση επιτρέπουν στον χρήστη να ενημερώσει τον κωδικό πρόσβασης του στο σύστημα.

Ιδιότητες:

Password: Αναπαριστά τον νέο κωδικό πρόσβασης που θα οριστεί για τον συγκεκριμένο χρήστη.

```
public class UpdatePasswordDto
{
    public string Password { get; set; }
}
```

Κώδικας 2.8: Το αντικείμενο DTO που χρησιμοποιείται για την ενημέρωση κωδικού ενός χρήστη.

Με τη χρήση αυτού του DTO, ο χρήστης έχει τη δυνατότητα να αλλάξει τον κωδικό πρόσβασής του, προσφέροντας ένα επιπλέον επίπεδο ασφαλείας και ελέγχου για τον προσωπικό του λογαριασμό.

2.4.9 RegisterUserDto

Η κλάση “RegisterUserDto” σχεδιάστηκε για τη διαχείριση των πληροφοριών καταχώρησης νέου χρήστη στο σύστημα. Αυτή η κλάση παρέχει τα απαραίτητα πεδία για τη δημιουργία ενός νέου λογαριασμού χρήστη κατά την εγγραφή του.

Ιδιότητες:

Email: Αναπαριστά το email του νέου χρήστη και είναι υποχρεωτικό.

Password: Αναπαριστά τον κωδικό πρόσβασης του νέου χρήστη και είναι υποχρεωτικό.

Role: Αναπαριστά το ρόλο του χρήστη, με προεπιλεγμένη τιμή 2 αν δεν καθοριστεί διαφορετικά.

```
public class RegisterUserDto
{
    [Required]
    public string Email { get; set; }

    [Required]
    public string Password { get; set; }

    public int? Role { get; set; } = 2;
}
```

Κώδικας 2.9: Το αντικείμενο DTO που χρησιμοποιείται κατά την εγγραφή χρήστη.

Η χρήση του "RegisterUserDto" βοηθά στη διασφάλιση ότι κατά την εγγραφή, τα απαραίτητα στοιχεία όπως το email και ο κωδικός πρόσβασης παρέχονται, ενώ παράλληλα παρέχεται η ευελιξία να καθοριστεί ο ρόλος του χρήστη. Με αυτό τον τρόπο, διασφαλίζουμε τη σωστή καταγραφή των πληροφοριών κατά τη διαδικασία εγγραφής νέων χρηστών στην εφαρμογή μας.

2.4.10 RegisterUserResponseDto

Ως συνέχεια του προηγούμενου, η κλάση "RegisterUserResponseDto" χρησιμοποιείται ως απάντηση μετά την επιτυχημένη εγγραφή ενός χρήστη. Οι ιδιότητες που περιλαμβάνει παρέχουν βασικές πληροφορίες για τον νέο χρήστη.

Ιδιότητες:

Id: Το μοναδικό αναγνωριστικό του χρήστη.

Email: Αναπαριστά το email του χρήστη.

Enabled: Ένδειξη κατάστασης ενεργοποίησης του λογαριασμού. Επιστρέφει false σαν υπενθύμιση ότι ένας χρήστης πρέπει να επαληθεύσει το email του.

Note: Περιέχει πιθανή σημείωση σχετικά με τον λογαριασμό του χρήστη.

```
public class RegisterUserResponseDto
{
    public int Id { get; set; }
    public string Email { get; set; }
    public bool Enabled { get; set; }
    public string? Note { get; set; }
}
```

Κώδικας 2.10: Το αντικείμενο DTO που χρησιμοποιείται σαν απάντηση, κατά την εγγραφή χρήστη.

Αυτό το DTO παρέχει συνοπτική πληροφόρηση σχετικά με τον νέο χρήστη, καθιστώντας ευκολότερη την παρακολούθηση των βασικών του στοιχείων.

2.4.11 UserImageDto

Η κλάση "UserImageDto" αποτελεί ένα κρίσιμο τμήμα του μεγαλύτερου "UserDto", παρέχοντας πληροφορίες σχετικά με τις εικόνες που σχετίζονται με έναν χρήστη. Η χρήση αυτού του DTO ενισχύει την οργάνωση και τη διαχείριση των εικόνων του χρήστη.

Ιδιότητες:

Id: Το μοναδικό αναγνωριστικό της εικόνας.

ImageLocation: Η τοποθεσία του αρχείου της εικόνας.

Label: Πιθανή ετικέτα που περιγράφει το περιεχόμενο της εικόνας (προαιρετικό).

```
public class UserImageDto
{
    public int Id { get; set; }
    public string ImageLocation { get; set; }
    public string? Label { get; set; }
}
```

Κώδικας 2.11: Το αντικείμενο DTO που χρησιμοποιείται μέσα στο "UserDto", και περιγράφει μία φωτογραφία χρήστη.

2.4.12 UserProfilePictureResponseDto

Η κλάση "UserProfilePictureResponseDto" αποτελεί ένα σημαντικό στοιχείο του μεγαλύτερου "UserDto", εστιάζοντας στην αναπαράσταση της εικόνας προφίλ ενός χρήστη. Η χρήση αυτού του DTO ενισχύει τη διαχείριση και την ανάκτηση της πληροφορίας της εικόνας προφίλ ενός χρήστη.

Ιδιότητες:

Id: Το μοναδικό αναγνωριστικό της εικόνας.

ImageLocation: Η τοποθεσία του αρχείου της εικόνας.

Label: Πιθανή ετικέτα που περιγράφει το περιεχόμενο της εικόνας (προαιρετικό).

```
public class UserProfilePictureResponseDto
{
    public int Id { get; set; }
    public string ImageLocation { get; set; }
    public string? Label { get; set; }
}
```

Κώδικας 2.12: Το αντικείμενο DTO που αντιπροσωπεύει την φωτογραφία προφίλ ενός χρήστη.

Η διαφορά με το προηγούμενο, είναι ότι αυτή δημιουργήθηκε για να κρατάει μόνο την προφίλ φωτογραφία, ενώ η προηγούμενη κρατάει όλες τις φωτογραφίες χρήστη.

Αυτό το DTO επιτρέπει την ανάκτηση συγκεκριμένων πληροφοριών σχετικά με την εικόνα προφίλ ενός χρήστη, επιτρέποντας τη διαχείριση των σχετικών πληροφοριών της με ένα δομημένο και αποτελεσματικό τρόπο. Επιπλέον είναι μέρος ενός άλλου DTO και δεν επιστρέφεται ποτέ μόνο του.

2.4.13 UploadUserBioPdfDto

Η κλάση “UploadUserBioPdfDto” διευκολύνει τη μεταφόρτωση του βιογραφικού ενός χρήστη. Αυτό το DTO είναι σχεδιασμένο να δέχεται το base64 περιεχόμενο του αρχείου PDF, προσφέροντας μια ευέλικτη λύση για τη μεταφορά και αποθήκευση του βιογραφικού σε κάποιον διακομιστή.

Ιδιότητες:

UserBioPdf: Το base64 περιεχόμενο του αρχείου PDF που αντιπροσωπεύει το βιογραφικό του χρήστη.

```
public class UploadUserBioPdfDto
{
    public byte[] UserBioPdf { get; set; }
}
```

Κώδικας 2.13: Το αντικείμενο DTO που αποθηκεύει το βιογραφικό του χρήστη σε base64 μορφή.

Χρησιμοποιώντας αυτό το DTO, οι χρήστες μπορούν εύκολα να μεταφορτώνουν το βιογραφικό τους στο σύστημα, παρέχοντας το base64 του PDF και βελτιστοποιώντας έτσι τη διαδικασία αποθήκευσης και διαχείρισης των βιογραφικών τους στο πλαίσιο του συστήματος.

2.4.14 CreateAccountRequestDto

Η κλάση αυτή χρησιμοποιείται από τους απλούς χρήστες όταν θέλουν να υποβάλουν μία αξίωση για ένα προκατασκευασμένο λογαριασμό, διεκδικώντας έτσι τον συγκεκριμένο καλλιτέχνη από την βάση σαν δικό τους.

Ιδιότητες:

PersonId: Ο μοναδικός αριθμός του καλλιτέχνη στην βάση.

IdentificationDocument: Κάποια είδους ταυτοποίηση, την οποία ο χρήστης υποβάλλει. Η μεταβλητή κρατάει το base64 string περιεχόμενο του PDF αρχείου.

```
public class CreateAccountRequestDto
{
    public int PersonId { get; set; }
    public byte[] IdentificationDocument { get; set; } //byte array of pdf.
}
```

Κώδικας 2.14: Το αντικείμενο DTO που χρησιμοποιείται κατά το claim ενός καλλιτέχνη.

2.5 ApiResponse

Η κλάση `ApiResponse` χρησιμοποιείται για να διαχειρίζεται την απόκριση σε αιτήσεις που γίνονται στο API. Κάθε αίτηση χρήστη επιστρέφεται με το γενικευμένο πρότυπο του ApiResponse και βρίσκεται στον φάκελο `ResponseWrapperFolder`. Αναλύονται τα χαρακτηριστικά του:

Success Property: Αναπαριστά την επιτυχία της αίτησης. Εάν η τιμή είναι true, η αίτηση θεωρείται επιτυχημένη.

Message Property: Παρέχει ένα μήνυμα που περιγράφει το αποτέλεσμα της αίτησης.

ErrorCode Property: Καθορίζει τυχόν κωδικό σφάλματος που σχετίζεται με την αποτυχία της αίτησης. Εάν η αίτηση είναι επιτυχής, η τιμή αυτή είναι null.

Κατασκευαστές:

Ο πρώτος κατασκευαστής δέχεται έναν κωδικό σφάλματος (ErrorCode) και ένα μήνυμα, προκειμένου να δημιουργήσει μια αποτυχημένη απόκριση.

Ο δεύτερος κατασκευαστής χρησιμοποιείται για να δημιουργήσει μια επιτυχημένη απόκριση, με προεπιλεγμένο μήνυμα "Successful".

ToString Method: Επιστρέφει μια συμβολοσειρά JSON που περιέχει τα στοιχεία της απόκρισης. (Για μελλοντική χρήση αν χρειαστεί).

Η υποκλάση **ApiResponse<T>** παρέχει επιπλέον λειτουργικότητα που μπορεί συμπεριλάβει κάθε τύπο αντικειμένου στο αποτέλεσμα.

2.6 Επίλογος

Στο πλαίσιο αυτού του κεφαλαίου, εξετάσαμε εκτενώς τον ρόλο και τη σημασία των Data Transfer Objects (DTOs) στο πλαίσιο της εφαρμογής μας. Κατανοήσαμε τον τρόπο με τον οποίο τα DTOs συμβάλλουν στην αποτελεσματική, ασφαλή, και συνεκτική μεταφορά δεδομένων ανάμεσα στα διάφορα στρώματα της εφαρμογής μας.

Σχολιάσαμε τον ρόλο των DTOs στη διατήρηση συνοχής στα δεδομένα και πώς συμβάλλουν στην αποφυγή υπερβολικής μεταφοράς πληροφορίας. Εξετάσαμε επίσης πώς τα DTOs συμβάλλουν στην ασφάλεια των δεδομένων στη βάση δεδομένων, προστατεύοντας από πιθανούς κινδύνους όπως οι επιθέσεις με βάση τον τύπο.

Στη συνέχεια, παρουσιάσαμε συγκεκριμένα παραδείγματα DTOs που χρησιμοποιούνται στην εφαρμογή μας. Κάθε ένα από αυτά τα DTOs σχεδιάστηκε με προσοχή, λαμβάνοντας υπόψη τις αρχές σχεδιασμού και τις ανάγκες της εφαρμογής μας.

Ο σχεδιασμός των DTOs επικεντρώνεται σε αρχές όπως η απλότητα, η αποδοτικότητα, και η ασφάλεια, ενώ οργανώνονται εντός φακέλων για ευκολότερη διαχείριση. Αυτή η δομή σχεδίασης, μαζί με τις προαναφερθείσες αρχές, συνεισφέρει στη δημιουργία μιας αξιόπιστης και ευέλικτης αρχιτεκτονικής δεδομένων για την εφαρμογή μας.

Κεφάλαιο 3ο: Data Layer

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο, θα εξετάσουμε πιο αναλυτικά το Theatrical.Data library project της εφαρμογής μας, το οποίο αποτελεί τον πυρήνα των δεδομένων μας. Το Theatrical.Data περιλαμβάνει τα μοντέλα δεδομένων, τα enums και το DbContext που είναι κρίσιμα για τη λειτουργία και τη συνοχή της εφαρμογής.

3.2 Σκοπός του Theatrical.Data

Το Theatrical.Data σχεδιάστηκε με στόχο την αποτελεσματική αναπαράσταση των δεδομένων μας. Δίνεται έμφαση στον καθορισμό των μοντέλων, των σχέσεων μεταξύ τους, και τη διαμόρφωση του DbContext. Θα αναλυθούν μερικά από τα enums, όλα τα μοντέλα της εφαρμογής, καθώς και ένα τμήμα των ρυθμίσεων του fluent API.

3.3 Enums

Τα enums προσφέρουν έναν αποτελεσματικό τρόπο να οργανώσουμε και να αναπαραστήσουμε σταθερές τιμές με σαφήνεια. Τα enums του συγκεκριμένου project δεν περιορίζονται μόνο στο Theatrical.Data αλλά χρησιμοποιούνται ευρέως σε όλα τα υπόλοιπα projects της εφαρμογής. Με τη χρήση της δήλωσης `using Theatrical.Data.enums`, μπορούν να ενσωματωθούν εύκολα σε άλλα μέρη του κώδικα, προσφέροντας αυξημένη οργάνωση και συνεκτικότητα. Τα enums είναι οργανωμένα στο φάκελο: Theatrical.Data/enums.

3.3.1 ErrorCode

Το ErrorCode enum αποτελεί ένα σημαντικό στοιχείο της εφαρμογής, παρέχοντας έναν κατάλογο κωδικών λάθους που μπορεί να επιστραφεί από το σύστημα. Ο κατάλογος αυτός περιλαμβάνει τα ακόλουθα στοιχεία:

- NotFound: Το αντικείμενο δεν βρέθηκε.
- InvalidToken: Μη έγκυρο το token.
- AlreadyExists: Το αντικείμενο υπάρχει ήδη.
- Unauthorized: Ανεπιτυχής προσπάθεια εξουσιοδότησης.
- Forbidden: Η πρόσβαση στον πόρο απαγορεύεται.
- BadRequest: Λανθασμένο αίτημα από τον πελάτη.
- ServerError: Εσωτερικό σφάλμα του διακομιστή.
- InvalidEmail: Μη έγκυρη διεύθυνση email.
- AlreadyVerified: Το email έχει ήδη επαληθευτεί.
- _2FaEnabled: Η διπαραμετρική αυθεντικοποίηση είναι ενεργοποιημένη.
- _2FaFailure: Αποτυχία διπαραμετρικής αυθεντικοποίησης.
- _2FaDisabled: Η διπαραμετρική αυθεντικοποίηση είναι απενεργοποιημένη.
- AlreadyLoggedIn: Ο χρήστης έχει ήδη συνδεθεί.
- InvalidCharacters: Μη έγκυροι χαρακτήρες στα δεδομένα.
- UserAlreadyClaimedVenue: Ο χρήστης έχει ήδη καταχωρήσει τον χώρο.
- UserAlreadyClaimedEvent: Ο χρήστης έχει ήδη καταχωρήσει το event.
- TwilioError: Σφάλμα στην αποστολή μηνύματος μέσω Twilio.
- InsufficientBalance: Ανεπαρκές υπόλοιπο στον λογαριασμό.
- WrongDateFormat: Λανθασμένη μορφή ημερομηνίας.

```
public enum ErrorCode
{
    NotFound,
    InvalidToken,
    AlreadyExists,
    Unauthorized,
    Forbidden,
    BadRequest,
    ServerError,
    InvalidEmail,
    AlreadyVerified,
    _2FaEnabled,
    _2FaFailure,
    _2FaDisabled,
    AlreadyLoggedIn,
    InvalidCharacters,
    UserAlreadyClaimedVenue,
    UserAlreadyClaimedEvent,
    TwilioError,
    InsufficientBalance,
    WrongDateFormat
}
```

Κώδικας 3.1: Το enum του ErrorCode.

Ο κωδικός αυτού του enum επιστρέφεται σε ανεπιτυχή αιτήματα στο πεδίο του `ErrorCode` του `ApiResponse`, που θα αναλυθεί αργότερα.

3.3.2 SocialMedia

Το enum SocialMedia χρησιμοποιείται σε διάφορες μεθόδους, όπως οι .ValidateSocialMediaForDelete, .RemoveSocialMedia, κλπ., προκειμένου να προσδιορίσει το endpoint που προκάλεσε την εκάστοτε κλήση. Αυτή η χρήση επιτρέπει στο σύστημα να αναγνωρίζει δυναμικά τον τύπο του επιλεγμένου κοινωνικού μέσου και να προσαρμόζεται ανάλογα, εκτελώντας τις κατάλληλες ενέργειες για κάθε περίπτωση. Αυτή η προσέγγιση επιτρέπει στην εφαρμογή να διαχειρίζεται δυναμικά και αποτελεσματικά τις ενέργειες που σχετίζονται με τη διαγραφή ή την προσθήκη κάθε κοινωνικού μέσου.

```
public enum SocialMedia
{
    Facebook,
    Youtube,
    Instagram
}
```

Κώδικας 3.2: Το enum του SocialMedia.

3.4 Μοντέλα

Στο κεφάλαιο αυτό θα εξεταστούν τα μοντέλα δεδομένων που χρησιμοποιούνται στην εφαρμογή. Τα μοντέλα αυτά αντιπροσωπεύουν τα διάφορα δεδομένα που αποθηκεύονται στη βάση δεδομένων της εφαρμογής και αποτελούν ένα κρίσιμο κομμάτι του συστήματος.

3.4.1 User

Το μοντέλο User περιγράφει έναν χρήστη στο σύστημα. Οι ιδιότητες που περιλαμβάνονται:

Id: Ένα μοναδικό αναγνωριστικό για κάθε χρήστη.

Username: Το όνομα χρήστη του χρήστη, προαιρετικό και μοναδικό εφόσον υπάρχει το email.

Email: Η διεύθυνση email του χρήστη, μοναδικό πεδίο.

Password: Ο κωδικός του χρήστη, κατακερματισμένος (hashed).

Enabled: Ένδειξη εάν ο χρήστης έχει επαληθεύσει το email του.

VerificationCode: Ο κωδικός επαλήθευσης για το email, που δημιουργείται μετά την εγγραφή.

_2FA_enabled: Κατάσταση ενεργοποίησης διπαραμετρικής αυθεντικότητας.

_2FA_code: Ο κωδικός διπαραμετρικής αυθεντικοποίησης, που δημιουργείται κάθε φορά που ένας χρήστης, με ενεργοποιημένη την λειτουργεία 2FA, προσπαθεί να συνδεθεί.

UserSecret: Το μυστικό του χρήστη για τη διπαραμετρική αυθεντικοποίηση που χρησιμοποιείται για την δημιουργία και την επαλήθευση του 2FA code, του χρήστη. Δημιουργείται όταν ένας χρήστης ενεργοποιεί την λειτουργία 2FA για τον λογαριασμό του.

PerformerRoles: Οι ρόλοι καλλιτέχνη του χρήστη, προαιρετικοί.

Facebook, Youtube, Instagram: Οι σύνδεσμοι προς τα προφίλ του χρήστη σε κοινωνικά δίκτυα.

BioPdfLocation: Η τοποθεσία αποθήκευσης του αρχείου βιογραφίας του χρήστη.

PhoneNumber: Ο αριθμός τηλεφώνου του χρήστη.

PhoneNumberVerified: Εάν ο αριθμός τηλεφώνου έχει επαληθευτεί.

```
public class User
{
    public int Id { get; set; }
    public string? Username { get; set; }
    public string Email { get; set; } = null!;
    public string? Password { get; set; }
    public bool? Enabled { get; set; }
    public string? VerificationCode { get; set; }
    public bool _2FA_enabled { get; set; }
    public string? _2FA_code { get; set; }
    public string? UserSecret { get; set; }
    public List<string>? PerformerRoles { get; set; }
    public string? Facebook { get; set; }
    public string? Youtube { get; set; }
    public string? Instagram { get; set; }
    public string? BioPdfLocation { get; set; }
    public string? PhoneNumber { get; set; }
    public bool? PhoneNumberVerified { get; set; }

    //Navigational Properties
    public virtual List<UserImage>? UserImages { get; set; }
    public virtual List<UserAuthority> UserAuthorities { get; set; }
    public virtual List<Transaction> UserTransactions { get; set; }
    public virtual List<UserVenue> UserVenues { get; set; }
    public virtual List<UserEvent> UserEvents { get; set; }
}
```

Κώδικας 3.3: Η κλάση User.cs

3.4.2 Authority

Η κλάση Authority αντιπροσωπεύει μια αρμοδιότητα (Authority) στο σύστημά μας. Το μοντέλο αυτό περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό της αρμοδιότητας.

Name: Το όνομα της αρμοδιότητας. (admin, user, developer, claims manager)

Οι αρμοδιότητες χρησιμοποιούνται για τη διαχείριση των δικαιωμάτων των χρηστών στην εφαρμογή μας. Κάθε αρμοδιότητα αντιστοιχεί σε ένα συγκεκριμένο επίπεδο πρόσβασης ή δικαιώματος, επιτρέποντας τον προσδιορισμό των ενεργειών που μπορεί να εκτελέσει ένας χρήστης στην εφαρμογή.

3.4.3 UserAuthority

Η κλάση UserAuthority αντιπροσωπεύει τον σύνδεσμο μεταξύ ενός χρήστη και μιας αρμοδιότητας (ρόλου), Αυτό το μοντέλο διαθέτει τα εξής στοιχεία:

UserId: Το αναγνωριστικό του χρήστη.

AuthorityId: Το αναγνωριστικό της αρμοδιότητας ρόλου.

Επιπλέον, περιλαμβάνει τα εξής πεδία πλοήγησης, για το Entity Framework:

User: Μια πλοήγηση προς το σχετικό αντικείμενο χρήστη.

Authotiry: Μια πλοήγηση προς το αντικείμενο αρμοδιότητας.

3.4.4 Person

Η κλάση Person αντιπροσωπεύει έναν καλλιτέχνη στο σύστημα. Τα χαρακτηριστικά του μοντέλου περιλαμβάνουν:

Id: Το μοναδικό αναγνωριστικό του καλλιτέχνη.

Fullname: Το πλήρες όνομα του καλλιτέχνη.

SystemId: Το αναγνωριστικό του συστήματος το οποίο έβαλε την συγκεκριμένη εγγραφή.

Timestamp: Η χρονική στιγμή καταχώρησης του καλλιτέχνη.

HairColor: Το χρώμα των μαλλιών του καλλιτέχνη.

Height: Το ύψος του καλλιτέχνη.

EyeColor: Το χρώμα των ματιών του καλλιτέχνη.

Weight: Το βάρος του καλλιτέχνη.

Languages: Λίστα με τις γλώσσες που μιλάει ο καλλιτέχνης.

Description: Περιγραφή του καλλιτέχνη.

Bio: Βιογραφία του καλλιτέχνη.

Birthdate: Η ημερομηνία γέννησης του καλλιτέχνη.

Roles: Λίστα με τους ρόλους που διαδραματίζει τον καλλιτέχνη.

IsClaimed: Ένδειξη κατάστασης ισχύος (claimed) του καλλιτέχνη.

ClaimingStatus: Η κατάσταση της αξίωσης (claiming) του καλλιτέχνη.

3.4.5 System

Η κλάση System αναπαριστά ένα σύστημα. Το διαθέσιμα συστήματα που υπάρχουν είναι τα εξής:

- Python: 2
- C#: 3
- C: 4
- C++: 5
- Java: 10
- Android: 11
- iOS: 12
- Javascript: 13
- Testing Application: 14

Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό του συστήματος.

Name: Το όνομα του συστήματος.

3.4.6 AccountRequest

Το μοντέλο αυτό αποτελεί μια αναπαράσταση αιτήματος για claim ενός καλλιτέχνη (Person) από έναν χρήστη (User). Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό του αιτήματος.

UserId: Το αναγνωριστικό του χρήστη που υποβάλλει το αίτημα.

PersonId: Το αναγνωριστικό του προσώπου που αντιστοιχεί στο αίτημα.

IdentificationDocument: Ένα πεδίο που αποθηκεύει το αναγνωριστικό έγγραφο του αιτούντος.

ConfirmationStatus: Ο κατάσταση επιβεβαίωσης του αιτήματος.

CreatedAt: Η ημερομηνία και ώρα δημιουργίας του αιτήματος.

AuthorizedBy: Πεδίο που αναγράφει ποιος διαχειριστής εξουσιοδότησε το αίτημα.

Το AccountRequest συνδέεται με τα παρακάτω μοντέλα:

User: Ο χρήστης που υποβάλλει το αίτημα.

Person: Το πρόσωπο που σχετίζεται με το αίτημα.

Αυτό το μοντέλο παρέχει έναν τρόπο για τη δημιουργία λογαριασμών στο σύστημα, ενώ παράλληλα διατηρεί τα στοιχεία των χρηστών και των προσώπων που υποβάλλουν τα αιτήματα.

```
public class AccountRequest
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public int PersonId { get; set; }
    public string IdentificationDocument { get; set; }
    public ConfirmationStatus ConfirmationStatus { get; set; }
    public DateTime CreatedAt { get; set; }
```

```
    public string? AuthorizedBy { get; set; }

    //Navigational Properties
    public virtual User User { get; set; }
    public virtual Person Person { get; set; }
}
```

Κώδικας 3.4: Η κλάση AccountRequest.cs

3.4.7 AssignedUser

Το μοντέλο αυτό αντιπροσωπεύει τη σχέση μεταξύ χρήστη, προσώπου και ανατεθειμένου αιτήματος. Περιλαμβάνει τα παρακάτω:

Id: Το μοναδικό αναγνωριστικό της εγγραφής.

UserId: Το αναγνωριστικό του χρήστη που ανατίθεται.

PersonId: Το αναγνωριστικό του προσώπου που ανατίθεται.

RequestId: Το αναγνωριστικό του αιτήματος που ανατίθεται.

CreatedAt: Η ημερομηνία και ώρα δημιουργίας της εγγραφής.

Οι Σχέσεις και οι Πλοήγησης με άλλα αντικείμενα:

User: Ο χρήστης που ανατίθεται.

Person: Το πρόσωπο που ανατίθεται.

AccountRequest: Το αίτημα που ανατίθεται.

```
public class AssignedUser
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public int PersonId { get; set; }
    public int RequestId { get; set; }
    public DateTime CreatedAt { get; set; }

    //Navigational Properties
    public virtual User User { get; set; }
    public virtual Person Person { get; set; }
    public virtual AccountRequest AccountRequest { get; set; }
}
```

Κώδικας 3.5: Η κλάση AssignedUser.cs

3.4.8 ChangeLog

Το μοντέλο ChangeLog καταγράφει τις αλλαγές στη βάση δεδομένων και παρέχει πληροφορίες σχετικά με τον τύπο της εκδήλωσης, τον πίνακα, την τιμή, τη στήλη και τη χρονική στιγμή της αλλαγής. Τα χαρακτηριστικά του μοντέλου αυτού:

Id: Το μοναδικό αναγνωριστικό της εγγραφής.

EventType: Ο τύπος της εκδήλωσης (π.χ., προσθήκη, ενημέρωση, διαγραφή).

TableName: Ο πίνακας στον οποίο πραγματοποιήθηκε η αλλαγή.

Value: Η νέα τιμή μετά την αλλαγή.

CollumnName: Το όνομα της στήλης που επηρεάστηκε.

Timestamp: Η χρονική στιγμή της αλλαγής.

Το ChangeLog διευκολύνει την παρακολούθηση και τον έλεγχο των αλλαγών στη βάση δεδομένων, καταγράφοντας σημαντικές πληροφορίες που συνδέονται με τις εκδηλώσεις που λαμβάνουν χώρα.

3.4.9 Contribution

Το μοντέλο αυτό καταγράφει τις συνεισφορές σε παραγωγές, παρέχοντας πληροφορίες σχετικά με το πρόσωπο, την παραγωγή, το ρόλο, τον υπορόλο, το σύστημα και τη χρονική στιγμή της εγγραφής. Τα χαρακτηριστικά του μοντέλου:

Id: Το μοναδικό αναγνωριστικό της εγγραφής.

PersonId: Το αναγνωριστικό του προσώπου που συνεισέφερε.

ProductionId: Το αναγνωριστικό της παραγωγής στην οποία έγινε η συνεισφορά.

RoleId: Το αναγνωριστικό του ρόλου.

SubRole: Ο υπορόλος, εάν υπάρχει.

SystemId: Το αναγνωριστικό του συστήματος εγγραφής.

Timestamp: Η χρονική στιγμή της συνεισφοράς.

3.4.10 Event

Το μοντέλο αυτό αναπαριστά ένα συγκεκριμένο γεγονός στην εφαρμογή. Καταγράφει πληροφορίες σχετικά με την παραγωγή στην οποία ανήκει το event, τον χώρο (venue) όπου θα πραγματοποιηθεί/πραγματοποιήθηκε, την ημερομηνία διεξαγωγής, την κλίμακα των τιμών των εισιτηρίων, το σύστημα που πραγματοποίησε την εγγραφή, και άλλες λεπτομέρειες. Πιο συγκεκριμένα, τα χαρακτηριστικά που περιλαμβάνονται:

Id: Το μοναδικό αναγνωριστικό του event.

ProductionId: Το αναγνωριστικό της παραγωγής που σχετίζεται με το event.

VenueId: Το αναγνωριστικό του χώρου (venue) όπου θα πραγματοποιηθεί το event.

DateEvent: Η ημερομηνία διεξαγωγής του event.

PriceRange: Η κλίμακα των τιμών του event.

SystemId: Το αναγνωριστικό του συστήματος που σχετίζεται με το event.

Timestamp: Η χρονική στιγμή καταχώρησης του event.

IsClaimed: Ένδειξη εάν το event έχει ήδη δηλωθεί από χρήστη.

Επιπλέον σχέσεις πλοήγησης με άλλα αντικείμενα.

Production: Η σχετική παραγωγή που συνδέεται με το event.

System: Το σύστημα που σχετίζεται με το event.

Venue: Ο χώρος όπου θα πραγματοποιηθεί το event.

UserEvents: Λίστα με συνδέσεις με χρήστες που έχουν δηλώσει το event σαν δικό τους.

3.4.11 Image

Το μοντέλο αυτό αναπαριστά μια εικόνα στην εφαρμογή. Έχει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό της εικόνας.

ImageUrl: Το URL της εικόνας.

PersonId: Το αναγνωριστικό του προσώπου που σχετίζεται με την εικόνα.

3.4.12 Organizer

Το μοντέλο Organizer αναπαριστά έναν διοργανωτή στο σύστημά μας. Κάθε αντικείμενο αυτού του τύπου καταγράφει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό του διοργανωτή.

Name: Το όνομα του διοργανωτή.

Address: Η διεύθυνση του διοργανωτή.

Town: Η πόλη του διοργανωτή.

Postcode: Ο ταχυδρομικός κώδικας του διοργανωτή.

Phone: Ο αριθμός τηλεφώνου του διοργανωτή.

Email: Η διεύθυνση email του διοργανωτή.

Doy: Ο κωδικός ΔΟΥ του διοργανωτή.

Afm: Ο ΑΦΜ (Αριθμός Φορολογικού Μητρώου) του διοργανωτή.

SystemId: Το αναγνωριστικό του συστήματος εγγραφής.

Timestamp: Η χρονική στιγμή δημιουργίας του αντικειμένου.

Σχέσεις:

System: Το σύστημα στο οποίο ανήκει ο διοργανωτής.

Productions: Μια λίστα από παραγωγές που συνδέονται με τον διοργανωτή.

3.4.13 Production

Η κλάση αυτή αναπαριστά μια παραγωγή στο σύστημα. Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό της παραγωγής.

OrganizerId: Το αναγνωριστικό του διοργανωτή που συσχετίζεται με την παραγωγή.

Title: Ο τίτλος της παραγωγής.

Description: Η περιγραφή της παραγωγής.

Url: Η διεύθυνση URL της παραγωγής.

Producer: Ο παραγωγός της παραγωγής.

MediaUrl: Η διεύθυνση URL του πολυμέσου που σχετίζεται με την παραγωγή.

Duration: Η διάρκεια της παραγωγής (προαιρετικό).

SystemId: Το αναγνωριστικό του συστήματος εγγραφής.

Timestamp: Η χρονική στιγμή δημιουργίας της παραγωγής.

Σχέσεις:

Organizer: Ο διοργανωτής που συσχετίζεται με την παραγωγή.

System: Το σύστημα που έγινε η εγγραφή.

Contributions: Μια λίστα από συνεισφορές που συσχετίζονται με την παραγωγή.

Events: Μια λίστα από γεγονότα που συσχετίζονται με την παραγωγή.

3.4.14 Role

Η κλάση αυτή αναπαριστά έναν ρόλο στο σύστημά μας. Περιλαμβάνει τα εξής:

Id: Το μοναδικό αναγνωριστικό του ρόλου.

Role1: Ο τίτλος του ρόλου που μπορεί να έχει ένας καλλιτέχνης.

SystemId: Το αναγνωριστικό του συστήματος εγγραφής.

Timestamp: Η χρονική στιγμή δημιουργίας του ρόλου.

Σχέσεις:

System: Το σύστημα που έγινε η εγγραφή.

Contributions: Μια λίστα από συνεισφορές που συσχετίζονται με τον ρόλο.

3.4.15 UserEvent

Η κλάση αυτή χρησιμοποιείται για τον συσχετισμό των χρηστών με τα events στο σύστημά μας. Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό.

UserId: Το αναγνωριστικό του χρήστη που έχει κάνει claim σαν δικό του.

EventId: Το αναγνωριστικό του event.

DateCreated: Η χρονική στιγμή δημιουργίας της σχέσης.

Σχέσεις:

Event: Το event που σχετίζεται με τη σχέση.

User: Ο χρήστης που σχετίζεται με τη σχέση.

3.4.16 Venue

Η κλάση αυτή αντιπροσωπεύει τους χώρους στους οποίους μπορούν να λάβουν χώρα τα events. Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό του χώρου.

Title: Ο τίτλος του χώρου.

Address: Η διεύθυνση του χώρου.

SystemId: Το αναγνωριστικό του συστήματος εγγραφής.

Timestamp: Η χρονική στιγμή δημιουργίας του χώρου.

isClaimed: Ένδειξη εάν ο χώρος έχει γίνει claim από κάποιον χρήστη.

Σχέσεις:

System: Το σύστημα που έγινε η εγγραφή.

Events: Η λίστα των events που λαμβάνουν χώρα στον χώρο.

UserVenues: Η λίστα των συσχετίσεων χρηστών με τον συγκεκριμένο χώρο.

```
public class Venue
{
    public int Id { get; set; }
    public string? Title { get; set; }
    public string? Address { get; set; }
    public int SystemId { get; set; }
    public DateTime Timestamp { get; set; }
    public bool isClaimed { get; set; }

    //Navigational Properties
    public virtual System System { get; set; } = null!;
    public virtual List<Event> Events { get; set; }
    public virtual List<UserVenue> UserVenues { get; set; }
}
```

Κώδικας 3.6: Η κλάση Venue.cs

3.4.17 UserVenue

Η κλάση αυτή καταγράφει τις συσχετίσεις μεταξύ χρηστών και χώρων που έχουν δημιουργηθεί μετά από την διεκδίκηση ενός χώρου από έναν χρήστη. Περιλαμβάνει τα εξής:

Id: Το μοναδικό αναγνωριστικό της συσχέτισης.

UserId: Το αναγνωριστικό του χρήστη που έχει δημιουργήσει τη συσχέτιση.

VenueId: Το αναγνωριστικό του χώρου.

DateCreated: Η χρονική στιγμή δημιουργίας της συσχέτισης.

Σχέσεις:

User: Ο χρήστης που δημιούργησε τη συσχέτιση.

Venue: Ο χώρος που έχει δημιουργηθεί και έχει διεκδικηθεί από τον χρήστη.

```
public class UserVenue
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public int VenueId { get; set; }
    public DateTime DateCreated { get; set; }

    // Navigation properties
    public virtual User User { get; set; }
    public virtual Venue Venue { get; set; }
}
```

Κώδικας 3.7: Η κλάση UserVenue.cs

3.4.18 UserImage

Η κλάση αυτή καταγράφει τις εικόνες που ανήκουν σε έναν χρήστη. Κάθε εικόνα συσχετίζεται με έναν συγκεκριμένο χρήστη. Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό της εικόνας.

UserId: Το αναγνωριστικό του χρήστη που ανήκει η εικόνα.

ImageLocation: Η τοποθεσία του αρχείου της εικόνας.

Label: Μία ετικέτα για την εικόνα.

IsProfile: Ένδειξη εάν η εικόνα αντιπροσωπεύει τη φωτογραφία προφίλ του χρήστη.

Σχέσεις:

User: Ο χρήστης στον οποίον ανήκει η εικόνα.

```
public class UserImage
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public string ImageLocation { get; set; }
    public string? Label { get; set; }
    public bool? IsProfile { get; set; }

    //Navigational Properties
    public virtual User User { get; set; }
}
```

Κώδικας 3.8: Η κλάση UserImage.cs

3.4.19 Transaction

Η κλάση αυτή καταγράφει τις χρηματικές συναλλαγές που σχετίζονται με έναν χρήστη. Περιλαμβάνει τα εξής χαρακτηριστικά:

Id: Το μοναδικό αναγνωριστικό της συναλλαγής.

UserId: Το αναγνωριστικό του χρήστη που πραγματοποίησε την συναλλαγή.

CreditAmount: Το ποσό που πιστώθηκε στον λογαριασμό του χρήστη.

AmountPaid: Το ποσό που καταβλήθηκε.

DiscountAmount: Ενδεχόμενο ποσό έκπτωσης.

Reason: Ο λόγος ή η περιγραφή της συναλλαγής.

SessionId: Το αναγνωριστικό συνεδρίας του Stripe σχετικό με τη συναλλαγή.

StripeEventId: Το αναγνωριστικό συμβάντος Stripe σχετικό με τη συναλλαγή.

DateCreated: Η ημερομηνία δημιουργίας της συναλλαγής.

Σχέσεις:

User: Ο χρήστης που σχετίζεται με τη συναλλαγή.

Η κλάση αυτή συνεισφέρει στην αποθήκευση πληροφοριών σχετικά με τις χρηματικές κινήσεις των χρηστών της εφαρμογής.

```
public class Transaction
{
    public int Id { get; set; }
    public int UserId { get; set; }
    public decimal CreditAmount { get; set; }
    public decimal AmountPaid { get; set; }
    public decimal? DiscountAmount { get; set; }
    public string Reason { get; set; }
    public string? SessionId { get; set; }
    public string? StripeEventId { get; set; }
    public DateTime DateCreated { get; set; }

    //Navigational Properties
    public virtual User User { get; set; }
}
```

Κώδικας 3.9: Η κλάση Transaction.cs

3.5 Entity Framework

Η εφαρμογή χρησιμοποιεί ένα πλαίσιο ORM που παρέχει μια ευέλικτη και εύχρηστη διεπαφή για την αλληλεπίδραση με τις σχετικές βάσεις δεδομένων εντός της πλατφόρμας (.NET) της εφαρμογής. Το Entity Framework (EF) είναι ένα πλαίσιο χαρτογράφησης αντικειμένου-σχεσιακής απεικόνισης (ORM) για την πλατφόρμα .NET. Η κύρια λειτουργία του είναι να παρέχει μια ευκολότερη και πιο αφηρημένη από τη βάση δεδομένων προσέγγιση, επιτρέποντας στους προγραμματιστές να εργάζονται με αντικείμενα στη γλώσσα προγραμματισμού τους αντί να ασχολούνται απευθείας με κώδικα SQL. Οι λόγοι για τους οποίους χρησιμοποιείται το Entity Framework (EF) είναι οι παρακάτω:

Αφαιρετικότητα από τη Βάση Δεδομένων: Ο προγραμματιστής μπορεί να σχεδιάσει το μοντέλο δεδομένων της εφαρμογής του χρησιμοποιώντας κλάσεις και συσχετίσεις, ενώ το EF αναλαμβάνει τον παραμετροποιημένο κώδικα για τη διαχείριση των αλληλεπιδράσεων με τη βάση δεδομένων.

Προγραμματιστική Ευκολία: Ο προγραμματιστής μπορεί να χρησιμοποιήσει LINQ (Language Integrated Query) για να εκτελέσει ερωτήματα στη βάση δεδομένων, προσφέροντας έναν αφηρημένο και φυσικό τρόπο ανάκτησης δεδομένων.

Αυτόματη Διαχείριση Σχήματος: Το EF μπορεί να δημιουργήσει ή να ενημερώσει τη βάση δεδομένων ανάλογα με τις αλλαγές στο μοντέλο των δεδομένων.

Συμβατότητα με Πολλαπλές Βάσεις Δεδομένων: Το Entity Framework παρέχει στον προγραμματιστή τη δυνατότητα να αντιμετωπίζει πολλαπλές βάσεις δεδομένων με άνεση. Μπορεί να διαχειριστεί διαφορετικούς παρόχους βάσεων δεδομένων με τη χρήση διαφορετικών παραμέτρων σύνδεσης κατά την αρχικοποίηση του DbContext. Αυτό επιτρέπει στην εφαρμογή να λειτουργεί σε περιβάλλοντα που χρησιμοποιούν πολλαπλές βάσεις δεδομένων ή περιοχές διάφορων πελατών χωρίς προβλήματα συμβατότητας.

Υποστήριξη Σχέσεων: Το EF προσφέρει ισχυρή υποστήριξη για τις σχέσεις μεταξύ των πινάκων. Ο προγραμματιστής μπορεί να καθορίσει σχέσεις μεταξύ των οντοτήτων με χρήση των μεθόδων Fluent API, προσθέτοντας περιορισμούς και ορίζοντας τους τύπους των συσχετίσεων, όπως 1-προς-1, πολλά-προς-1, και πολλά-προς-πολλά.

Portability: Το EF προσφέρει μεγάλη φορητότητα, καθώς ο κώδικας μοντελοποίησης δεδομένων μπορεί να μεταφερθεί μεταξύ διαφόρων παρόχων βάσεων δεδομένων. Αυτό σημαίνει ότι, αν χρειαστεί, ο προγραμματιστής μπορεί να αλλάξει τον πάροχο βάσης δεδομένων χωρίς να χρειαστεί να αλλάξει πολύ τον κώδικα της εφαρμογής.

Συνοψίζοντας, η χρήση του Entity Framework προσφέρει ευελιξία και αφαιρετικότητα από τη βάση δεδομένων, ενώ παράλληλα παρέχει ισχυρή υποστήριξη για τη διαχείριση των σχέσεων μεταξύ των οντοτήτων και την φορητότητα του κώδικα μοντελοποίησης δεδομένων.

Στη συνέχεια ακολουθεί ένα μικρό κομμάτι μοντελοποίησης του αντικειμένου “User” στο Entity Framework.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Theatrical.Data.Models;

namespace Theatrical.Data.Context;

public class TheatricalPlaysDbContext : DbContext
{
    public virtual DbSet<User> Users { get; set; } = null!;
    // Επιπλέον κώδικας…

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {

        modelBuilder.Entity<User>(entity =>
        {
            entity.ToTable("users");

            entity.HasKey(e => e.Id);

            entity.HasIndex(e => e.Email, "email")
                .IsUnique();

           // Άλλα properties…


            entity.HasMany(u => u.UserImages)
                .WithOne(ui => ui.User)
                .HasForeignKey(ui => ui.UserId)
                .OnDelete(DeleteBehavior.Cascade);
        });
        // Επιπλέον κώδικας…
      }
}
```

Κώδικας 3.10: Κομμάτι κώδικα μοντελοποίησης του “User” στο Fluent API του Entity Framework.

3.6 Επίλογος

Στο πλαίσιο αυτής της ενότητας εξετάσαμε εκτενώς τα μοντέλα, τα enums, και το μέρος του Entity Framework που σχεδιάστηκαν και υλοποιήθηκαν στο πλαίσιο “Theatrical.Data”. Τα μοντέλα μας αντιπροσωπεύουν τις βασικές οντότητες της εφαρμογής μας, ενώ τα enums μας προσφέρουν έναν αποτελεσματικό τρόπο για να διαχειριστούμε περιορισμούς και επιλογές.

Ο τρόπος που χρησιμοποιήσαμε το Entity Framework επιτρέπει την αποτελεσματική αναπαράσταση των δεδομένων μας και τη διαχείριση των αλληλεπιδράσεών τους με τη βάση δεδομένων. Με τη χρήση του Fluent API, προσαρμόσαμε τον τρόπο που το EF δημιουργεί τους πίνακες και διαχειρίζεται τις σχέσεις μεταξύ τους, προσφέροντας έτσι μια εύκολη και ευέλικτη λύση.

Συνοψίζοντας, η ενότητα “Theatrical.Data” αποτελεί το θεμέλιο για την αποτελεσματική ανάπτυξη και λειτουργία της εφαρμογής μας, προσφέροντας ένα στέρεο, καλά σχεδιασμένο υπόβαθρο για τον χειρισμό των δεδομένων μας.

Στην επόμενη ενότητα, θα εξετάσουμε περαιτέρω τον κώδικα και τις λειτουργίες της εφαρμογής μας, επικεντρώνοντας την προσοχή μας στον τρόπο με τον οποίο αλληλεπιδρά με τα δεδομένα και παρέχει τις λειτουργίες που αναμένουν οι χρήστες μας. Πιο συγκεκριμένα θα εξετάσουμε το Service Layer της εφαρμογής.

Κεφάλαιο 4ο: Service Layer

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο, εξετάζουμε λεπτομερώς το Theatrical.Service, library project της εφαρμογής μας. Το Theatrical.Service αποτελεί κρίσιμο κομμάτι της επιχειρησιακής λογικής της εφαρμογής, δρώντας ως γέφυρα μεταξύ των δεδομένων της βάσης και των υπηρεσιών που παρέχονται στο πλαίσιο της εφαρμογής μας.

4.2 Σκοπός του Service Layer

Το Service Layer είναι υπεύθυνο για την ομαλή επικοινωνία με τα δεδομένα της βάσης δεδομένων και την παροχή διάφορων υπηρεσιών που απαιτούνται στο πλαίσιο της εφαρμογής μας.

4.3 Δομή του Service Layer

Το Service Layer περιλαμβάνει μια σειρά υπηρεσιών, καθεμία από τις οποίες αναλαμβάνει συγκεκριμένες επιχειρησιακές λειτουργίες. Σημαντική πτυχή της δομής αυτής αποτελεί η στενή σχέση μεταξύ των υπηρεσιών και του Data Layer (Theatrical.Data). Σχεδόν όλες οι υπηρεσίες του παρόντος έργου αλληλεπιδρούν με το Data Layer για την πρόσβαση στα μοντέλα της βάσης και το dbContext.

Κάθε υπηρεσία λειτουργεί ως διακριτό επίπεδο, προσφέροντας υπηρεσίες που αντιστοιχούν σε συγκεκριμένες λειτουργίες της εφαρμογής. Κατά την ανάπτυξη των υπηρεσιών, η προσεκτική σχεδίαση της επικοινωνίας τους με το Data Layer, και στη συνέχεια με το API layer, εξασφαλίζει τη συνέπεια των λειτουργιών, ενισχύοντας τη συνολική απόδοση και εύκολη επεκτασιμότητα και συντήρηση του έργου.

4.3.1 Λεπτομέρειες Δομής του Έργου

Το **Theatrical.Services** είναι οργανωμένο σε διάφορες υποκατηγορίες για να διευκολύνει τη συντήρηση και την ανάπτυξη. Κάθε υποκατηγορία παρέχει συγκεκριμένη λειτουργικότητα στο πλαίσιο του Service Layer.

Theatrical.Services > Caching

Στον φάκελο αυτό βρίσκεται ο κώδικας που αφορά τον μηχανισμό κασαρίσματος (caching).

Theatrical.Services > Curators

Στον φάκελο αυτό βρίσκονται οι κλάσεις που αφορούν το καθάρισμα των δεδομένων (data cleaning). Επίσης περιέχει και υποφάκελο ο οποίος θα αναλύεται στη συνέχεια.

Theatrical.Services > Email

Στον φάκελο αυτό βρίσκεται ο κώδικας που σχετίζεται με τη λειτουργικότητα του ηλεκτρονικού ταχυδρομείου. Εδώ εντοπίζεται η υπηρεσία και οι λειτουργίες που αφορούν την αποστολή email στους χρήστης της εφαρμογής.

Theatrical.Services > Pagination

Στον φάκελο αυτό βρίσκεται ο κώδικας για την υπηρεσία της σελιδοποίησης των δεδομένων.

Theatrical.Services > PersonService

Στον φάκελο αυτό βρίσκεται ο κώδικας για την υπηρεσία των προσώπων (καλλιτεχνών) της εφαρμογής. (Δεν έχει σχέση με τους χρήστες της εφαρμογής User).

Theatrical.Services > PhoneVerification > Twilio

Στον φάκελο αυτό βρίσκεται ο κώδικας για την υπηρεσία επιβεβαίωσης των κινητών τηλεφώνων των χρηστών μέσω της υπηρεσίας του Twilio.

Theatrical.Services > Repositories

Στον φάκελο αυτό βρίσκεται ο κώδικας για τις υπηρεσίες που αλληλεπιδρούν με τη βάση δεδομένων. Αυτές οι υπηρεσίες, επίσης γνωστές ως repositories, υλοποιούν λειτουργίες που περιλαμβάνουν την ανάκτηση, ενημέρωση, διαγραφή και γενική διαχείριση των δεδομένων στη βάση δεδομένων της εφαρμογής μας.

Theatrical.Services > Security > AuthorizationFilters

Στον φάκελο αυτό βρίσκεται ο κώδικας που υλοποιεί υπηρεσίες σχετικές με την ασφάλεια. Αυτές οι υπηρεσίες είναι υπεύθυνες για την εφαρμογή φίλτρων εξουσιοδότησης στο πλαίσιο του API. Ο κώδικας περιλαμβάνει τις λειτουργίες που ελέγχουν την πρόσβαση των χρηστών σε συγκεκριμένους πόρους, λαμβάνοντας υπόψη τους κανόνες εξουσιοδότησης που καθορίζονται στο σύστημα.

Theatrical.Services > Security > Jwt

Ο φάκελος `Jwt` περιέχει τον κώδικα που υλοποιεί την υπηρεσία για τη δημιουργία και έλεγχο (verify) των JWT. Αυτή η υπηρεσία αναλαμβάνει τον κρίσιμο ρόλο της δημιουργίας ασφαλών token για την αυθεντικοποίηση των χρηστών και τον έλεγχο της εγκυρότητάς τους κατά τη διάρκεια των αιτημάτων στην εφαρμογή μας.

Theatrical.Services > Validation

Στον φάκελο αυτό βρίσκεται ο κώδικας που υλοποιεί υπηρεσίες σχετικές με την επιβεβαίωση των δεδομένων πριν από την ενημέρωση και τη διαγραφή από τη βάση. Αυτές οι υπηρεσίες είναι αναγκαίες για τη διασφάλιση της συνοχής και της ακεραιότητας των δεδομένων κατά την επικοινωνία με τη βάση δεδομένων. Ο κώδικας περιλαμβάνει λειτουργίες που εξετάζουν και επιβεβαιώνουν την ορθότητα των δεδομένων προτού γίνει οποιαδήποτε ενέργεια ενημέρωσης ή διαγραφής, ενισχύοντας την αξιοπιστία του συστήματος.

Και οι υπόλοιπες κλάσεις υπηρεσιών που δεν ανήκουν σε κάποιο συγκεκριμένα φάκελο μέσα στο παρόν έργου βιβλιοθήκης ονομαστικά είναι οι εξής: AccountRequestService.cs, ContributionService.cs, EventService.cs, LogService.cs, MinioService.cs, OrganizerService.cs, ProductionService.cs, RoleService.cs, ShowService.cs, TransactionService.cs, UserEventService.cs, UserService.cs, UserVenueService.cs, VenueService.cs.

Ο κώδικας που περιέχεται σε κάθε υποκατηγορία αναλύεται λεπτομερώς στο πλαίσιο της παρούσας ενότητας. Κάθε υπηρεσία εξυπηρετεί συγκεκριμένες λειτουργικότητες, συνεισφέροντας στην ενίσχυση της συνολικής απόδοσης, της ευκολίας συντήρησης και της διασφάλισης της σωστής λειτουργίας της εφαρμογής. Αυτή η δομή παρέχει μια καλή οργάνωση και διαχείριση του κώδικα, προωθώντας τη βέλτιστη πρακτική κατά την ανάπτυξη και τη συντήρηση του συστήματος μας.

4.4 Caching

Ο Ρόλος του Caching σε Σύγχρονες Εφαρμογές

Στον σύγχρονο κόσμο των εφαρμογών, το Caching αναδεικνύεται ως ένα καίριο εργαλείο για τη βελτίωση της απόδοσης και της εμπειρίας των χρηστών. Η διατήρηση και η αποτελεσματική διαχείριση προσωρινών δεδομένων συμβάλλουν στη μείωση του χρόνου απόκρισης των εφαρμογών, καθιστώντας τις πιο αποτελεσματικές και αποδοτικές.

Memory Caching στην Εφαρμογή μας

Στο πλαίσιο της εφαρμογής μας, χρησιμοποιούμε το Memory Caching που παρέχει το πλαίσιο ανάπτυξης .NET. Αυτή η τεχνική επιτρέπει την αποθήκευση προσωρινών δεδομένων απευθείας στη μνήμη του εξυπηρετητή, επιτυγχάνοντας ταχύτατη πρόσβαση σε πληροφορίες που απαιτούνται συχνά από την εφαρμογή.

Ο φάκελος Caching περιλαμβάνει τον κώδικα που διαχειρίζεται αυτόν τον μηχανισμό, με επιμέρους υποκατηγορίες που βελτιστοποιούν τη διαχείριση της μνήμης και την ανάκτηση προσωρινών δεδομένων. Με αυτόν τον τρόπο, επιτυγχάνεται ο συντονισμός μεταξύ της ευελιξίας και της αποδοτικότητας, προσφέροντας μια βελτιστοποιημένη εμπειρία χρήστη στην εφαρμογή μας.

Ο μηχανισμός του Caching υλοποιείται στο πλαίσιο των Repositories, τα οποία αναλύονται στη συνέχεια.

4.4.1 Caching.cs

Η κλάση **Caching** αποτελεί τον πυρήνα του μηχανισμού κασαρίσματος (caching) της εφαρμογής μας. Ο μηχανισμός αυτός είναι υπεύθυνος για την αποθήκευση και ανάκτηση δεδομένων από τη μνήμη, επιτυγχάνοντας έτσι βελτιστοποίηση της απόδοσης και της αποκρισιμότητας της εφαρμογής.

Διεπαφή **ICaching**:

Η διεπαφή **ICaching** ορίζει δύο βασικές μεθόδους για την αποθήκευση και ανάκτηση δεδομένων:

- GetOrSetAsync: Χρησιμοποιείται για την ανάκτηση δεδομένων. Εάν τα δεδομένα δεν βρίσκονται στην cache, ανακτώνται από την πηγή και στη συνέχεια αποθηκεύονται στη cache για μελλοντική χρήση. Σχεδόν πάντα χρησιμοποιείται αυτή, στο πλαίσιο της εφαρμογής μας.

- GetOrSetAsyncWithSlidingWindow: Παρόμοια με την προηγούμενη, με τη διαφορά ότι προσθέτει ένα παράθυρο στην χρονική ισχύ των δεδομένων στην cache. Κατά την ανανέωση της κρυφής μνήμης, τα δεδομένα παραμένουν έγκυρα ένα καθορισμένο χρονικό διάστημα, προτού λήξει το παράθυρο. Η κλάση αυτή δημιουργήθηκε μόνο για μία περίπτωση όπου ένα request ζητάει έναν υπερβολικά μεγάλο αριθμό δεδομένων από τη βάση δεδομένων, έτσι ώστε να μειωθεί ο χρόνος απόκρισης μετά το αρχικό request, σε συνάρτηση με το παράθυρο λήξης.

Υλοποίηση Κλάσης Caching

Η κλάση Caching υλοποιεί τη διεπαφή ICaching και χρησιμοποιεί τον μηχανισμό μνήμης του .NET, τον IMemoryCache, για τη διαχείριση της κρυφής μνήμης.

Κατά την ανάκτηση δεδομένων, η κλάση ελέγχει πρώτα εάν τα δεδομένα βρίσκονται ήδη στην cache. Εάν όχι, τα δεδομένα ανακτώνται από την πηγή (μέσω της παρεχόμενης συνάρτησης retrieve). Στη συνέχεια, τα δεδομένα αποθηκεύονται στην cache, με προσαρμοσμένες επιλογές χρησιμοποιώντας το MemoryCacheEntryOptions.

Ακολουθεί ο κώδικας caching.

```
using Microsoft.Extensions.Caching.Memory;
namespace Theatrical.Services.Caching;

public interface ICaching
{
    Task<T> GetOrSetAsync<T>(string key, Func<Task<T>> retrieve);
    Task<T> GetOrSetAsyncWithSlidingWindow<T>(string key, Func<Task<T>> retrieve);
}

public class Caching : ICaching
{
    private readonly IMemoryCache _memoryCache;

    public Caching(IMemoryCache memoryCache)
    {
        _memoryCache = memoryCache;
```

```
    }

    public async Task<T> GetOrSetAsync<T>(string key, Func<Task<T>> retrieve)
    {
        if (!_memoryCache.TryGetValue(key, out T cacheValue))
        {
            cacheValue = await retrieve();

            if (cacheValue is not null)
            {
                MemoryCacheEntryOptions cacheOptions = new MemoryCacheEntryOptions()
                    .SetAbsoluteExpiration(TimeSpan.FromMinutes(1));

                _memoryCache.Set(key, cacheValue, cacheOptions);
            }
        }

        return cacheValue;
    }

    public async Task<T> GetOrSetAsyncWithSlidingWindow<T>(string key, Func<Task<T>> retrieve)
    {
        if (!_memoryCache.TryGetValue(key, out T cacheValue))
        {
            cacheValue = await retrieve();

            if (cacheValue is not null)
            {
                MemoryCacheEntryOptions cacheOptions = new MemoryCacheEntryOptions()
                    .SetAbsoluteExpiration(TimeSpan.FromMinutes(1))
                    .SetSlidingExpiration(TimeSpan.FromSeconds(10));

                _memoryCache.Set(key, cacheValue, cacheOptions);
            }
        }
        return cacheValue;
    }
}
```

Κώδικας 4.1: Ο μηχανισμός υλοποίησης του caching.

4.5 Curators Καθαρισμού Δεδομένων

Οι Curators Καθαρισμού Δεδομένων αντιπροσωπεύουν ένα σημαντικό κομμάτι της επιχειρησιακής λογικής μας που επικεντρώνεται στη διασφάλιση της ακρίβειας, της εγκυρότητας και της ομαλής λειτουργίας των δεδομένων μας. Ο ρόλος τους είναι να εφαρμόζουν κανόνες και διαδικασίες που εξασφαλίζουν την ποιοτική διαχείριση των δεδομένων, αντιμετωπίζοντας πιθανά προβλήματα και διατηρώντας τη βάση δεδομένων μας σε καλή κατάσταση.

4.5.1 Ρόλος των Curators Καθαρισμού Δεδομένων

Οι Curators Καθαρισμού Δεδομένων εκτελούν διάφορες λειτουργίες, συμπεριλαμβανομένων:

Έλεγχος Ακρίβειας: Επιβεβαίωση και διόρθωση ανακριβειών στα δεδομένα, εξασφαλίζοντας ότι οι πληροφορίες είναι ενημερωμένες και σωστές.

Επικύρωση Συμπληρωματικών Πληροφοριών: Προσθήκη ή ενημέρωση πρόσθετων πληροφοριών που ενισχύουν την πληρότητα των δεδομένων.

Διαγραφή Μη Σχετικών Δεδομένων: Αφαίρεση παλαιών, ξεπερασμένων ή μη σχετικών δεδομένων που δεν συνεισφέρουν στη σωστή λειτουργία της εφαρμογής.

4.5.2 Δομή και Αντικείμενα απαντήσεων

Οι Curators που ασχολούνται με τον καθαρισμό δεδομένων οργανώνονται στον φάκελο `Theatrical.Services > Curators` με την εξής δομή:

- DataCreationCurators > CuratorIncomingData.cs: η κύρια κλάση που χρησιμοποιείται κατά την εισαγωγή δεδομένων.
- Responses > όλες οι κλάσεις απαντήσεων, οι οποίες αναλύονται πιο κάτω.
- Curator.cs: η κύρια κλάση curator.
- RoleSimplifierCurator.cs: η κλάση που απλοποιεί τους ρόλους στη βάση.

4.5.3 DataCreationCurators

Ο φάκελος DataCreationCurators περιλαμβάνει Curators που αναλαμβάνουν τον ρόλο του καθαρισμού των νέων δεδομένων κατά τη διαδικασία εισαγωγής στο σύστημα.

Συγκεκριμένα, η κλάση **CuratorIncomingData**.cs χρησιμοποιείται κατά την εισαγωγή νέων δεδομένων στο σύστημα. Κάθε νέα εγγραφή ενός καλλιτέχνη (Person) στο σύστημα υποβάλλεται σε αυτήν την κλάση καθαρισμού. Η αρμοδιότητα της κλάσης είναι να εξασφαλίζει την εγκυρότητα των πεδίων του αντικειμένου Person, απαλλάσσοντάς τα από περιττά κενά, αχρείαστα σύμβολα και άλλες ανωμαλίες.

Η **CuratorIncomingData** χρησιμοποιεί εκφράσεις Regex για να πραγματοποιήσει σύνθετους ελέγχους και διορθώσεις στα πεδία, όπως για παράδειγμα, τα χρώματα μαλλιών, τα χρώματα ματιών, το βάρος, το ύψος και άλλα. Ο σκοπός είναι η εξασφάλιση της συνολικής ποιότητας και συνέπειας των δεδομένων που εισάγονται στο σύστημα.

4.5.4 Curator Responses

Ο φάκελος Responses περιέχει Curators που δημιουργούν απαντήσεις που παράγονται από το σύστημα. Η γενική κλάση **CuratorResponse** χρησιμοποιείται ως template για τις απαντήσεις σε API requests. Εδώ είναι η δομή της κλάσης:

```
public class CuratorResponse
{
    public int CorrectedObjects { get; set; }
    public int OutOf { get; set; }

    public CuratorResponse(int correctedObjects, int outOf)
    {
        CorrectedObjects = correctedObjects;
        OutOf = outOf;
    }
}
```

Κώδικας 4.2: CuratorResponse.cs

Η παραπάνω κλάση παρέχει ένα γενικό πλαίσιο για τις απαντήσεις των Curators στα API requests, παρέχοντας πληροφορίες σχετικά με τον αριθμό των διορθωμένων αντικειμένων και τον συνολικό αριθμό των αντικειμένων που ελέγχονται.

Στην συνέχεια, υπάρχουν υποκλάσεις που επεκτείνουν τη βασική **CuratorResponse** για να παρέχουν ειδικές απαντήσεις για διάφορα API requests. Παρακάτω παρατίθενται ορισμένα παραδείγματα:

```
public class CurateResponsePeople<T> : CuratorResponse
{
    public T? PeopleCorrected { get; set; }

    public CurateResponsePeople(T? data, int correctedObjects, int outOf) :
base(correctedObjects, outOf)
    {
        PeopleCorrected = data;
    }
}
```

Κώδικας 4.3: CuratorResponsePeople.cs

Η παραπάνω κλάση CurateResponsePeople<T> επεκτείνει τη γενική CuratorResponse και παρέχει ειδική απάντηση για αιτήματα που αφορούν τη διόρθωση δεδομένων σχετικά με άτομα. Περιλαμβάνει τον τύπο T για τα διορθωμένα δεδομένα και πληροφορίες σχετικά με τον αριθμό των διορθωμένων αντικειμένων και τον συνολικό αριθμό των αντικειμένων που ελέγχονται.

Αντίστοιχες υποκλάσεις, όπως CurateResponseContributions<T>, CurateResponseOrganizers<T>, κλπ., προσφέρουν εξειδικευμένες απαντήσεις για τα αντίστοιχα αιτήματα.

Μια πιο γενική κλάση είναι η **CurateResponseEverything**, η οποία παρέχει μια πιο γενική προσέγγιση, συγκεντρώνοντας πληροφορίες σχετικά με τον αριθμό των διορθωμένων αντικειμένων για κάθε κατηγορία, όπως συνεισφορές, διοργανωτές, καλλιτέχνες, παραγωγές, ρόλοι, και χώροι. Αυτό δίνει μια ολοκληρωμένη εικόνα των διορθώσεων που πραγματοποιήθηκαν, προσφέροντας μια ολιστική επισκόπηση των αλλαγών.

4.6 Email Service

Η κλάση **EmailService**.cs βρίσκεται μέσα στο φάκελο `Theatrical.Services > Email` και είναι υπεύθυνη για τη διαχείριση της αποστολής emails στους χρήστες της εφαρμογής.

Συγκεκριμένα, παρέχει μια διεπαφή **IEmailService** που ορίζει διάφορες μεθόδους για την αποστολή email, καθώς και μια υλοποίηση με την κλάση EmailService.

Οι βασικές λειτουργίες της κλάσης περιλαμβάνουν την αποστολή emails για την επιβεβαίωση λογαριασμού, την ενεργοποίηση/απενεργοποίηση της διπαράμετρης πιστοποίησης, την αποστολή προσωρινού κωδικού πρόσβασης, καθώς και ειδοποιήσεις για την έγκριση ή την απόρριψη λογαριασμών. Επιπλέον, υπάρχει η δυνατότητα αποστολής κωδικού διπαράμετρης πιστοποίησης για τη διαδικασία σύνδεσης.

Η κλάση εκμεταλλεύεται τη βιβλιοθήκη SmtpClient για την αποστολή email, χρησιμοποιώντας τον SMTP server της υπηρεσίας Gmail. Ακόμη, για ορισμένες αποστολές χρησιμοποιεί τη βιβλιοθήκη MailKit για επιπλέον λειτουργικότητα και ευελιξία.

Συνολικά, η κλάση **EmailService** συμβάλλει στην ασφαλή και αξιόπιστη διαχείριση των επικοινωνιακών πτυχών της εφαρμογής, εξασφαλίζοντας παράλληλα την απαραίτητη επικοινωνία με τους χρήστες.

Ακολουθεί ένα παράδειγμα, από την κλάση αυτή:

```
public async Task SendConfirmationEmailAsync(string email, string url)
{
    var message = new MailMessage();
    message.From = new MailAddress( userEmail);
    message.To.Add(email);
    message.Subject = "Account Confirmation";

    message.Body = $"Please confirm your email address by clicking the following link: {url}";

    using (var client = new SmtpClient("smtp.gmail.com", 587))
    {
        client.EnableSsl = true;
        client.Credentials = new NetworkCredential(_userEmail, _emailPassword);

        await client.SendMailAsync(message);
    }
}
```

Κώδικας 4.4: Μέθοδος από την κλάση EmailService, η οποία στέλνει email σε χρήστη που κάνει εγγραφή.

```
public interface IEmailService
{
    Task SendConfirmationEmailAsync(string email, string url);
    Task Send2FaVerificationCode(User user, string totpCode);
    Task SendConfirmationEmailTwoFactorActivated(string email);
    Task SendConfirmationEmailTwoFactorDeactivated(string email);
    Task SendTemporaryPassword(string email, string temporaryPassword);
    Task SendApprovalEmail(string email, string personFullname);
    Task SendDisApprovalEmail(string email, string personFullname);
    Task SendRequestConfirmationEmail(string email, string personFullname);
}
```

Κώδικας 4.5: Μέθοδοι που υλοποιούνται στο EmailService

4.7 Pagination Service

Αυτό το κεφάλαιο επικεντρώνεται στον τρόπο που η εφαρμογή μας διαχειρίζεται τη σελιδοποίηση των αποτελεσμάτων. Η σχετική λειτουργικότητα βρίσκεται οργανωμένη μέσα στον φάκελο Pagination. Το service για το την σελιδοποίηση παρέχει την υλοποίηση στο αρχείο PaginationService.cs και μια διεπαφή που ορίζει μια γενική μέθοδο για τον διαχωρισμό των αποτελεσμάτων σε σελίδες σε όλη την εφαρμογή.

Η διεπαφή **IPaginationService** παρέχει μια μέθοδο GetPaginated που χρησιμοποιείται για τη λήψη σελιδοποιημένων αποτελεσμάτων. Αυτή η μέθοδος λαμβάνει τα απαραίτητα ορίσματα όπως ο αριθμός της σελίδας, το πλήθος των αποτελεσμάτων ανά σελίδα, τη συλλογή των αντικειμένων και μια συνάρτηση που μετατρέπει τα αντικείμενα στο επιθυμητό DTO.

Η κλάση **PaginationService** υλοποιεί τη διεπαφή και παρέχει τη λειτουργία του σελιδοποιημένου αποτελέσματος. Βασικά, ο χρήστης μπορεί να ορίσει τη σελίδα που θέλει να δει και τον αριθμό των αποτελεσμάτων που επιθυμεί ανά σελίδα. Αν αυτά τα δύο ορίσματα είναι κενά, τότε επιστρέφονται όλα τα αποτελέσματα χωρίς σελιδοποίηση.

Στη συνέχεια, η κλάση ελέγχει τη συνολική ποσότητα των αποτελεσμάτων και υπολογίζει τον συνολικό αριθμό των σελίδων που θα πρέπει να υπάρχουν, λαμβάνοντας υπόψη τον αριθμό των αποτελεσμάτων ανά σελίδα. Στη συνέχεια, επιστρέφει μόνο τα αποτελέσματα της σελίδας που ζητήθηκε.

4.7.1 Λειτουργία της κλάσης Σελιδοποίησης

Όταν ο χρήστης καλεί τη μέθοδο **GetPaginated**, έχει τη δυνατότητα να διαμορφώσει το αποτέλεσμα ανάλογα με τις ανάγκες του.

Αν και η κλάση προσφέρει μια απλή και γενική λειτουργία, μπορεί επίσης να χρησιμοποιηθεί με διάφορους τρόπους, εξατομικευμένους για τις ανάγκες της εφαρμογής. Η ύπαρξη ενός μετασχηματιστή (mapper) επιτρέπει την ευέλικτη αντιστοίχιση των αντικειμένων σε DTOs ενώ οι παράμετροι σελίδας και μεγέθους δίνουν στον χρήστη τον έλεγχο επί της σελιδοποίησης.

Επίσης, σημειώνεται πως ο κώδικας ελέγχει εάν οι παράμετροι σελίδας και μεγέθους είναι κενές, προσφέροντας μια λειτουργία όπου επιστρέφονται όλα τα αποτελέσματα χωρίς σελιδοποίηση. Παρέχεται επίσης η λειτουργία όπου αν δοθεί μόνο ο αριθμός σελίδας, επιστέφονται αυτόματα τα 10 πρώτα αποτελέσματα της συγκεκριμένης σελίδας. Επιπλέον, αν δοθεί μόνο ο αριθμός του μεγέθους σελίδας, επιστρέφει πάντα την πρώτη σελίδα, με τον καθορισμένο αριθμό αποτελεσμάτων.

Ακολουθεί ο κώδικας της σελιδοποίησης:

```
public interface IPaginationService
{
    10 usages  1 implementation  new *
    PaginationResult<TDto> GetPaginated<T, TDto>(int? page, int? size, IEnumerable<T> items, Func<IEnumerable<T>, IEnumerable<TDto>> mapper);
}
```

Κώδικας 4.6: Η μέθοδος που παρέχει η διεπαφή IPaginationService

```
public class PaginationService : IPaginationService
{
    /// <summary>
    /// /// Pagination Behavior:
    ///           only page: returns the specified page, with 10 results per page,
    ///           only size: returns always the 1st page, with specified sized results.
    /// </summary>
    /// <param name="page">integer</param>
    /// <param name="size">integer</param>
    /// <param name="items">the list of X that you want to paginate</param>
    /// <param name="mapper">map your items that you want to include</param>
    /// <typeparam name="T"></typeparam>
    /// <typeparam name="TDto"></typeparam>
    /// <returns>A PaginationResult with the desired results.</returns>
    0+10 usages  nikosv
    public PaginationResult<TDto> GetPaginated<T, TDto>(int? page, int? size, IEnumerable<T> items, Func<IEnumerable<T>, IEnumerable<TDto>> mapper)
    {
        List<TDto> dtos = new();

        if (page is null && size is null)
        {
            dtos.AddRange( collection: mapper(items));

            var response = new PaginationResult<TDto>
            {
                Results = dtos,
                CurrentPage = null,
                TotalPages = null
            };

            return response;
        }

        size ??= 10;
        page ??= 1;

        var pageResults = (float)size;
        var pageCount = (int)Math.Ceiling(items.Count() / pageResults);

        var itemsPaged IEnumerable<T> = items
            .Skip((page.Value - 1) * (int)pageResults)
            .Take((int)pageResults);

        dtos.AddRange( collection: mapper(itemsPaged));

        var response1 = new PaginationResult<TDto>
        {
            Results = dtos,
            CurrentPage = page,
            TotalPages = pageCount
        };

        return response1;
    }
```

Κώδικας 4.7: PaginationService.cs

4.8 Υπηρεσία Επιβεβαίωσης Τηλεφωνικού Αριθμού Χρήστη

Αυτό το κεφάλαιο επικεντρώνεται στο Service για την επιβεβαίωση τηλεφωνικών αριθμών των χρηστών μέσω μηνυμάτων. Χρησιμοποιείται το πακέτο nuget που παρέχεται από το Twilio, για την αλληλεπίδραση με την εφαρμογή μας. Το TwilioService βρίσκεται μέσα στον φάκελο `PhoneVerification > Twilio`.

4.8.1 Twilio

Το Twilio είναι μια πλατφόρμα cloud communications που παρέχει υπηρεσίες επικοινωνίας μέσω του διαδικτύου. Προσφέρει APIs και εργαλεία για την ενσωμάτωση της επικοινωνίας μέσω τηλεφώνου, SMS, βίντεο και άλλων μέσων στις εφαρμογές. Με το Twilio, οι προγραμματιστές μπορούν να δημιουργήσουν ισχυρές λειτουργίες επικοινωνίας σε εφαρμογές, όπως επιβεβαίωση αριθμού τηλεφώνου, αποστολή SMS, κλήσεις και άλλα.

4.8.2 Υλοποίηση στην Εφαρμογή μας:

Στην εφαρμογή χρησιμοποιείται το Twilio για την υλοποίηση μιας υπηρεσίας επιβεβαίωσης του τηλεφωνικού αριθμού του χρήστη. Κατά τη διαδικασία, ο χρήστης καλείται να εισάγει τον τηλεφωνικό του αριθμό. Έπειτα, το σύστημα του Twilio αποστέλλει έναν κωδικό επιβεβαίωσης μέσω SMS στον καθορισμένο αριθμό. Ο χρήστης εισάγει τον κωδικό που το έχει σταλθεί, στη συνέχεια στέλνεται στο Twilio το οποίο επιστρέφει ένα αποτέλεσμα αν είναι σωστό ή όχι. Το endpoint της εφαρμογής μας `/api/user/request-verification-phone-number`, αλληλεπιδρά με το TwilioService της εφαρμογής μας, το οποίο στέλνει αιτήματα στο Twilio.

Η κλάση **TwilioService** παρέχει μια υλοποίηση της διεπαφής **ITwilioService** και χρησιμοποιείται για την επιβεβαίωση του τηλεφωνικού αριθμού του χρήστη χρησιμοποιώντας την υπηρεσία του Twilio.

Περιγραφή Μεθόδου SendVerificationCode:

- Η μέθοδος αυτή χρησιμοποιεί το Twilio για να στείλει έναν κωδικό επιβεβαίωσης στον τηλεφωνικό αριθμό που περνιέται ως παράμετρος.
- Εάν η κατάσταση της επιβεβαίωσης είναι "pending", τότε επιστρέφεται ένα αντικείμενο ValidationReport με επιτυχία και το μήνυμα "Pending".
- Εάν υπάρξει οποιοδήποτε άλλο πρόβλημα κατά την αποστολή του κωδικού, επιστρέφεται ένα αντικείμενο ValidationReport με αποτυχία, με ένα κατάλληλο μήνυμα σφάλματος και κωδικό σφάλματος Twilio.

Περιγραφή Μεθόδου CheckVerificationCode:

- Η μέθοδος αυτή χρησιμοποιεί το Twilio για να ελέγξει τον κωδικό επιβεβαίωσης που εισήχθη από τον χρήστη.
- Εάν ο έλεγχος επιβεβαίωσης είναι επιτυχής και η κατάσταση είναι "approved", τότε επιστρέφεται ένα αντικείμενο ValidationReport με επιτυχία και το μήνυμα "Approved!".
- Εάν υπάρξει οποιοδήποτε πρόβλημα κατά τον έλεγχο του κωδικού, επιστρέφεται ένα αντικείμενο ValidationReport με αποτυχία, με ένα κατάλληλο μήνυμα σφάλματος και κωδικό σφάλματος Twilio.

Στη συνέχεια ακολουθεί η υλοποίηση, μίας από τις δύο μεθόδους που παρέχονται:

```
public interface ITwilioService
{
    Task<ValidationReport> SendVerificationCode(string phoneNumber);
    Task<ValidationReport> CheckVerificationCode(string phoneNumber, string verificationCode);
}
```

Κώδικας 4.8: Η μέθοδοι που παρέχει η διεπαφή ITwilioService

```
public class TwilioService : ITwilioService
{
    private readonly string? _twilioServiceSid;

    public TwilioService(IConfiguration configuration)
    {
        var twilioAccountSid = configuration["Twilio:AccountSid"];
        var twilioAuthToken = configuration["Twilio:AuthToken"];
        _twilioServiceSid = configuration["Twilio:ServiceSid"];

        TwilioClient.Init(twilioAccountSid, twilioAuthToken);
    }

    public async Task<ValidationReport> CheckVerificationCode(string phoneNumber, string
verificationCode)
    {
        try
        {
            var verificationCheck = await VerificationCheckResource.CreateAsync(
                to: phoneNumber,
                code: verificationCode,
                pathServiceSid: _twilioServiceSid
            );

            if (verificationCheck.Status == "approved")
            {
                return new ValidationReport
                {
                    Success = true,
                    Message = "Approved!"
                };
            }

            var report = new ValidationReport
            {
                Success = false,
                Message = $"There was an error verifying the code:
{verificationCheck.Status}",
                ErrorCode = ErrorCode.TwilioError
            };
            return report;
        }
        catch (Exception)
        {
            var report = new ValidationReport
            {
                Success = false,
                Message = "There was an error verifying the code, please check the code is
correct and try again",
                ErrorCode = ErrorCode.TwilioError
            };
            return report;
        }
    }
}
```

Κώδικας 4.9: Υλοποίηση μεθόδου CheckVerificationCode του TwilioService.cs

Η κλάση διαχειρίζεται επικοινωνία με την υπηρεσία Twilio, ενσωματώνοντας ασφαλή και αξιόπιστη λειτουργικότητα για την επιβεβαίωση του τηλεφωνικού αριθμού των χρηστών.

4.9 Security

Στην ενότητα αυτή αναλύεται η σημαντικότητα της ασφάλειας της εφαρμογής και οι μηχανισμοί που χρησιμοποιούνται, για αυθεντικοποίηση και εξουσιοδότηση των χρηστών κατά την αλληλεπίδραση τους με τα API endpoints. Αυτοί οι μηχανισμοί έχουν οργανωθεί στους φακέλους `Security > Jwt` και `Security > AuthorizationFilters`.

4.9.1 JWT

Τα JWT, ή JSON Web Tokens, είναι συμπαγή, ασφαλή μέσα αναπαράστασης αξιώσεων που μεταφέρονται μεταξύ δύο μερών. Χρησιμοποιούνται συνήθως για έλεγχο ταυτότητας και εξουσιοδότηση στην ανάπτυξη ιστοσελίδων και υπηρεσιών. Τα JWT αποτελούνται από τρία μέρη: μια κεφαλίδα (Header), ένα φορτίο (Payload) και μια υπογραφή (Signature).

Συγκεκριμένα, ο φάκελος `Security > Jwt` περιλαμβάνει λειτουργίες σχετικές με τη δημιουργία, την επικύρωση και τη διαχείριση των JWT (JSON Web Tokens).

Πιο ειδική ανάλυση περιγράφεται παρακάτω:

Header:

Η επικεφαλίδα αποτελείται συνήθως από δύο μέρη: τον τύπο του token, ο οποίος είναι JWT, και τον αλγόριθμο υπογραφής που χρησιμοποιείται, όπως HMAC SHA256 ή RSA. Κωδικοποιείται με Base64Url για να σχηματίσει το πρώτο μέρος του JWT.

Payload:

Το δεύτερο μέρος του token είναι το φορτίο, το οποίο περιέχει αξιώσεις. Οι αξιώσεις είναι δηλώσεις σχετικά με μια οντότητα (συνήθως τον χρήστη) και πρόσθετα δεδομένα, όπως ο ρόλος του στη βάση, το email του, και άλλα. Υπάρχουν τρεις τύποι ισχυρισμών: registered, public, private claims.

- Registered Claims: Πρόκειται για ισχυρισμούς που δεν είναι καταχωρημένοι αλλά ορίζονται στην προδιαγραφή JWT ως δημόσιοι. Ορισμένα παραδείγματα περιλαμβάνουν “iss” (εκδότης), “exp” (χρόνος λήξης), “sub” (υποκείμενο) και άλλα. Οι καταχωρημένες αξιώσεις παρέχουν έναν τυποποιημένο τρόπο μετάδοσης πληροφοριών σχετικά με το token.

- Public Claims: Οι δημόσιοι ισχυρισμοί στο φορτίο JWT είναι προσαρμοσμένοι ισχυρισμοί που προορίζονται για δημόσια κατανάλωση. Μπορούν να περιέχουν γενικές πληροφορίες όπως το όνομα και το email ενός χρήστη. Πρόκειται για ισχυρισμούς που δεν είναι καταχωρημένοι αλλά ορίζονται στην προδιαγραφή του JWT ως δημόσιοι.

- Private Claims: Πρόκειται για προσαρμοσμένους ισχυρισμούς που δημιουργούνται για την ανταλλαγή πληροφοριών μεταξύ των μερών που συμφωνούν στη χρήση τους και δεν είναι ούτε καταχωρισμένοι ούτε δημόσιοι ισχυρισμοί. Η ονομασία τους πρέπει να συμφωνηθεί από τα εμπλεκόμενα προς χρήση μέρη.

Ακολουθεί ένα παράδειγμα ωφέλιμου φορτίου JWT που περιλαμβάνει καταχωρημένες, δημόσιες και ιδιωτικές αξιώσεις:

```
{
"sub": "1234567890",          // Registered claim (Subject)
"name": "John Doe",           // Public claim
"admin": true,                // Private claim
"exp": 1672531199,            // Registered claim (Expiration Time)
"custom_data": {
    "role": "developer",        // Private claim within a custom_data object
    "account_type": "premium"   // Private claim within a custom_data object
 }
}
```

Κώδικας 4.10: Παράδειγμα JWT Payload.

4.9.2 Χρήση JWT

Η κλάση **TokenService** παρέχει μια υλοποίηση του **ITokenService**. Παρέχει λειτουργίες για τη δημιουργία και επαλήθευση JWT στο πλαίσιο της αυθεντικοποίησης και εξουσιοδότησης των χρηστών στην εφαρμογή. Αναλύονται παρακάτω λεπτομερώς οι δύο μέθοδοι:

GenerateToken(User user):

- Η μέθοδος αυτή δημιουργεί ένα JWT για έναν συγκεκριμένο χρήστη.
- Πρώτα, καθορίζονται τα διάφορα claims που θα περιλαμβάνονται στο JWT. Στο παράδειγμα, περιλαμβάνεται το email του χρήστη και ο ρόλος του.
- Στη συνέχεια, ορίζονται τα χαρακτηριστικά του JWT, όπως ο χρόνος λήξης, ο αλγόριθμος υπογραφής και άλλες παράμετροι.
- Ο τελικός JWT δημιουργείται και επιστρέφεται ως αντικείμενο JwtDto.

VerifyToken(string token):

- Η μέθοδος αυτή επαληθεύει ένα περασμένο JWT.
- Κατασκευάζονται παράμετροι επαλήθευσης όπως το κλειδί υπογραφής, ο εκδότης, ο παραλήπτης και άλλες.
- Το JWT επαληθεύεται χρησιμοποιώντας τον JwtSecurityTokenHandler.
- Εάν η επαλήθευση είναι επιτυχής, επιστρέφεται ένα ClaimsPrincipal που περιέχει τις πληροφορίες του χρήστη. Σε διαφορετική περίπτωση, επιστρέφεται null.

Ακολουθεί ο κώδικας υλοποίησης της **ITokenService**:

```
public interface ITokenService
{
    JwtDto GenerateToken(User user);
    ClaimsPrincipal? VerifyToken(string token);
}
public class TokenService : ITokenService
{
    private static readonly TimeSpan TokenLifetime = TimeSpan.FromHours(1);
    private readonly IConfiguration _config;

    public TokenService(IConfiguration config)
    {
        _config = config;
    }
    public JwtDto GenerateToken(User user)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
```

```
        var jwtOptions = _config.GetSection("JwtOptions").Get<JwtOptions>();
        var key = Encoding.UTF8.GetBytes(jwtOptions.SigningKey);

        var claims = new List<Claim>
        {
            new(JwtRegisteredClaimNames.Email, user.Email)
        };

        var userIntRole = user.UserAuthorities.FirstOrDefault()?.AuthorityId;

        var userRole = userIntRole switch
        {
            1 => "admin",
            2 => "user",
            3 => "developer",
            4 => "claims manager",
            _ => "Invalid role"
        };

        claims.Add(new Claim(ClaimTypes.Role, userRole));

        var securityKey = new SymmetricSecurityKey(key);
        var signingCredentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(claims),
            Expires = DateTime.UtcNow.Add(TokenLifetime),
            SigningCredentials = signingCredentials,
            Issuer = jwtOptions.Issuer,
            Audience = jwtOptions.Audience
        };

        var token = tokenHandler.CreateToken(tokenDescriptor);
        var tokenString = tokenHandler.WriteToken(token);
        var jwtDto = new JwtDto
        {
            access_token = tokenString,
            token_type = "bearer",
            expires_in = (int)TokenLifetime.TotalSeconds
        };

        return jwtDto;
    }
    public ClaimsPrincipal? VerifyToken(string token)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var jwtOptions = _config.GetSection("JwtOptions").Get<JwtOptions>();
        var key = Encoding.UTF8.GetBytes(jwtOptions.SigningKey);

        var validationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = true,
            ValidIssuer = jwtOptions.Issuer,
            ValidateAudience = true,
            ValidAudience = jwtOptions.Audience,
            ValidateLifetime = true
        };

        try
        {
            var principal = tokenHandler.ValidateToken(token, validationParameters, out  );
            return principal;
        }
        catch (Exception ex)
        {
            return null;
        }
    }

}
```

Κώδικας 4.11: Η υλοποίηση της διεπαφής ITokenService

4.9.3 Authorization Filters

Τα Φίλτρα εξουσιοδότησης αποτελούν ένα σημαντικό κομμάτι για την ασφάλεια, καθώς ρυθμίζουν την πρόσβαση σε συγκεκριμένα endpoints της εφαρμογής. Συνήθως, χρησιμοποιούνται για να ελέγξουν την έγκυρη εξουσιοδότηση των αιτήσεων από τους χρήστες. Ο κώδικας των φίλτρων οργανώνεται στον φάκελο `Security > AuthorizationFilters`.

Η υλοποίηση της βασικής κλάσης **AuthorizationFilterBase** χρησιμοποιεί το **IAuthorizationFilter** από το namespace **Microsoft.AspNetCore.Mvc.Filters**. Αυτή η κλάση αποτελεί τη βάση για τη δημιουργία φίλτρων εξουσιοδότησης και περιέχει λογική που εφαρμόζεται πριν από την πρόσβαση σε ένα API endpoint. Εξετάζονται τα κύρια στοιχεία:

RequiredRole: Αυτή η ιδιότητα αφορά τους ρόλους που απαιτούνται για πρόσβαση. Είναι μια αφηρημένη ιδιότητα που πρέπει να υλοποιηθεί στις υποκλάσεις, καθορίζοντας ποιους ρόλους πρέπει να έχει ο χρήστης.

Constructor: Ο κατασκευαστής δέχεται ένα αντικείμενο ITokenService, που χρησιμοποιείται για τον έλεγχο της εγκυρότητας του JWT token.

OnAuthorization: Αυτή η μέθοδος εκτελεί τους ακόλουθους ελέγχους:

- Έλεγχος Token: Έλεγχος αν υπάρχει το JWT token και αν έχει σωστή μορφή.
- Έλεγχος Εγκυρότητας Token: Χρήση του ITokenService για τον έλεγχο της εγκυρότητας του token.
- Έλεγχος Ρόλων: Έλεγχος αν ο χρήστης έχει τους απαιτούμενους ρόλους για πρόσβαση.

Σε περίπτωση που κάποιος από αυτούς τους ελέγχους αποτύχει, η μέθοδος διαμορφώνει και επιστρέφει ένα αντίστοιχο αντικείμενο ObjectResult που περιέχει ένα αντικείμενο ApiResponse με το κατάλληλο μήνυμα σφάλματος και κατάσταση HTTP (Http Status Code).

Η κλάση **AuthorizationFilterBase**, δίνεται παρακάτω:

```
public abstract class AuthorizationFilterBase : IAuthorizationFilter
{
    protected abstract string[] RequiredRole { get; }
    private readonly ITokenService _tokenService;

    protected AuthorizationFilterBase(ITokenService tokenService)
    {
        _tokenService = tokenService;
    }

    public void OnAuthorization(AuthorizationFilterContext context)
    {
        string? bearerToken = context.HttpContext.Request.Headers["Authorization"];

        if (string.IsNullOrEmpty(bearerToken))
        {
            var errorResponse = new ApiResponse(ErrorCode.Unauthorized,"You did not provide a
JWT token");
            context.Result = new ObjectResult(errorResponse){StatusCode =
(int)HttpStatusCode.Unauthorized};
            return;
        }

        if (!bearerToken.StartsWith("Bearer "))
        {
            var errorResponse1 = new ApiResponse(ErrorCode.BadRequest, "Correct Format: Bearer
YourToken");
            context.Result = new ObjectResult(errorResponse1) { StatusCode =
(int)HttpStatusCode.BadRequest };
            return;
        }
```

```
        string jwtToken = bearerToken.Substring(7);
        var claimsPrincipal = _tokenService.VerifyToken(jwtToken);

        if (claimsPrincipal is null)
        {
            var errorResponse2 = new ApiResponse(ErrorCode.InvalidToken, "Invalid Token");
            context.Result = new ObjectResult(errorResponse2) { StatusCode =
(int)HttpStatusCode.Unauthorized };
            return;
        }

        if (!RequiredRole.Any(role => context.HttpContext.User.IsInRole(role)))
        {
            var errorResponse3 = new ApiResponse(ErrorCode.Forbidden, "You are not allowed to
make changed or see the context of this request");
            context.Result = new ObjectResult(errorResponse3) { StatusCode =
(int)HttpStatusCode.Forbidden };
            return;
        }
    }
}
```

Κώδικας 4.12: Η αφηρημένη κλάση AuthorizationFilterBase.

4.9.3.1 Υποκλάσεις

Οι υποκλάσεις (φίλτρα) που υπάρχουν στην εφαρμογή είναι οι εξής: **AdminAuthorizationFilter, AnyRoleAuthorizationFilter, ClaimsManagerAndAdminAuthorizationFilter, ClaimsManagerAuthorizationFilter, UserAuthorizationFilter.** Να σημειωθεί ότι μπορούν να προσθεθούν επιπλέον υποκλάσεις με άλλους ρόλους ή συνδυασμό ρόλων, απλά επεκτείνοντας από τη βασική κλάση, και να χρησιμοποιηθούν πολύ εύκολα στα endpoints του API.

Η υποκλάση **AdminAuthorizationFilter** επεκτείνει την βασική **AuthorizationFilterBase** και ορίζει τις πρόσθετες απαιτήσεις ρόλων για την πρόσβαση. Παρακάτω παρουσιάζονται τα βασικά χαρακτηριστικά της υποκλάσης:

RequiredRole Property: Στην υποκλάση, η ιδιότητα RequiredRole παρακάμπτει και επιστρέφει έναν πίνακα με έναν μόνο ρόλο, τον ρόλο "admin". Αυτό σημαίνει ότι επιτρέπεται η πρόσβαση μόνο στους χρήστες που έχουν το ρόλο "admin".

Constructor: Ο κατασκευαστής δεν παίρνει παραμέτρους εκτός από το ITokenService, το οποίο περνά στον γονικό κατασκευαστή (base(tokenService)).

Ο κώδικας παρουσιάζεται παρακάτω:

```
using Theatrical.Services.Security.Jwt;

namespace Theatrical.Services.Security.AuthorizationFilters;

public class AdminAuthorizationFilter : AuthorizationFilterBase
{
    protected override string[] RequiredRole => new[] {"admin"};

    public AdminAuthorizationFilter(ITokenService tokenService) : base(tokenService)
    {
    }
}
```

Κώδικας 4.13: Το φίλτρο για το admin role.

Με παρόμοιο τρόπο είναι δομημένες και οι υπόλοιπες υποκλάσεις και δεν απαιτείται περαιτέρω ανάλυση.

4.10 Repository

Στο κεφάλαιο 4.10, παρουσιάζεται η υλοποίηση του Repository Pattern στην εφαρμογή μας. Το Repository Pattern, ένα δημοφιλές σχέδιο αρχιτεκτονικής, επιτρέπει τον διαχωρισμό της λογικής της επικοινωνίας με τη βάση δεδομένων από τον κώδικα της εφαρμογής. Αυτή η προσέγγιση καθιστά τον κώδικα πιο ευανάγνωστο, συντηρήσιμο και επαναχρησιμοποιήσιμο. Το κεφάλαιο παρέχει μια λεπτομερή εξέταση των κλάσεων που εφαρμόζουν το Repository Pattern και διευκρινίζει τον τρόπο που αυτές αλληλεπιδρούν με το DbContext της εφαρμογής, προσφέροντας ένα ολοκληρωμένο και οργανωμένο σύστημα διαχείρισης των δεδομένων. Ο κώδικας των Repositories μπορεί να βρεθεί μέσα στον φάκελο `Repositories` του `Theatrical.Services`. Κάποιες κλάσεις ενδέχεται να χρησιμοποιούν το Caching service που προαναφέρθηκε.

4.10.1 UserRepository

Η διεπαφή **IUserRepository** περιέχει μια σειρά από μεθόδους που επιτρέπουν τη διαχείριση και αλληλεπίδραση με τα δεδομένα των χρηστών στη βάση δεδομένων. Κάποιες από αυτές τις μεθόδους:

- Task<User?> Get(string email): Επιστρέφει έναν χρήστη με βάση το email του.
- Task<User?> Get(int id): Επιστρέφει έναν χρήστη με βάση το μοναδικό του αναγνωριστικό (id).
- Task<User?> GetByUsername(string username): Επιστρέφει έναν χρήστη με βάση το όνομα χρήστη του.
- Task<User> Register(User user, int userRole): Εγγράφει ένα νέο χρήστη στη βάση δεδομένων, συμπεριλαμβανομένου του ρόλου χρήστη.
- Task<decimal> GetUserBalance(int id): Επιστρέφει το υπόλοιπο του χρήστη με βάση το μοναδικό του αναγνωριστικό.
- Task EnableAccount(User user): Επιβεβαιώνει το email του χρήστη. Αφού ο χρήστης πατήσει στο email επιβεβαίωσης.
- Task<User?> SearchToken(string token): Αναζητά ένα χρήστη βάσει ενός δοθέντος token.
- Task<User?> SearchOtp(string otp): Αναζητά ένα χρήστη βάσει ενός δοθέντος κωδικού OTP.

Η κλάση **UserRepository** υλοποιεί τη διεπαφή **IUserRepository** και διαχειρίζεται την επικοινωνία με τη βάση δεδομένων για τις λειτουργίες που σχετίζονται με τους χρήστες. Ας αναλύσουμε τις πρώτες μεθόδους:

public async Task<User?> GetByUsername(string username): Αυτή η μέθοδος χρησιμοποιεί το Entity Framework για να αναζητήσει και να επιστρέψει έναν χρήστη βάσει του ονόματος χρήστη του ή αν δεν βρεθεί επιστρέφει κενό (null).

public async Task<User?> Get(string email): Αντίστοιχα, αυτή η μέθοδος αναζητά έναν χρήστη βάσει του email του.

public async Task<User?> Get(int id): Αυτή η μέθοδος αναζητά και επιστρέφει έναν χρήστη βάσει του μοναδικού αναγνωριστικού του.

Οι μέθοδοι χρησιμοποιούν ασύγχρονες λειτουργίες (async Task) για να διαχειριστούν αποτελέσματα από τη βάση δεδομένων, επιτρέποντας την αποτελεσματική διαχείριση των αιτημάτων σε περιβάλλοντα πολλαπλής νηματικότητας (multi-threading).

Επιπλέον, η κλάση δέχεται στον constructor της το TheatricalPlaysDbContext και το ILogRepository. Το TheatricalPlaysDbContext παρέχει πρόσβαση στη βάση δεδομένων, ενώ το ILogRepository πιθανόν χρησιμοποιείται για την καταγραφή καταγραφής σε μια αρχείο καταγραφής ή άλλες λειτουργίες καταγραφής.

Στη συνέχεια παρουσιάζεται ο κώδικας υλοποίησης κάποιων μεθόδων:

```
public interface IUserRepository
{
    Task<User?> Get(string email);
    Task<User?> Get(int id);
    Task<User?> GetByUsername(string username);
    Task<User> Register(User user, int userRole);
    Task<User?> GetUserIncludingAuthorities(string email);
    Task<decimal> GetUserBalance(int id);
    Task EnableAccount(User user);
    Task<User?> SearchToken(string token);
    Task<User?> SearchOtp(string otp);
    Task Update2Fa(User user, string otp);
    Task Activate2Fa(User user, string userSecret);
    Task Deactivate2Fa(User user);
    Task<User?> GetUserAuthoritiesAndTransactions(string email);
    Task UpdateFacebook(User user, string link);
    Task UpdateInstagram(User user, string link);
    Task UpdateYoutube(User user, string link);
    Task RemoveFacebook(User user);
    Task RemoveYoutube(User user);
    Task RemoveInstagram(User user);
    Task OnRequestApproval(User user, Person person);
    Task UpdateUsername(User user, string username);
    Task UpdatePassword(User user, string hashedPassword);
    Task UploadPhoto(UserImage userImage);
    Task AddRole(User user, string role);
    Task RemoveRole(User user, string role);
    Task<List<UserImage>?> GetUserImages(int userId);
    Task RemoveUserImage(UserImage userImage);
    Task<UserImage?> GetUserImage(int imageId);
    Task<UserImage?> GetFirstProfileImage(int userId);
    Task UnsetProfilePhoto(UserImage userImage);
    Task SetProfilePhoto(UserImage userImage);
    Task SetBioPdfLocation(User user, string location);
    Task UnsetBio(User user);
    Task UpdateVerifiedPhoneNumber(User user, string number);
    Task RegisterPhoneNumber(User user, string phoneNumber);
    Task RemovePhoneNumber(User user);
}
public class UserRepository : IUserRepository
{
    private readonly TheatricalPlaysDbContext _context;
    private readonly ILogRepository _logRepository;

    public UserRepository(TheatricalPlaysDbContext context, ILogRepository logRepository)
    {
        _context = context;
        _logRepository = logRepository;
    }

    public async Task<User?> GetByUsername(string username)
    {
        return await _context.Users.FirstOrDefaultAsync(u => u.Username == username);
    }

    public async Task<User?> Get(string email)
    {
        return await _context.Users.FirstOrDefaultAsync(u => u.Email == email);
    }

    public async Task<User?> Get(int id)
    {
        return await _context.Users.FindAsync(id);
    }
```

Κώδικας 4.14: Η διεπαφή IUserRepository και η υλοποίηση κάποιων μεθόδων του στην UserRepository.

Στην συνέχεια περιγράφεται η εγγραφή χρήστη στο πλαίσιο που έχει άμεση πρόσβαση με την βάση:

Η μέθοδος **Register** υπεύθυνη για την εγγραφή νέου χρήστη στη βάση δεδομένων, παρουσιάζει τα εξής βήματα:

- await _context.Users.AddAsync(user);: Αυτό το βήμα προσθέτει τον νέο χρήστη (user) στην συλλογή των χρηστών στο TheatricalPlaysDbContext. Η χρήση της AddAsync δηλώνει ότι η ενέργεια πρέπει να εκτελεστεί ασύγχρονα.

- await _context.SaveChangesAsync();: Αυτή η μέθοδος αποθηκεύει τις αλλαγές στη βάση δεδομένων. Εδώ, μετά την προσθήκη του νέου χρήστη, αποθηκεύονται οι αλλαγές στη βάση.

- await _logRepository.UpdateLogs("insert", "users", ...);: Το UpdateLogs καλείται για να ενημερώσει τα logs του συστήματος με την εισαγωγή νέου χρήστη. Παρέχονται λεπτομέρειες όπως το id και το email του νέου χρήστη.

- Στη συνέχεια, δημιουργείται ένα νέο αντικείμενο UserAuthority που συνδέει τον νέο χρήστη με το ρόλο που του έχει ανατεθεί. Ανάλογα με την τιμή της μεταβλητής userRole (1 για admin, 2 για user, 3 για developer), ο ρόλος του χρήστη ορίζεται.

- await _context.UserAuthorities.AddAsync(userAuthorities);: Το αντικείμενο UserAuthority προστίθεται στη συλλογή των user authorities στο TheatricalPlaysDbContext.

- await _context.SaveChangesAsync();: Πάλι, εκτελείται η SaveChangesAsync για να αποθηκευτούν οι αλλαγές στη βάση δεδομένων, περιλαμβάνοντας και την εισαγωγή του νέου UserAuthority.

```
public async Task<User> Register(User user, int userRole)
{
    await _context.Users.AddAsync(user);            //adds user
    await _context.SaveChangesAsync();

    await  logRepository.UpdateLogs("insert", "users", new List<(string ColumnName, string
Value)>
    {
        ("id", user.Id.ToString()),
        ("email", user.Email)
    });

    var userAuthorities = new UserAuthority         //adds user authorities
    {
        UserId = user.Id,
        AuthorityId = userRole                      //1 for admin, 2 for user, 3 for developer
    };

    await _context.UserAuthorities.AddAsync(userAuthorities);
    await _context.SaveChangesAsync();

    return user;
}
```

Κώδικας 4.15: Η υλοποίηση της Register μεθόδου του στην UserRepository.

4.10.2 PersonRepository

Με παρόμοια λογική είναι οργανωμένο και αυτό το Repository. Περιέχει μεθόδους για την διαχείριση των καλλιτεχνών (Person) στη βάση. Παρουσιάζονται και εξετάζονται κάποιες από τις μεθόδους που περιέχονται:

IPersonRepository:

```
public interface IPersonRepository
{
    Task<Person> Create(Person person);
    Task<List<Person>> Get();
    Task Delete(Person person);
    Task<Person?> Get(int id);
    Task<List<Person>?> GetByRole(string role);
    Task<List<Person>?> GetByLetter(string initials);
    Task<Person?> GetByName(string name);
    Task<List<PersonProductionsRoleInfo>?> GetProductionsOfPerson(int personId);
    Task<List<Image>?> GetPersonsImages(int personId);
    Task UpdateRange(List<Person> people);
    Task<List<Image>?> GetImages();
    Task CreateRequest(Person person);
    Task ApproveRequest(RequestActionDto requestActionDto);
    Task RejectRequest(RequestActionDto requestActionDto);
    Task DeleteTestData();
    Task<List<Person>?> GetByNameRange(List<CreatePersonDto> persons);
    Task<List<Person>> CreateRange(List<Person> finalPeopleToAdd);
    Task SaveListChanges();
}
```

Κώδικας 4.16: Η διεπαφή IPersonRepository.

Create: Αυτή η μέθοδος χρησιμοποιείται για τη δημιουργία μιας νέας εγγραφής Person στη βάση δεδομένων. Λαμβάνει ένα αντικείμενο Person ως παράμετρο και επιστρέφει το νέο αντικείμενο Person που δημιουργήθηκε.

Get: Αυτή η μέθοδος επιστρέφει μια λίστα όλων των αντικειμένων Person που υπάρχουν στη βάση δεδομένων.

Delete: Χρησιμοποιείται για τη διαγραφή ενός συγκεκριμένου αντικειμένου Person από τη βάση δεδομένων.

Get(int id): Επιστρέφει το αντικείμενο Person με βάση το συγκεκριμένο id.

GetByRole(string role): Επιστρέφει μια λίστα αντικειμένων Person βάσει του ρόλου που έχουν στην παράσταση ή το έργο.

GetByLetter(string initials): Επιστρέφει μια λίστα αντικειμένων Person βάσει των αρχικών του ονόματος.

GetByName(string name): Επιστρέφει το αντικείμενο Person με βάση το όνομα.

GetProductionsOfPerson(int personId): Επιστρέφει μια λίστα με τις παραστάσεις (productions) στις οποίες συμμετέχει ο συγκεκριμένος άνθρωπος.

GetPersonsImages(int personId): Επιστρέφει μια λίστα με τις εικόνες που σχετίζονται με τον συγκεκριμένο άνθρωπο.

UpdateRange(List<Person> people): Χρησιμοποιείται για την ενημέρωση μιας λίστας αντικειμένων Person.

GetImages(): Επιστρέφει μια λίστα με όλες τις εικόνες που υπάρχουν στη βάση δεδομένων.

Συνεχίζοντας παρουσιάζονται μερικές από τις μεθόδους που υλοποιεί το **PersonRepotisory**:

```
public class PersonRepository : IPersonRepository
{
    private readonly TheatricalPlaysDbContext _context;
    private readonly ICaching _caching;

    public PersonRepository(TheatricalPlaysDbContext context, ICaching caching)
    {
        _context = context;
        _caching = caching;
    }

    public async Task<Person> Create(Person person)
    {
        await _context.Persons.AddAsync(person);
        await _context.SaveChangesAsync();
        return person;
    }

    public async Task<List<Person>> Get()
    {
        var people = await _caching.GetOrSetAsync("allpeople",
            async () => await _context.Persons
                .AsNoTracking()
                .Include(p => p.Images)
                .ToListAsync());

        return people;
    }

    public async Task<Person?> Get(int id)
    {
        var person = await _caching.GetOrSetAsync($"person_{id}", async () => await
_context.Persons.FindAsync(id));

        return person;
    }

    public async Task DeleteTestData()
    {
        var testPersons = await _context.Persons.Where(p => p.SystemId == 14).ToListAsync();
        _context.Persons.RemoveRange(testPersons);
        await _context.SaveChangesAsync();
    }

    public async Task ApproveRequest(RequestActionDto requestActionDto)
    {
        requestActionDto.Person.IsClaimed = true;
        requestActionDto.Person.ClaimingStatus = ClaimingStatus.Unavailable;

        await _context.SaveChangesAsync();
    }

    public async Task RejectRequest(RequestActionDto requestActionDto)
    {
        requestActionDto.Person.ClaimingStatus = ClaimingStatus.Available;
        await _context.SaveChangesAsync();
    }

    public async Task<List<Person>?> GetByRole(string role)
    {
        var people = await _caching.GetOrSetAsync($"persons_with_role_{role}", async () =>
        {
            return await _context.Persons
                .Where(p => p.Contributions.Any(c => c.Role.Role1 == role))
                .ToListAsync();
        });

        return people;
    }
```

Κώδικας 4.17: Η υλοποίηση PersonRepository.

Οι υπόλοιπες κλάσεις Repository είναι παρόμοιες και θα δοθούν ενδεικτικά.

```
public interface IAccountRequestRepository
{
    Task<AccountRequest> CreateRequest(AccountRequest accountRequest);
    Task<List<AccountRequest>> GetAll();
    Task ApproveRequest(RequestActionDto requestActionDto);
    Task RejectRequest(RequestActionDto requestActionDto);
    Task<AccountRequest?> Get(int requestId);
}
```

Κώδικας 4.18: Διεπαφή IAccountRequestRepository.

```
public interface IAssignedUserRepository
{
    Task AddAssignedPerson(AssignedUser? assignedUser);
    Task<AssignedUser?> GetByUserId(int userId);
    Task<Person?> GetClaimedPersonForUser(int userId);
}
```

Κώδικας 4.19: Διεπαφή IAssignedUserRepository.

```
public interface IContributionRepository
{
    Task<List<Contribution>> Get();
    Task<Contribution> Create(Contribution contribution);
    Task<List<Contribution>> GetSpecific(int personId, int productionId, int roleId);
    Task<List<Contribution>> GetByRole(int roleId);      //Methods with grey color are
implemented but not user.
    Task<List<Contribution>> GetByProduction(int productionId);
    Task<List<Contribution>> GetByPerformer(int personId);

    Task<(bool productionExists, bool performerExists, bool roleExists)> CheckExists(int
performerId, int productionId,
        int roleId);

    Task UpdateRange(List<Contribution> contributions);
    Task RemoveRange(List<Contribution> contributions);
    Task<List<Contribution>> CreateRange(List<Contribution> contributionsToCreate);
}
```

Κώδικας 4.20: Διεπαφή IContributionRepository.

```
public interface IEventRepository
{
    Task<List<Event>?> Get();
    Task<Event?> GetEvent(int id);
    Task<Event> Create(Event newEvent);
    Task Delete(Event deletingEvent);
    Task UpdatePriceEvent(Event @event, UpdateEventDto eventDto);
    Task<List<Event>> GetEventsForPerson(int personId);
    Task<List<Event>?> GetEventsForProduction(int productionId);
    Task<List<Event>> CreateEvents(List<Event> events);
    Task<List<Show>> GetShows();
    Task<Event> UpdateEvent(Event eventToUpdate);
}
```

Κώδικας 4.21: Διεπαφή IEventRepository.

```
public interface ILogRepository
{
    Task<List<ChangeLog>> GetLogs();
    Task UpdateLogs(string eventType, string tableName, List<(string ColumnName, string
Value)> columns);
}
```

Κώδικας 4.22: Διεπαφή ILogRepository.

```
public interface IOrganizerRepository
{
    Task<Organizer?> Get(int id);
    Task<List<Organizer>?> Get();
    Task Create(Organizer organizer);
    Task Delete(Organizer organizer);
    Task<List<Organizer>> UpdateRange(List<Organizer> organizers);
    Task<List<Organizer>> GetOrganizersByNames(List<string> organizersNames);
    Task<List<Organizer>> CreateRange(List<Organizer> organizers);
}
```

Κώδικας 4.23: Διεπαφή IOrganizerRepository.

```
public interface IProductionRepository
{
   Task Create(Production production);
   Task<List<Production>?> Get();
   Task<Production?> GetProduction(int id);
   Task Delete(Production production);
   Task<List<Production>> UpdateRange(List<Production> productions);
   Task<List<Production>> GetProductionsByTitles(List<string> productionsTitles);
   Task<List<Production>> CreateRange(List<Production> productions);
}
```

Κώδικας 4.24: Διεπαφή IProductionRepository.

```
public interface IRoleRepository
{
    Task<List<Role>> GetRoles();
    Task<Role?> GetRole(int id);
    Task<Role> CreateRole(Role roles);
    Task DeleteRole(Role roles);
    Task<Role?> GetRoleByName(string searchRole);
    Task<List<Role>> UpdateRange(List<Role> roles);
    Task<List<Role>> CreateRoleRange(List<Role> rolesToAdd);
}
```

Κώδικας 4.25: Διεπαφή IRoleRepository.

```
public interface ITransactionRepository
{
    Task<Transaction> PostTransaction(Transaction transaction);
    Task<List<Transaction>> GetTransactions(int userId);
    Task<Transaction?> GetTransaction(int transactionId);
    Task<List<User>> GetUsersWithVerifiedEmailNotPaid();
    Task PostTransactions(List<Transaction> transactions);
}
```

Κώδικας 4.26: Διεπαφή ITransactionRepository.

```
public interface IUserEventRepository
{
    Task<List<UserEvent>> GetUserEventsAsync();
    Task<List<Event>?> GetClaimedEventsByUser(int userId);
    Task<User?> GetUserWithEvents(string email);
    Task Claim(Event @event);
    Task Create(UserEvent userEvent);
}
```

Κώδικας 4.27: Διεπαφή IUserEventRepository.

```
public interface IUserVenueRepository
{
    Task<User?> GetUserWithVenues(string email);
    Task Create(UserVenue userVenue);
    Task Claim(Venue venue);
    Task<List<Venue>?> GetClaimedVenuesForUser(int userId);
}
```

Κώδικας 4.28: Διεπαφή IUserVenueRepository.

```
public interface IVenueRepository
{
    Task<List<Venue>?> Get();
    Task<Venue?> Get(int id);
    Task<Venue> Create(Venue venue);
    Task Delete(Venue venue);
    Task<Venue> Update(Venue venue, VenueUpdateDto venueUpdateDto);
    Task<List<Venue>> UpdateRange(List<Venue> venues);
    Task<List<Production>?> GetVenueProductions(int venueId);
    Task<List<Venue>> GetVenuesByTitles(List<string> titles);
    Task<List<Venue>> CreateRange(List<Venue> venues);
    Task<Venue?> GetVenueByTitle(string venueTitle);
}
```

Κώδικας 4.29: Διεπαφή IVenueRepository.

4.11 Κλάσεις για Επικύρωση Δεδομένων

Συνεχίζοντας, σε αυτό το κεφάλαιο αναλύονται οι κλάσεις που χρησιμοποιούνται για την επικύρωση των δεδομένων στην εφαρμογή. Η επικύρωση δεδομένων αποτελεί ένα σημαντικό κομμάτι στη διασφάλιση της ορθότητας και της αξιοπιστίας των πληροφοριών που εισάγονται στο σύστημα.

Εξετάζονται διάφορες κλάσεις που εκτελούν διαδικασίες επικύρωσης, καλύπτοντας διάφορες πτυχές των δεδομένων που εισάγονται. Οι κλάσεις επικύρωσης δεδομένων συνδέονται και έχουν πρόσβαση στις κλάσεις του Repository για ανάκτηση των δεδομένων από τη βάση. Οι μέθοδοι των κλάσεων επικύρωσης χρησιμοποιούνται μόνο από τις μεθόδους των κλάσεων των Controller του API.

4.11.1 Κλάση UserValidationService

Η κλάση **UserValidationService** είναι υπεύθυνη για την επικύρωση δεδομένων που σχετίζονται με τον χρήστη. Εκτελεί απαραίτητους ελέγχους για την ορθότητα των πληροφοριών που σχετίζονται με τη δημιουργία ή τροποποίηση λογαριασμού χρήστη. Εκτελεί την υλοποίηση της διεπαφής **IUserValidationService**.

Εξετάζονται μερικές επιλεγμένες μέθοδοι.

4.11.1.1 Μέθοδος ValidateForRegister

Η μέθοδος **ValidateForRegister** εκτελεί διάφορους ελέγχους για την επικύρωση των δεδομένων που χρησιμοποιούνται κατά την εγγραφή νέου χρήστη. Ας αναλύσουμε τα βήματα που περιέχει:

Επικύρωση της μορφής του email:

- Χρησιμοποιείται ένα regex για να ελεγχθεί αν το email έχει τη σωστή μορφή.
- Αν το email δεν πληροί τα κριτήρια της μορφής, επιστρέφεται ένα αντικείμενο ValidationReport με κατάλληλο μήνυμα λάθους και κωδικό λάθους InvalidEmail.

Έλεγχος για ύπαρξη του email στη βάση δεδομένων:

- Κάνει χρήση του IUserRepository για να αναζητήσει έναν χρήστη με βάση το δοσμένο email.
- Αν υπάρχει ήδη χρήστης με το συγκεκριμένο email, επιστρέφεται ένα αντικείμενο ValidationReport με κατάλληλο μήνυμα λάθους και κωδικό λάθους AlreadyExists.

Επιτυχής Επικύρωση:

- Αν και οι δύο έλεγχοι προηγούμενων βημάτων περάσουν, επιστρέφεται ένα αντικείμενο ValidationReport με επιτυχές μήνυμα.

```
public async Task<ValidationReport> ValidateForRegister(RegisterUserDto userdto)
{var report = new ValidationReport();
    var emailPattern = @"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";
    if (!Regex.IsMatch(userdto.Email, emailPattern))
    {
        report.Success = false; report.Message = "Please provide a valid email!";
        report.ErrorCode = ErrorCode.InvalidEmail;
        return report;
    }
    var user = await _repository.Get(userdto.Email);
    if (user is not null)
    {
        report.Message = "email field already exists!";
        report.Success = false;
        report.ErrorCode = ErrorCode.AlreadyExists;
        return report;
    }
    report.Message = "User with this email can be created";
    report.Success = true;
    return report;
}
```

Κώδικας 4.30: Μέθοδος Επικύρωσης Εγγραφής ValidateForRegister.

4.11.1.2 Μέθοδος ValidateForLogin

Η μέθοδος **ValidateForLogin** εκτελεί διάφορους ελέγχους για την επικύρωση των στοιχείων που χρησιμοποιούνται κατά την σύνδεση ενός χρήστη. Αναλύονται τα βήματα που περιέχει:

Ανάκτηση Χρήστη με Συμπερίληψη Αρμοδιοτήτων:

- Χρησιμοποιεί το IUserRepository για να αναζητήσει έναν χρήστη βάσει του δοσμένου email.

- Η μέθοδος GetUserIncludingAuthorities περιλαμβάνει επίσης τις αρμοδιότητες (roles) του χρήστη.
- Αν ο χρήστης δεν βρεθεί, επιστρέφει ένα αντικείμενο ValidationReport με κατάλληλο μήνυμα λάθους και κωδικό λάθους NotFound, μαζί με null για τον χρήστη.

Έλεγχος Συνδυασμού Email/Κωδικού:

- Χρησιμοποιεί τον αλγόριθμο BCrypt για να ελέγξει αν ο κωδικός πρόσβασης που παρέχεται στο loginUserDto ταιριάζει με τον αποθηκευμένο κωδικό πρόσβασης του χρήστη.
- Αν ο κωδικός πρόσβασης δεν είναι σωστός, επιστρέφεται ένα αντικείμενο ValidationReport με κατάλληλο μήνυμα λάθους και κωδικό λάθους NotFound, μαζί με null για τον χρήστη.

Έλεγχος 2FA:

- Αν ο έλεγχος του συνδυασμού email/κωδικού είναι επιτυχής και ο χρήστης έχει ενεργοποιημένο το 2FA, επιστρέφεται ένα αντικείμενο **ValidationReport** με μήνυμα "2FA is already enabled, check your email if you attempted to login", επιτυχές status και κωδικό λάθους **2FaEnabled**.

Επιτυχής Επικύρωση:

- Αν όλοι οι έλεγχοι προηγούμενων βημάτων περάσουν, επιστρέφεται ένα αντικείμενο ValidationReport με επιτυχές μήνυμα και τα στοιχεία του χρήστη.

Ακολουθεί ο κώδικας:

```
public async Task<(ValidationReport report, User? user)> ValidateForLogin(LoginUserDto
loginUserDto)
{
    var report = new ValidationReport();
    var user = await _repository.GetUserIncludingAuthorities(loginUserDto.Email);

    if (user is null)
    {
        report.Message = "User not found";
        report.Success = false;
        report.ErrorCode = ErrorCode.NotFound;
        return (report, null);
    }

    if (!_userService.VerifyPassword(user.Password!, loginUserDto.Password))
    {
        report.Message = "User with this combination not found";
        report.Success = false;
        report.ErrorCode = ErrorCode.NotFound;
        return (report, null);
    }

    if (user. 2FA enabled)
    {
        report.Message = "2FA is already enabled, check your email if you attempted to
login.";
        report.Success = true;
        report.ErrorCode = ErrorCode._2FaEnabled;
        return (report, user);
    }

    report.Message = "User Verified";
    report.Success = true;
    return (report, user);

}
```

Κώδικας 4.31: Μέθοδος Επικύρωσης Στοιχείων για σύνδεση ValidateForLogin.

Αυτές και άλλες πολλές μέθοδοι και κλάσεις υπάρχουν, για την σωστή διαχείριση των δεδομένων στην εφαρμογή.

4.12 BCrypt

Ως αλγόριθμος κατακερματισμού του κωδικού πρόσβασης του χρήστη χρησιμοποιείται το BCrypt.

το BCrypt είναι μια κρυπτογραφική συνάρτηση κατακερματισμού (hash function) που συχνά χρησιμοποιείται για την ασφαλή αποθήκευση κωδικών πρόσβασης. Είναι ένας αλγόριθμος κατακερματισμού που σχεδιάστηκε να είναι αργός και ανθεκτικός στις επιθέσεις κατά της κατάλληλης επεξεργαστικής ισχύος (brute-force attacks).

Ο βασικός σκοπός του BCrypt είναι να εξασφαλίσει ότι, ακόμη και αν οι κακόβουλοι εξαναγκαστούν να κλέψουν τους κατακερματισμένους κωδικούς πρόσβασης, θα τους είναι δύσκολο να αποκαλύψουν τον αρχικό κωδικό πρόσβασης.

Το BCrypt συνήθως χρησιμοποιείται σε συνδυασμό με έναν αλγόριθμο κατακερματισμού (όπως SHA-256) για περαιτέρω ενίσχυση της ασφάλειας.

4.13 Service Classes

Το service layer επίσης παρέχει τυποποιημένες κλάσεις με μεθόδους, που χρησιμεύουν στην αλλαγή μοντέλων σε DTOs και αντίστροφα. Πιο σύνηθες, χρησιμοποιούνται από τους Controller του API, μετά την επικύρωση των δεδομένων από τις κλάσεις επικύρωσης δεδομένων που περιγράφηκαν προηγουμένως.

Έχουν πρόσβαση στις κλάσεις των Repositories και χρησιμοποιούνται για εισαγωγή, μεταβολή ή διαγραφή.

Λόγο του μεγάλου αριθμού κλάσεων, περιγράφονται μόνο επιλεγμένες κλάσεις και μέθοδοι.

4.13.1 UserService

Η κλάση αυτή χρησιμοποιείται για λειτουργίες που έχουν να κάνουν με τους χρήστες τις εφαρμογής. Περιέχει μία διεπαφή **IUserService** και μία υλοποίηση **UserService**.

Η διεπαφή παρέχει ένα εύρος μεθόδων που υλοποιούνται στην κλάση UserService:

```
public interface IUserService
{
    Task<RegisterUserResponseDto> Register(RegisterUserDto registerUserDto, string
verificationToken);
    JwtDto GenerateToken(User user);
    Task EnableAccount(User user);
    string GenerateOTP(User user);
    Task Save2FaCode(User user, string totpCode);
    Task ActivateTwoFactorAuthentication(User user);
    Task DeactivateTwoFactorAuthentication(User user);
    ClaimsPrincipal? VerifyToken(string token);
    Task<UserDto> ConstructUserInfo(User user);
    Task UpdateFacebook(User user, string link);
    Task UpdateInstagram(User user, string link);
```

```
    Task UpdateYoutube(User user, string link);
    Task RemoveSocialMedia(User user, SocialMedia socialMedia);
    Task UpdateUsername(UpdateUsernameDto updateUsernameDto);
    Task UpdatePassword(UpdatePasswordDto updatePasswordDto, User user);
    Task<string> SetTemporaryPassword(User user);
    Task UploadPhoto(User user, UpdateUserPhotoDto updateUserPhotoDto);
    Task AddRole(User user, string role);
    Task RemoveRole(User user, string role);
    Task SetProfilePhoto(User user, UserImage userImage, SetProfilePhotoDto
setProfilePhotoDto);
    Task RemoveUserImage(UserImage userImage);
    Task UnsetProfilePhoto(UserImage userImage);
    Task SetBio(User user, string location);
    Task UnsetBio(User user);
    Task UpdateVerifiedPhoneNumber(User user, string number);
    Task RegisterPhoneNumber(User user, string phoneNumber);
    Task RemovePhoneNumber(User user);
}

public class UserService : IUserService
{
    private readonly IUserRepository _repository;
    private readonly ITokenService _tokenService;
    private readonly IAssignedUserRepository _assignedUserRepo;
    private readonly IUserVenueRepository _userVenueRepo;
    private readonly IUserEventRepository _userEventRepo;

    public UserService(IUserRepository repository, ITokenService tokenService,
IAssignedUserRepository assignedUserRepository,
        IUserVenueRepository userVenueRepository, IUserEventRepository userEventRepository)
    {
        _repository = repository;
        _tokenService = tokenService;
        _assignedUserRepo = assignedUserRepository;
        _userVenueRepo = userVenueRepository;
        _userEventRepo = userEventRepository;
    }
```

Κώδικας 4.32: Διεπαφή IUserService και η υλοποίηση της.

4.13.1.1 Register

Αναλύεται η μέθοδος της εγγραφής νέου χρήστη στο σύστημα.

Κατακερματισμός Κωδικού Πρόσβασης:

- Ο κωδικός πρόσβασης που παρέχεται από τον χρήστη κατακερματίζεται χρησιμοποιώντας την κρυπτογραφική συνάρτηση BCrypt.Net.

Δημιουργία Νέου Χρήστη:

- Δημιουργείται ένα νέο αντικείμενο χρήστη (User) με τα παρακάτω στοιχεία:
- Ηλεκτρονική διεύθυνση (Email) που παρέχεται από τον χρήστη.
- Κατακερματισμένος κωδικός πρόσβασης.
- Ενεργοποίηση (Enabled) που ορίζεται σε false.
- Κωδικός επαλήθευσης (VerificationCode) που παρέχεται κατά την εγγραφή.
- Ενεργοποίηση διπλού παράγοντα (2FA_enabled) που ορίζεται σε false.

Έλεγχος και Προεπιλεγμένος Ρόλος:

- Ελέγχει αν ο ρόλος που παρέχεται από τον χρήστη είναι έγκυρος (1 για admin, 2 για user, 3 για developer).

- Αν δεν είναι έγκυρος, ορίζεται προεπιλεγμένα στον ρόλο 2 (user).

Κλήση Μεθόδου Εγγραφής του Χρήστη στο Αποθετήριο:

- Καλεί τη μέθοδο Register του αποθετηρίου (_repository) για να εγγραφεί ο νέος χρήστης στο σύστημα.
- Περνά το αντικείμενο χρήστη και τον κωδικό ρόλο ως παραμέτρους.

Δημιουργία Αντικειμένου Απάντησης:

- Δημιουργεί ένα αντικείμενο απάντησης (RegisterUserResponseDto) που περιλαμβάνει τα ακόλουθα στοιχεία:
- Ταυτότητα χρήστη (Id).
- Ηλεκτρονική διεύθυνση (Email).
- Κατάσταση ενεργοποίησης (Enabled).
- Μήνυμα ειδοποίησης προς τον χρήστη ("Check your email for the verification link to enable your account!").

Επιστροφή του Αντικειμένου Απάντησης:

- Επιστρέφεται το αντικείμενο απάντησης ως απόκριση της μεθόδου.

Η μέθοδος αυτή χειρίζεται την εγγραφή ενός νέου χρήστη, καλώντας τις αντίστοιχες υπηρεσίες και αποθηκεύοντας τα αποτελέσματα σε ένα αντικείμενο απάντησης που στέλνεται πίσω στον χρήστη.

Ακολουθεί ο κώδικας της λογικής:

```
public async Task<RegisterUserResponseDto> Register(RegisterUserDto registerUserDto, string
verificationToken)
{
    string hashedPassword = BCrypt.Net.BCrypt.HashPassword(registerUserDto.Password);

    User user = new User
    {
        Email = registerUserDto.Email,
        Password = hashedPassword,
        Enabled = false,
        VerificationCode = verificationToken,
         2FA enabled = false
    };

    if (! (registerUserDto.Role.Equals(1) || registerUserDto.Role.Equals(2) ||
registerUserDto.Role.Equals(3)) || registerUserDto.Role is null)
    {
        registerUserDto.Role = 2;
    }

    var userCreated = await _repository.Register(user, (int)registerUserDto.Role);
    var userDtoRole = new RegisterUserResponseDto
    {
        Id = userCreated.Id,
        Email = userCreated.Email,
        Enabled = (bool)userCreated.Enabled!,
        Note = "Check your email for the verification link to enable your account!"
    };
    return userDtoRole;
}
```

Κώδικας 4.33: Μέθοδος Register του UserService.cs

4.13.1.2 GenerateToken

Η μέθοδος αυτή δημιουργεί ένα JWT με βάση τις πληροφορίες του χρήστη. Τα στοιχεία που περιλαμβάνονται στο JWT είναι το email του χρήστη και ο ρόλος του. Επίσης, το JWT υπογράφεται με ένα μυστικό κλειδί (HMAC-SHA256), προσδίδοντας έτσι αυθεντικότητα και ακεραιότητα. Τέλος, το JWT έχει μια χρονική διάρκεια ισχύος (Expires), και τα απαραίτητα πεδία για την ενσωμάτωσή του στο αίτημα του πελάτη (π.χ., access_token, token_type, expires_in). Ο κώδικας της οποίας αναλύθηκε περισσότερο στο Security JWT.

Η μέθοδος αυτή χρησιμοποιείται από τους controllers /user/login, /login2Fa, /refreshToken

```
public JwtDto GenerateToken(User user)
{
    var jwtDto = _tokenService.GenerateToken(user);

    return jwtDto;
}
```

Κώδικας 4.34: Μέθοδος GenerateToken του UserService.cs

4.13.1.3 ConstructUserInfo

Η μέθοδος ConstructUserInfo να κατασκευάζει ένα αντικείμενο UserDto με βάση τα στοιχεία του χρήστη (User) και άλλες συναφείς πληροφορίες. Χρησιμοποιείται από το endpoint /api/user/info για ανάκτηση όλων των πληροφοριών του χρήστη. Αναλύονται τα βήματα της μεθόδου:

1. Λαμβάνει τον ρόλο του χρήστη (userIntRole) και μετατρέπει τον αριθμητικό κωδικό του σε αντίστοιχο όνομα ρόλου (π.χ., 1 για admin, 2 για user κλπ.).
2. Λαμβάνει τις εικόνες του χρήστη μέσω της μεθόδου GetUserImages από το αποθετήριο.
3. Δημιουργεί αντικείμενο UserProfilePhotoDto για τη φωτογραφία προφίλ του χρήστη.
4. Λαμβάνει το πρόσωπο που έχει δηλωθεί από το χρήστη ως "claimed" (δηλαδή ότι του ανήκει) μέσω της μεθόδου GetClaimedPersonForUser
5. Λαμβάνει τις χώρες και τα γεγονότα που έχει δηλωθεί ως "claimed" από το χρήστη, χρησιμοποιώντας τις μεθόδους GetClaimedVenuesForUser και GetClaimedEventsForUser.
6. Δημιουργεί ένα αντικείμενο UserDto με βάση τις παραπάνω πληροφορίες και τα αντικείμενα που ανακτώνται από τις βοηθητικές μεθόδους.

```
public async Task<UserDto> ConstructUserInfo(User user)
{

    var userIntRole = user.UserAuthorities.FirstOrDefault()?.AuthorityId;
    var userImages = await _repository.GetUserImages(user.Id);
    var userImagesDto = CreateUserImagesDto(userImages);
    var userProfilePhoto = GetUserProfilePhotoFromImagesList(userImages);
    var userProfilePictureResponseDto = CreateUserProfilePhotoDto(userProfilePhoto);
    var claimedPerson = await GetClaimedPersonForUser(user);
    var claimedVenues = await GetClaimedVenuesForUser(user);
    var claimedEvents = await GetClaimedEventsForUser(user);

    var userRole = userIntRole switch
    {
        1 => "admin",
        2 => "user",
        3 => "developer",
        4 => "claims manager",
        _ => "Invalid role"
    };

    var userDto = new UserDto
    {
        Id = user.Id,
        Username = user.Username,
```

```
        Email = user.Email,
        EmailVerified = user.Enabled,
        _2FA_enabled = user._2FA_enabled,
        Role = userRole,
        Transactions = user.UserTransactions.Select(t => new TransactionDtoFetch
        {
            UserId = t.UserId,
            CreditAmount = t.CreditAmount,
            Reason = t.Reason,
            DateCreated = t.DateCreated
        }).ToList(),
        Facebook = user.Facebook,
        Youtube = user.Youtube,
        Instagram = user.Instagram,
        Balance = user.UserTransactions.Sum(t => t.CreditAmount),
        UserImages = userImagesDto,
        ProfilePhoto = userProfilePictureResponseDto,
        PerformerRoles = user.PerformerRoles,
        BioPdfLocation = user.BioPdfLocation,
        ClaimedPerson = claimedPerson,
        ClaimedVenues = claimedVenues,
        ClaimedEvents = claimedEvents,
        PhoneNumber = user.PhoneNumber,
        PhoneVerified = user.PhoneNumberVerified,
    };

    return userDto;
}//Χρησιμοποιείται από το endpoint /api/user/info για όλες τις πληροφορίες του χρήστη.
```

Κώδικας 4.35: Μέθοδος ConstructUserInfo του UserService.cs

4.14 Επίλογος

Συνοπτικά, το service layer της εφαρμογής παρουσιάζει ένα σύνολο υπηρεσιών που διαχειρίζονται τη λογική επιχείρησης, την αλληλεπίδραση με τη βάση δεδομένων, και την ασφαλή επικοινωνία με τα API της εφαρμογής. Μέσα από τον κώδικα, παρουσιάσαμε διάφορα μέρη του service layer που καλύπτουν διάφορες λειτουργίες.

Στην ενότητα αυτή αναλύθηκε ο τρόπος με τον οποίο διαχειριζόμαστε την επικοινωνία με το Twilio για την επιβεβαίωση των αριθμών τηλεφώνου, προσφέροντας έναν αξιόπιστο μηχανισμό επαλήθευσης ταυτότητας. Εξετάστηκαν επίσης οι πτυχές της ασφάλειας μέσω του χρήση των JWT για τον έλεγχο αυθεντικότητας, καθώς και τα φίλτρα εξουσιοδότησης που ελέγχουν την πρόσβαση στα endpoints.

Οι κλάσεις του Repository Pattern επιτρέπουν την αποτελεσματική διαχείριση των δεδομένων στη βάση δεδομένων, με στόχο τη διατήρηση της διάκρισης μεταξύ της λογικής της εφαρμογής και των λεπτομερειών της αποθήκευσης δεδομένων.

Τέλος, οι υπηρεσίες επαληθεύσεων (validation) προσφέρουν ένα σύνολο καλά καθορισμένων κανόνων και ελέγχων για την εξασφάλιση της ορθής λειτουργίας της εφαρμογής και της ασφάλειάς της.

Κεφάλαιο 5ο: Api Layer

5.1 Εισαγωγή

Στο τελευταίο κεφάλαιο, η εστίαση στρέφεται προς το Api Layer, το οποίο αποτελεί τη διεπαφή μέσω της οποίας οι χρήστες αλληλεπιδρούν με την εφαρμογή. Αυτή η ενότητα καλύπτει τους controllers του API, το configuration startup της εφαρμογής σε περιβάλλον .NET, καθώς και λεπτομέρειες όπως η διαμόρφωση του Swagger, η χρήση του Dependency Injection, και η ενσωμάτωση των Razor Pages.

Οι controllers αποτελούν το κύριο μέρος του Api Layer, χειρίζονται τα αιτήματα των χρηστών και ανακατευθύνουν τις κλήσεις προς τις κατάλληλες υπηρεσίες του Service Layer. Το configuration startup της εφαρμογής παρέχει τις βασικές ρυθμίσεις και επιτρέπει τη σωστή λειτουργία του περιβάλλοντος εκτέλεσης της εφαρμογής.

Επιπλέον, το Swagger, ως εργαλείο διαχείρισης και τεκμηρίωσης του API, έχει ρυθμιστεί έτσι ώστε να παρέχει ένα φιλικό και ενημερωτικό περιβάλλον προς το χρήστη. Το Dependency Injection ενσωματώνεται για την αποτελεσματική διαχείριση των κλάσεων της εφαρμογής και την επιτάχυνση του κώδικα.

Τέλος, οι σελίδες Razor προσφέρουν έναν απλό και επεκτάσιμο τρόπο για τη δημιουργία δυναμικών ιστοσελίδων μέσω της χρήσης μιας σχεδιαστικής σύνθεσης. Αυτό δίνει τη δυνατότητα στην εφαρμογή να παρέχει διάφορες σελίδες χρηστών με βάση τις ανάγκες και τις λειτουργικότητες.

Συνδυάζοντας όλους τους παραπάνω μηχανισμούς, το επίπεδο API γίνεται μια ολοκληρωμένη και αποτελεσματική διεπαφή μεταξύ της εφαρμογής και των χρηστών, εξασφαλίζοντας τη σωστή λειτουργικότητα και την απρόσκοπτη επικοινωνία μεταξύ των διαφόρων επιπέδων.

5.2 Διαμόρφωση κλάσης εκκίνησης

Στην κλάση Startup (Program.cs), η παραμετροποίηση ξεκινά με τη δημιουργία ενός κατασκευαστή εφαρμογών ιστού (WebApplication.CreateBuilder(args)). Αυτός ο κατασκευαστής χρησιμοποιείται για τη ρύθμιση και διαμόρφωση διαφόρων πτυχών της εφαρμογής.

5.2.1 Authentication Configuration

Η εφαρμογή χρησιμοποιεί αυθεντικοποίηση JWT (JSON Web Token). Οι ρυθμίσεις JWT ανακτώνται από το αρχείο ρυθμίσεων, επίσης προστίθεται το σχήμα ελέγχου ταυτότητας, μαζί με τον έλεγχο ταυτότητας JWT bearer. Καθορίζονται οι παράμετροι για την επικύρωση του token, συμπεριλαμβανομένου του εκδότη, του ακροατηρίου και του κλειδιού υπογραφής.

5.2.2 Authorization Configuration

Οι υπηρεσίες εξουσιοδότησης προστίθενται στο container, προετοιμάζοντας την εφαρμογή για έλεγχο πρόσβασης βάσει ρόλων.

5.2.3 Controllers Configuration

Οι ρυθμίσεις JSON serialization για τους ελεγκτές έχουν ρυθμιστεί έτσι ώστε να χειρίζονται με αξιοπρέπεια τους κύκλους αναφοράς και τις μηδενικές τιμές.

5.2.4 Διαμόρφωση Swagger

Το Swagger ενσωματώνεται στην εφαρμογή για τη δημιουργία documentation API. Διαμορφώνεται το περιβάλλον εργασίας Swagger, καθορίζοντας τον τίτλο και την έκδοση του API.

5.2.5 Dependency Injection

Η εφαρμογή χρησιμοποιεί κατά κόρον το dependency injection. Κάθε κλάση σε κάθε επίπεδο της εφαρμογής χρησιμοποιεί dependency injection, με τα service classes που χρειάζονται να χρησιμοποιήσουν. Διάφορες υπηρεσίες καταχωρούνται με dependency injection. Σε αυτές τις υπηρεσίες περιλαμβάνονται αποθετήρια και υπηρεσίες επικύρωσης για διάφορες οντότητες, όπως πρόσωπα, ρόλοι, διοργανωτές, χώροι, χρήστες, παραγωγές, εκδηλώσεις, συνεισφορές, αρχεία καταγραφής, συναλλαγές, υπηρεσία JWT token, υπηρεσία σελιδοποίησης, υπηρεσίες επιμελητών, υπηρεσία ηλεκτρονικού ταχυδρομείου, υπηρεσία προσωρινής αποθήκευσης και άλλα.

5.2.6 Σύνδεση Βάσης Δεδομένων

Η σύνδεση με τη βάση δεδομένων διαμορφώνεται χρησιμοποιώντας τον πάροχο PostgreSQL.

5.2.7 Διαμόρφωση CORS:

CORS (Cross-Origin Resource Sharing) έχει ρυθμιστεί ώστε να επιτρέπει οποιαδήποτε προέλευση, μέθοδο και επικεφαλίδα.

5.2.8 Διαμόρφωση καταγραφής

Το Serilog έχει ρυθμιστεί για καταγραφή, επιτρέποντας λεπτομερή και δομημένη έξοδο καταγραφής. Η μορφοποίηση του αρχείου καταγραφής της κονσόλας έχει ρυθμιστεί για μια καθαρή παρουσίαση.

5.2.9 Διαμόρφωση σελίδων Razor

Οι σελίδες Razor προστίθενται στις υπηρεσίες, υποδεικνύοντας ένα δυναμικό στοιχείο UI, ειδικά για τα τελικά σημεία επαλήθευσης ηλεκτρονικού ταχυδρομείου.

5.2.10 Κατασκευή και εκτέλεση εφαρμογής

Τέλος, κατασκευάζεται η εφαρμογή και ρυθμίζεται το pipeline αιτήσεων HTTP. Αυτό περιλαμβάνει τη ρύθμιση για το Swagger, την ανακατεύθυνση HTTPS, τη δρομολόγηση, το χειρισμό CORS, τον έλεγχο ταυτότητας και την εξουσιοδότηση. Οι ελεγκτές αντιστοιχίζονται και η εφαρμογή εκτελείται.

5.3 Πακέτα που χρησιμοποιούνται στην εφαρμογή

Η εφαρμογή χρησιμοποιεί διάφορα πακέτα για την ενίσχυση της λειτουργικότητάς της και την παροχή βασικών χαρακτηριστικών. Ακολουθεί κατάλογος των πακέτων που χρησιμοποιούνται:

5.3.1 MailKit:

Παρέχει υποστήριξη για την αποστολή και τη λήψη μηνυμάτων ηλεκτρονικού ταχυδρομείου.

5.3.2 Microsoft.AspNetCore.Authentication.JwtBearer

Ενεργοποιεί τον έλεγχο ταυτότητας JWT για την εξασφάλιση των τελικών σημείων της εφαρμογής.

5.3.3 Microsoft.AspNetCode.Mvc.NewtonsoftJson

Βελτιώνει τη σειριοποίηση JSON στο ASP.NET Core MVC χρησιμοποιώντας το Newtonsoft.Json.

5.3.4 Microsoft.EntityFrameworkCore.Design

Παρέχει υποστήριξη σε επίπεδο σχεδιασμού για το Entity Framework Core, αποσκοπεί κυρίως σε εργασίες όπως η παραγωγή κώδικα για μεταναστεύσεις βάσεων δεδομένων, για δημιουργία μοντέλων από μια υπάρχουσα βάση δεδομένων, κ.λπ., οι οποίες γίνονται συνήθως κατά τη φάση της ανάπτυξης.

5.3.5 Minio

Ένας client .NET για το MinIO, που επιτρέπει την αλληλεπίδραση με τις υπηρεσίες αποθήκευσης MinIO.

5.3.6 Npgsql.EntityFrameworkCore.PostgreSQL

Ο πάροχος PostgreSQL για το Entity Framework Core, διευκολύνοντας τις λειτουργίες βάσης δεδομένων με την PostgreSQL.

5.3.7 Serilog

Μια ισχυρή βιβλιοθήκη καταγραφής για εφαρμογές .NET, που προσφέρει δομημένες και ευέλικτες δυνατότητες καταγραφής.

5.3.8 Serilog.AspNetCore

Επεκτείνει τη Serilog για την απρόσκοπτη ενσωμάτωση με εφαρμογές ASP.NET Core.

5.3.9 Serilog.Sinks.Console

Λειτουργία καταγραφής γράφει συμβάντων στην κονσόλα με προσαρμόσιμο στυλ.

5.3.10 Stripe.net

Ο .NET πελάτης για το API της Stripe, που χρησιμοποιείται για το χειρισμό λειτουργιών που σχετίζονται με πληρωμές.

5.3.11 Swashbuckle.AspNetCore

Ενσωματώνει το Swagger για τη δημιουργία τεκμηρίωσης API και παρέχει ένα web-based UI για την εξερεύνηση και τη δοκιμή του API.

5.3.12 System.IdentityModel.Tokens.Jwt

Παρέχει υποστήριξη JWT για τη δημιουργία, την ανάλυση και την επικύρωση JWT.

5.3.13 BCrypt.Net

Μια υλοποίηση .NET του αλγορίθμου κατακερματισμού κωδικών πρόσβασης bcrypt για ασφαλή αποθήκευση κωδικών πρόσβασης.

5.3.14 Otp.NET

Μια βιβλιοθήκη για τη δημιουργία και επικύρωση OTPs (One-Time Passwords) για έλεγχο ταυτότητας δύο παραγόντων.

5.3.15 Twilio

Ο πελάτης .NET για το API της Twilio, που χρησιμοποιείται για την αποστολή μηνυμάτων SMS και το χειρισμό υπηρεσιών επικοινωνίας.

Αυτά τα πακέτα συμβάλλουν συλλογικά στην ασφάλεια, τη λειτουργικότητα και την αποδοτικότητα της εφαρμογής, καλύπτοντας τομείς όπως ο έλεγχος ταυτότητας, η καταγραφή, η αλληλεπίδραση με βάσεις δεδομένων, ο χειρισμός ηλεκτρονικού ταχυδρομείου και η ενσωμάτωση εξωτερικών υπηρεσιών.

5.4 Controllers

Οι ελεγκτές συμβάλλουν καθοριστικά στη διαμόρφωση της αρχιτεκτονικής και της λειτουργικότητας των εφαρμογών ιστού. Ένας ελεγκτής χρησιμεύει ως μεσολαβητής μεταξύ της διεπαφής χρήστη και της υποκείμενης λογικής της εφαρμογής, συντονίζοντας τη ροή δεδομένων και απαντήσεων. Ακολουθεί μια εισαγωγή στους ελεγκτές στο .NET, επισημαίνοντας τη σημασία και τη χρησιμότητά τους.

5.4.1 Σκοπός των Ελεγκτών

1. Χειρισμός αιτημάτων χρηστών: Οι ελεγκτές είναι ειδικά σχεδιασμένοι για να χειρίζονται εισερχόμενα αιτήματα HTTP από πελάτες, συνήθως προγράμματα περιήγησης στο διαδίκτυο ή εφαρμογές για κινητά τηλέφωνα. Ενεργούν ως σημείο εισόδου για αυτά τα αιτήματα, κατευθύνοντάς τα στις κατάλληλες μεθόδους με βάση το ζητούμενο URI (Uniform Resource Identifier) και το λεκτικό HTTP (GET, POST κ.λπ.).

2. Επεξεργασία επιχειρησιακής λογικής: Οι ελεγκτές είναι υπεύθυνοι για την εκτέλεση της επιχειρησιακής λογικής της εφαρμογής. Ερμηνεύουν την είσοδο του χρήστη, αλληλεπιδρούν με τα μοντέλα και τις κλάσεις των άλλων επιπέδων της εφαρμογής για την ανάκτηση ή τον χειρισμό δεδομένων και ενορχηστρώνουν τη συνολική ροή εργασιών της εφαρμογής.

3. Διαχωρισμός των προβλημάτων: Οι ελεγκτές συμβάλλουν στην αρχή του διαχωρισμού των προβλημάτων στην ανάπτυξη λογισμικού. Με το διαχωρισμό της εφαρμογής σε διακριτά στοιχεία (μοντέλα, προβολές, ελεγκτές), γίνεται πιο αρθρωτή και ευκολότερη η συντήρησή της. Οι ελεγκτές εστιάζουν στην είσοδο του χρήστη και στη ροή της εφαρμογής, εξασφαλίζοντας μια καθαρή και οργανωμένη βάση κώδικα.

4. Παραγωγή απαντήσεων: Μετά την επεξεργασία, οι ελεγκτές παράγουν τις κατάλληλες απαντήσεις HTTP. Οι απαντήσεις μπορεί να περιλαμβάνουν περιεχόμενο HTML, δεδομένα JSON ή άλλες μορφές ανάλογα με τη φύση του αιτήματος.

5.4.2 Βασικά συστατικά και χαρακτηριστικά

1. Αρχιτεκτονική MVC: Οι ελεγκτές αποτελούν ένα αναπόσπαστο μέρος του αρχιτεκτονικού προτύπου Model-View-Controller (MVC), το οποίο προάγει το διαχωρισμό των προβλημάτων. Οι ελεγκτές χειρίζονται τη λογική της εφαρμογής (Controller), οι προβολές διαχειρίζονται την παρουσίαση και τα μοντέλα αναπαριστούν τα δεδομένα.

2. Δρομολόγηση και αντιστοίχιση URL: Οι ASP.NET controllers μέθοδοι συνδέονται με συγκεκριμένες διαδρομές, επιτρέποντας έτσι στο πλαίσιο να αντιστοιχίζει τα εισερχόμενα αιτήματα στις αντίστοιχες ενέργειες του ελεγκτή. Αυτό επιτρέπει καθαρά και δομημένα μοτίβα URL.

3. Ανάπτυξη RESTful API: Στην ανάπτυξη API, οι ελεγκτές παίζουν σημαντικό ρόλο στην υλοποίηση υπηρεσιών RESTful. Ορίζουν τελικά σημεία και ενέργειες που ανταποκρίνονται σε διάφορες μεθόδους HTTP, διευκολύνοντας την επικοινωνία μεταξύ πελατών και διακομιστών.

4. Dependency Injection: Οι ελεγκτές στο ASP.NET Core επωφελούνται από το σύστημα έγχυσης εξαρτήσεων, επιτρέποντας στους προγραμματιστές να εισάγουν υπηρεσίες, αποθετήρια ή άλλες εξαρτήσεις απευθείας στους ελεγκτές. Αυτό προάγει την ευελιξία και τη δυνατότητα δοκιμών.

5. Δρομολόγηση βάσει χαρακτηριστικών: Οι ελεγκτές μπορούν να αξιοποιήσουν τη δρομολόγηση βάσει χαρακτηριστικών, επιτρέποντας στους προγραμματιστές να ορίζουν διαδρομές απευθείας στις ενέργειες (μεθόδους) του ελεγκτή. Αυτό προσφέρει έναν πιο δηλωτικό και εύχρηστο τρόπο δόμησης των διαδρομών.

6. Ενσωμάτωση Middleware: Οι ελεγκτές ενσωματώνονται απρόσκοπτα με τη διασωλήνωση ενδιάμεσου λογισμικού (middleware pipeline) στο ASP.NET Core, επιτρέποντας τη συμπερίληψη πρόσθετων βημάτων επεξεργασίας, όπως ο έλεγχος ταυτότητας, η εξουσιοδότηση και ο χειρισμός εξαιρέσεων.

7. Unit Testing: Ο διαχωρισμός των προβλημάτων στην αρχιτεκτονική MVC καθιστά τους ελεγκτές εξαιρετικά ανθεκτικούς σε δοκιμές. Οι προγραμματιστές μπορούν να γράψουν δοκιμές μονάδας για να διασφαλίσουν ότι οι ενέργειες του ελεγκτή συμπεριφέρονται όπως αναμένεται σε διαφορετικά σενάρια.

Συνοψίζοντας, οι ελεγκτές στο .NET αποτελούν τα βασικά στοιχεία που διευκολύνουν την οργάνωση, τη δρομολόγηση και την εκτέλεση της λογικής της εφαρμογής σε απάντηση σε εισερχόμενα αιτήματα HTTP. Συμβάλλουν στη συντηρησιμότητα, την επεκτασιμότητα και την ευελιξία των εφαρμογών ιστού και των APIs που είναι χτισμένα στο πλαίσιο ASP.NET.

5.4.3 UserController

Σε αυτή την ενότητα, εμβαθύνονται οι ιδιαιτερότητες του UserController μέσα στην εφαρμογή. Το UserController είναι ένα κεντρικό στοιχείο που διαχειρίζεται τις αλληλεπιδράσεις που σχετίζονται με τους χρήστες, από την εγγραφή των χρηστών μέχρι τη διαχείριση των διαδικασιών ελέγχου ταυτότητας. Διαδραματίζει κρίσιμο ρόλο στην οργάνωση της απρόσκοπτης ροής πληροφοριών μεταξύ της διεπαφής χρήστη και των υποκείμενων υπηρεσιών της backend εφαρμογής.

Τελικό σημείο εγγραφής χρήστη:

Το τελικό σημείο POST `/api/User/register` χρησιμεύει ως ο κύριος μηχανισμός για την εγγραφή των χρηστών στην εφαρμογή. Αυτό το τελικό σημείο δέχεται ως είσοδο ένα αντικείμενο RegisterUserDto, το οποίο περιλαμβάνει βασικές λεπτομέρειες όπως το email, τον κωδικό πρόσβασης και έναν προαιρετικό ρόλο. Το τελικό σημείο επιστρέφει ένα RegisterUserResponseDto που περιέχει κρίσιμες πληροφορίες σχετικά με τη διαδικασία εγγραφής, όπως το αναγνωριστικό του χρήστη, το email, την κατάσταση επαλήθευσης και πρόσθετες σημειώσεις.. Οι χρήστες μπορούν να εγγραφούν με διαφορετικούς ρόλους, όπως διαχειριστής, χρήστης ή προγραμματιστής, με βάση την καθορισμένη παράμετρο ρόλου. Εάν δεν έχει οριστεί κανένας ρόλος, δημιουργείται ο προεπιλεγμένος λογαριασμός χρήστη.

Token επαλήθευσης και επιβεβαίωση ηλεκτρονικού ταχυδρομείου:

Μετά την επιτυχή επιβεβαίωση των στοιχείων εγγραφής, ο UserController παράγει ένα μοναδικό διακριτικό επαλήθευσης και στέλνει ένα email επιβεβαίωσης στον εγγεγραμμένο χρήστη. Το μήνυμα ηλεκτρονικού ταχυδρομείου περιέχει έναν σύνδεσμο επαλήθευσης, ο οποίος κατευθύνει τον χρήστη σε

ένα καθορισμένο τελικό σημείο για να επιβεβαιώσει τη διεύθυνση ηλεκτρονικού ταχυδρομείου του. Αυτή η διαδικασία διασφαλίζει την ασφάλεια και την εγκυρότητα των χρηστών.

Τελικό σημείο σύνδεσης χρήστη (POST /api/user/login):

Το τελικό σημείο POST /api/user/login χρησιμεύει ως πύλη για τον έλεγχο ταυτότητας των χρηστών εντός της εφαρμογής. Δέχεται ένα αντικείμενο LoginUserDto, το οποίο περιέχει το email και τον κωδικό πρόσβασης του χρήστη. Η διαδικασία ελέγχου ταυτότητας πραγματοποιείται με ασφάλεια και μετά την επιτυχή επικύρωση, το τελικό σημείο επιστρέφει ένα JSON Web Token (JWT). Αυτό το JWT χρησιμεύει ως ένα ασφαλές και χρονικά περιορισμένο διακριτικό ελέγχου ταυτότητας, το οποίο παρέχει στους χρήστες πρόσβαση σε προστατευμένους πόρους εντός της εφαρμογής.

Τελικό σημείο ανάκτησης πληροφοριών (GET /api/user/info):

Το τελικό σημείο GET /api/user/info επιτρέπει στους χρήστες να ανακτούν ένα ολοκληρωμένο σύνολο πληροφοριών που σχετίζονται με τη συνδεδεμένη συνεδρία τους. Για να έχουν πρόσβαση σε αυτό το τελικό σημείο, οι χρήστες πρέπει να συμπεριλάβουν ένα έγκυρο JWT στην επικεφαλίδα αίτησης, διασφαλίζοντας ότι μόνο οι πιστοποιημένοι χρήστες μπορούν να έχουν πρόσβαση στις εξατομικευμένες πληροφορίες τους.

Χειρισμός σφαλμάτων και απαντήσεις διακομιστή:

Ο UserController είναι εξοπλισμένος με ισχυρούς μηχανισμούς χειρισμού σφαλμάτων για την αντιμετώπιση διαφόρων σεναρίων. Σε περίπτωση σφαλμάτων επικύρωσης, ο ελεγκτής κατασκευάζει ενημερωτικές απαντήσεις σφάλματος, προσδιορίζοντας τη φύση του σφάλματος και τους σχετικούς κωδικούς σφάλματος. Επιπλέον, τα απροσδόκητα σφάλματα του διακομιστή προκαλούν κατάλληλες απαντήσεις για την αποφυγή περιττής διακοπής της εμπειρίας του χρήστη.

Ενσωμάτωση με εξωτερικές υπηρεσίες:

Για να εμπλουτιστεί η εμπειρία του χρήστη, ο UserController ενσωματώνεται πλήρως με διάφορες εξωτερικές υπηρεσίες. Η υπηρεσία IEmailService χρησιμοποιείται για την αποστολή μηνυμάτων ηλεκτρονικού ταχυδρομείου επιβεβαίωσης, εξασφαλίζοντας μια απλοποιημένη διαδικασία επαλήθευσης. Επιπλέον, το IMinioService χρησιμοποιείται για το χειρισμό των αλληλεπιδράσεων με το Minio, επιτρέποντας την αποτελεσματική διαχείριση των αρχείων και των περιουσιακών στοιχείων (εγγράφων) που σχετίζονται με το χρήστη.

Ενσωμάτωση Twilio:

Ο UserController συνεργάζεται με την υπηρεσία ITwilioService για την επιβεβαίωση τηλεφωνικού αριθμού του χρήστη. Της οποίας η αρμοδιότητα είναι να επικοινωνεί με το API του Twilio και να επιστρέφει κατάλληλη απάντηση για το API επίπεδο.

Περιληπτικά αναφέρονται οι μέθοδοι:

Εγγραφή και επαλήθευση:

- *POST* `/api/user/register`: Οι χρήστες μπορούν να εγγραφούν στην εφαρμογή, λαμβάνοντας ένα διακριτικό επαλήθευσης μέσω ηλεκτρονικού ταχυδρομείου για επιβεβαίωση μέσω ηλεκτρονικού ταχυδρομείου.
- *GET* `/api/user/verify-email`: Επαληθεύει το ηλεκτρονικό ταχυδρομείο ενός χρήστη χρησιμοποιώντας το παρεχόμενο διακριτικό GUID.

Σύνδεση και πιστοποίηση:

- *POST* `/api/user/login`: Παρέχει τη δυνατότητα στους χρήστες να συνδεθούν με ασφάλεια, αποκτώντας ένα JWT για τις επόμενες αυθεντικοποιημένες αιτήσεις.
- *POST* `/api/user/login/2fa/{code}`: Για χρήστες με έλεγχο ταυτότητας 2 παραγόντων, διευκολύνει την είσοδο μετά τη λήψη ενός κωδικού μιας χρήσης.

Διαχείριση προφίλ χρήστη:

- *GET* `/api/user/info`: Ανακτά ολοκληρωμένες πληροφορίες χρήστη χρησιμοποιώντας ένα έγκυρο JWT.
- *POST* `/api/user/update/username/{username}`: Επιτρέπει στους συνδεδεμένους χρήστες να ενημερώσουν το όνομα χρήστη τους.
- *PUT* `/api/User/Update/Password`: Επιτρέπει στους συνδεδεμένους χρήστες να ενημερώσουν τον κωδικό πρόσβασής τους.

Διαχείριση αριθμού τηλεφώνου:

- *POST* `/api/user/register/phoneNumber`: Προσθέτει τον αριθμό τηλεφώνου ενός χρήστη στη βάση δεδομένων.
- *DELETE* `/api/user/remove/phoneNumber`: Διαγράφει τον αριθμό τηλεφώνου ενός χρήστη από τη βάση δεδομένων.
- *POST* `/api/user/request-verification-phone-number`: Ζητάει έναν κωδικό επαλήθευσης από έναν εξωτερικό πάροχο (Twilio).
- *POST* `/api/user/confirm-verification-phone-number`: Επιβεβαιώνει τον κωδικό επαλύθευσης.

Αυθεντικοποίηση δύο παραγόντων:

- *POST* `/api/user/enable2fa`: Επιτρέπει στους χρήστες να επιλέξουν επαλήθευση 2 παραγόντων με χρήση email.
- *POST* `/api/user/disable2fa`: Επιτρέπει στους χρήστες να επιλέξουν την απενεργοποίηση της επαλήθευσης 2 παραγόντων.

Σύνδεσμοι στα μέσα κοινωνικής δικτύωσης:

- *PUT* `/api/User/@/facebook`, *PUT* `/api/User/@/youtube`, *PUT* `/api/User/@/Instagram`: Οι χρήστες μπορούν να προσθέσουν τους συνδέσμους τους στα μέσα κοινωνικής δικτύωσης.
- *DELETE* `/api/User/@/facebook`, *DELETE* `/api/User/@/youtube`, *DELETE* `/api/User/@/Instagram`: Οι χρήστες μπορούν να διαγράψουν τους συνδέσμους των κοινωνικών μέσων τους.

Κατάσταση οικονομικών καταστάσεων και συναλλαγών:

- *GET* `/api/user/{id}/balance`: Μόνο για διαχειριστές για την προβολή του υπολοίπου ενός χρήστη.
- *GET* `/api/Transactions/user/{id}`: Μόνο για διαχειριστές. Επιστρέφει όλες τις συναλλαγές ενός συγκεκριμένου χρήστη.

Φωτογραφία προφίλ και εικόνες:

- *POST* `/api/user/UploadPhoto`: Επιτρέπει στους χρήστες να ανεβάσουν μια φωτογραφία με προαιρετική περιγραφή και κατάσταση προφίλ.
- *POST* `/api/user/set/profile-photo`: Ορίζει μια φωτογραφία ως εικόνα προφίλ.
- *PUT* `/api/user/unset/profile-photo/{imageId}`: Ακυρώνει τη φωτογραφία προφίλ ενός χρήστη.
- *DELETE* `/api/user/remove/image/{imageId}`: Αφαιρεί μόνιμα μια εικόνα από τη βάση δεδομένων.

Ρόλοι και βιογραφικό του καλλιτέχνη:

- *POST* `/api/user/Add-artist-role`: Προσθέτει ρόλους καλλιτέχνη που σχετίζονται με τον χρήστη.
- *POST* `/api/user/remove-artist-role`: Αφαιρεί έναν ρόλο καλλιτέχνη.
- *POST* `/api/user/upload/bio`: Ανεβάζει το βιογραφικό ενός χρήστη σε μορφή PDF.
- *POST* `/api/user/unset/bio`: Αφαιρεί το βιογραφικό του χρήστη.

Ανάκτηση κωδικού πρόσβασης:

- *POST* `/api/user/ForgotPassword`: Για χρήστες που έχουν ξεχάσει τον κωδικό πρόσβασής τους, ξεκινά μια επαναφορά κωδικού πρόσβασης.

Ανανέωση JWT:

- *GET* `/api/user/refresh-token`: Χρησιμοποιείται για την ανανέωση ενός JWT πριν αυτό λήξει.

Συμπερασματικά, ο UserController ενεργεί ως πύλη για τις λειτουργίες με γνώμονα τον χρήστη, ενσωματώνοντας τις λειτουργίες της εγγραφής του χρήστη, της επαλήθευσης του ηλεκτρονικού ταχυδρομείου και της απρόσκοπτης αλληλεπίδρασης. Τα σαφώς καθορισμένα τελικά σημεία και οι ενσωματώσεις του συμβάλλουν στη συνολική ευρωστία και απόκριση της εφαρμογής μας, παρέχοντας στους χρήστες μια ασφαλή και αποτελεσματική πλατφόρμα για το θεατρικό τους ταξίδι.

5.4.4 StripeController: Διευκόλυνση απρόσκοπτων συναλλαγών με Stripe Integration

Σε αυτήν την ενότητα, εξετάζονται οι λειτουργίες του StripeController στην εφαρμογή μας, ένα βασικό στοιχείο που επιβλέπει την ενσωμάτωση με την υπηρεσία επεξεργασίας πληρωμών Stripe. Ο ελεγκτής χειρίζεται τη δημιουργία περιόδων πληρωμής και διαχειρίζεται τα συμβάντα webhook που ενεργοποιούνται από το Stripe, εξασφαλίζοντας μια ομαλή και ασφαλή εμπειρία συναλλαγών για τους χρήστες μας.

Δημιουργία συνεδρίας checkout Stripe:

Το τελικό σημείο CreateCheckoutSession, προσβάσιμο μέσω της διαδρομής /api/stripe/create-checkout-session, είναι υπεύθυνο για την έναρξη της δημιουργίας μιας συνεδρίας πληρωμής. Αξιοποιώντας το API της Stripe, κατασκευάζει τις απαραίτητες επιλογές για μια συνεδρία, συμπεριλαμβανομένων των στοιχείων γραμμής, των επιτρεπόμενων κωδικών προώθησης και των URL επιτυχίας και ακύρωσης. Στη συνέχεια, το τελικό σημείο ανακατευθύνει τον χρήστη στη σελίδα πληρωμής που φιλοξενείται από την Stripe, απλοποιώντας τη διαδικασία πληρωμής. Αν ο χρήστης τελειώσει την πληρωμή, ανακατευθύνεται σε μία προσαρμοσμένη σελίδα Razor της εφαρμογής μας.

Χειρισμός συμβάντων Webhook:

Το τελικό σημείο StripeWebhook ανταποκρίνεται σε εισερχόμενα συμβάντα webhook από το Stripe. Αυτά τα συμβάντα ενημερώνουν την εφαρμογή μας σχετικά με την κατάσταση των ολοκληρωμένων περιόδων πληρωμής. Ο controller επικυρώνει την αυθεντικότητα του webhook χρησιμοποιώντας την παρεχόμενη υπογραφή της Stripe και επεξεργάζεται το συμβάν με βάση τον τύπο του.

Γεγονός CheckoutSessionCompleted:

Εάν ο τύπος συμβάντος είναι CheckoutSessionCompleted, ο controller εξάγει σχετικές πληροφορίες από την ολοκληρωμένη συνεδρία, όπως η κατάσταση πληρωμής, το email του πελάτη και οι λεπτομέρειες της συναλλαγής. Στη συνέχεια, επικυρώνει τον χρήστη που συνδέεται με το email και εκτελεί τις απαραίτητες ενέργειες, όπως η δημοσίευση συναλλαγών και η ενημέρωση των πιστώσεων του χρήστη.

Επιτυχής πληρωμή:

Αν η πληρωμή είναι επιτυχής (paymentStatus.Equals("paid")), ο ελεγκτής περνάει την συναλλαγή, προσθέτοντας πιστώσεις στο λογαριασμό του χρήστη. Εξασφαλίζει την ομαλή διασύνδεση μεταξύ της επιβεβαίωσης πληρωμής του Stripe και του συστήματος συναλλαγών της εφαρμογής μας.

Μη επιτυχής πληρωμή:

Εάν η πληρωμή δεν είναι επιτυχής, ο ελεγκτής κοινοποιεί την κατάσταση στο Stripe, εξασφαλίζοντας διαφάνεια σχετικά με την εξέλιξη της συναλλαγής.

Ενσωματώσεις εξωτερικών υπηρεσιών (κλάσεων) της εφαρμογής με Dependency Injection:

Ο StripeController συνεργάζεται με διάφορες εξωτερικές υπηρεσίες για τη βελτίωση της λειτουργικότητάς του. Βασίζεται στην υπηρεσία IUserValidationService για την επικύρωση των πληροφοριών χρήστη που σχετίζονται με το συμβάν Stripe webhook. Επιπλέον, το ITransactionService διευκολύνει την ανάρτηση συναλλαγών με βάση την επιτυχή ολοκλήρωση της πληρωμής.

Εν κατακλείδι, ο StripeController εξυπηρετεί ως γέφυρα μεταξύ της εφαρμογής μας και της υπηρεσίας επεξεργασίας πληρωμών Stripe, προσφέροντας έναν ασφαλή και αξιόπιστο μηχανισμό για τη διεκπεραίωση οικονομικών συναλλαγών. Ο αξιόπιστος χειρισμός των συμβάντων webhook και η απρόσκοπτη ενσωμάτωσή του με το Stripe συμβάλλουν στην αναβαθμισμένη εμπειρία του χρήστη, καθιστώντας τον ένα σημαντικό στοιχείο στα πλαίσια του οικοσυστήματος της εφαρμογής μας.

5.4.5 AccountRequestsController

Σε αυτή την ενότητα, θα αναφερθούμε στις λειτουργίες του AccountRequestsController μέσα στην εφαρμογή μας. Αυτός ο ελεγκτής παίζει καθοριστικό ρόλο στο χειρισμό των αιτημάτων των χρηστών που σχετίζονται με τη διεκδίκηση ενός προσώπου (καλλιτέχνη). Παρέχει τέσσερις διαφορετικές μεθόδους, κάθε μία από τις οποίες εξυπηρετεί συγκεκριμένες ενέργειες στον κύκλο ζωής του αιτήματος λογαριασμού χρήστη.

Αίτηση Λογαριασμού:

Για τους απλούς χρήστες που επιθυμούν να διεκδικήσουν ένα πρόσωπο (καλλιτέχνη) ως τον εαυτό τους, το τελικό σημείο POST `/api/AccountRequests/RequestAccount` μπαίνει στο προσκήνιο. Οι χρήστες μπορούν να υποβάλουν τα αιτήματά τους παρέχοντας βασικές λεπτομέρειες με τη μορφή ενός αντικειμένου CreateAccountRequestDto. Αυτό το αντικείμενο περιλαμβάνει πεδία όπως το personId (που προσδιορίζει το πρόσωπο που θα ζητηθεί) και το identificationDocument (μια συμβολοσειρά base64 που αντιπροσωπεύει το έγγραφο PDF που επαληθεύει την ταυτότητα του χρήστη).

Αιτήματα για Διαχειριστές Αιτημάτων:

Το τελικό σημείο GET `api/AccountRequests/ClaimsManagers` χρησιμεύει ως ένα ολοκληρωμένο εργαλείο για τους διαχειριστές αιτημάτων για την προβολή όλων των αιτήσεων λογαριασμού που έχουν γίνει από απλούς χρήστες. Προσβάσιμο μόνο σε χρήστες με το ρόλο του διαχειριστή απαιτήσεων, αυτό το τελικό σημείο προσφέρει ορατότητα στις εκκρεμείς αιτήσεις, διευκολύνοντας την αποτελεσματική διαχείριση και λήψη αποφάσεων.

Έγκριση και απόρριψη αιτήσεων:

Οι διαχειριστές απαιτήσεων έχουν αποκλειστικά δικαιώματα έγκρισης ή απόρριψης αιτημάτων χρηστών. Το τελικό σημείο GET `/api/AccountRequests/Approve/{requestId}` επιτρέπει στους διαχειριστές απαιτήσεων να εγκρίνουν το αίτημα ενός συγκεκριμένου χρήστη, σηματοδοτώντας την αποδοχή της ζητούμενης συσχέτισης προσώπου.

Αντίθετα, το τελικό σημείο GET `/api/AccountRequests/Reject/{requestId}` δίνει τη δυνατότητα στους διαχειριστές απαιτήσεων να απορρίψουν το αίτημα ενός χρήστη, παρέχοντας μια συγκροτημένη ροή εργασιών για τη διαχείριση των απαιτήσεων των χρηστών.

Έλεγχος πρόσβασης βάσει ρόλων:

Η πρόσβαση σε αυτά τα τελικά σημεία ελέγχεται σχολαστικά με βάση τους ρόλους των χρηστών. Οι διαχειριστές απαιτήσεων, με υψηλές άδειες, μπορούν να αλληλεπιδρούν με όλες τις λειτουργίες, εξασφαλίζοντας ένα ελεγχόμενο και ασφαλές περιβάλλον για τη διαχείριση των αιτημάτων των χρηστών. Αυτός ο έλεγχος πρόσβασης βάσει ρόλων ενισχύει τη συνολική ασφάλεια και ακεραιότητα της διαδικασίας υποβολής αιτήσεων από χρήστες.

Περιληπτικά, ο AccountRequestsController χρησιμεύει ως ένας κεντρικός κόμβος για την ομαδοποίηση των αιτημάτων των χρηστών που σχετίζονται με τη διεκδίκηση προσώπων εντός της εφαρμογής. Τα ειδικά για κάθε ρόλο τερματικά του σημεία και ο προσεκτικός έλεγχος πρόσβασης συμβάλλουν σε μια απλοποιημένη και ασφαλή διαδικασία διεκδίκησης χρηστών, διασφαλίζοντας ότι μόνο εξουσιοδοτημένο προσωπικό μπορεί να διαχειριστεί και να επικυρώσει αυτά τα αιτήματα.

5.4.6 TransactionsController

Σε αυτή την ενότητα, θα εξεταστούν οι λειτουργίες του TransactionsController, ενός βασικού συστατικού της εφαρμογής μας που είναι υπεύθυνο για την εποπτεία και τη διαχείριση των συναλλαγών των χρηστών. Σχεδιασμένος αποκλειστικά για τους διαχειριστές, αυτός ο ελεγκτής παρέχει τρία διαφορετικά τελικά σημεία, καθένα από τα οποία είναι προσαρμοσμένο ώστε να προσφέρει ακριβείς πληροφορίες για τα δεδομένα που σχετίζονται με τις συναλλαγές.

Ανάκτηση μιας συναλλαγής με αναγνωριστικό:

Το τελικό σημείο GET `/api/Transactions/{id}` διευκολύνει τους διαχειριστές στην ανάκτηση λεπτομερών πληροφοριών σχετικά με μια συγκεκριμένη συναλλαγή. Καθορίζοντας το αναγνωριστικό της συναλλαγής, οι διαχειριστές μπορούν να έχουν πρόσβαση σε ολοκληρωμένα δεδομένα, όπως το ποσό της πίστωσης, την αιτία και την ημερομηνία δημιουργίας της συναλλαγής. Αυτό το τελικό σημείο χρησιμεύει ως πολύτιμο εργαλείο για τους διαχειριστές που επιθυμούν να ελέγξουν ή να διερευνήσουν μεμονωμένες συναλλαγές.

Ανάκτηση συναλλαγών χρήστη:

Το τελικό σημείο GET `/api/Transactions/user/{id}` δίνει τη δυνατότητα στους διαχειριστές να ανακτούν όλες τις συναλλαγές που σχετίζονται με έναν συγκεκριμένο χρήστη. Αυτή η λειτουργικότητα αποδεικνύεται πολύτιμη για την παρακολούθηση των οικονομικών δραστηριοτήτων των χρηστών και τη διασφάλιση της συμμόρφωσης με τις καθιερωμένες πολιτικές.

Πληρωμή επαληθευμένων email με εκκρεμείς πιστώσεις:

Το τελικό σημείο GET `/api/Transactions/Pay-Verified-Emails-Not-Paid` απευθύνεται σε ένα συγκεκριμένο σενάριο όπου οι χρήστες, μετά την επαλήθευση του email τους, δικαιούνται να λάβουν ένα μικρό ποσό πιστώσεων. Ωστόσο, σε περιπτώσεις όπου οι χρήστες έχουν επαληθεύσει το email τους αλλά δεν έχουν λάβει τα αντίστοιχα χρήματα, οι διαχειριστές μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να διορθώσουν την κατάσταση. Αυτή η λειτουργία διασφαλίζει ότι οι χρήστες λαμβάνουν αμέσως τις πιστώσεις που δικαιούνται, ενισχύοντας την ικανοποίηση των χρηστών και την ακεραιότητα του συστήματος.

Διοικητική εποπτεία:

Ο TransactionsController είναι προσβάσιμος αποκλειστικά από τους διαχειριστές, ενισχύοντας την αρχή των ελάχιστων προνομίων και διασφαλίζοντας ότι μόνο εξουσιοδοτημένο προσωπικό μπορεί να

έχει πρόσβαση και να διαχειρίζεται δεδομένα που σχετίζονται με συναλλαγές. Αυτός ο ελεγκτής κατέχει καθοριστικό ρόλο στη διατήρηση της διαφάνειας, της λογοδοσίας και της οικονομικής ακρίβειας εντός της εφαρμογής.

Συνοψίζοντας, ο TransactionsController χρησιμεύει ως βασικό εργαλείο για τους διαχειριστές για την πλοήγηση και τη διαχείριση των συναλλαγών των χρηστών. Τα προσεκτικά σχεδιασμένα τελικά σημεία του παρέχουν μια λεπτομερή και ολοκληρωμένη προβολή των μεμονωμένων συναλλαγών, ιστορικό συναλλαγών ανά χρήστη και έναν μηχανισμό για τη διόρθωση πιστωτικών ασυμφωνιών για χρήστες που έχουν επαληθεύσει το email τους αλλά δεν έχουν λάβει τα χρήματά τους. Αυτή η σχολαστική εποπτεία συμβάλλει στη συνολική οικονομική ακεραιότητα και την ικανοποίηση των χρηστών εντός της εφαρμογής.

5.4.7 ContributionController

Ο ContributionController διαχειρίζεται τις συνεργασίες των ηθοποιών/καλλιτεχνών με κάποια παραγωγή και με τον ρόλο του ηθοποιού σε μία παράσταση, επιτρέποντας στους χρήστες να διαχειρίζονται και να ανακτούν συνεισφορές που σχετίζονται με άτομα, παραγωγές και ρόλους. Ακολουθεί μια συνοπτική επισκόπηση των λειτουργιών που παρέχονται από αυτόν τον ελεγκτή:

Ανάκτηση συνεισφορών:

GET `/api/contributions`: Ανάκτηση όλων των συνεισφορών από τη βάση δεδομένων με προαιρετική σελιδοποίηση. Η απόκριση περιλαμβάνει έναν πίνακα αντικειμένων συνεισφορών, καθένα από τα οποία περιέχει πληροφορίες όπως id, peopleId, productionId, roleId, subRole, systemId και timestamp.

Δημιουργία μιας ενιαίας συνεισφοράς:

POST `/api/contributions`: Δημιουργεί μια νέα συσχέτιση μεταξύ ενός ατόμου, μιας παραγωγής και ενός ρόλου. Δέχεται ένα CreateContributionDto ως είσοδο, συμπεριλαμβανομένων των personId, productionId, roleId, subRole και systemId.

Δημιουργία πολλαπλών συνεισφορών:

POST `/api/Contributions/range`: Δημιουργεί πολλαπλές συνδέσεις ταυτόχρονα. Δέχεται μια λίστα αντικειμένων CreateContributionDto, καθένα από τα οποία καθορίζει το personId, productionId, roleId, subRole και systemId για τις αντίστοιχες συνεισφορές.

5.4.8 CurationController

Στον τομέα της διαχείρισης δεδομένων, ο CuratorController παίζει ζωτικό ρόλο στη διασφάλιση της καθαρότητας και της ακεραιότητας της βάσης δεδομένων της εφαρμογής μας. Αυτός ο ελεγκτής διαχειρίζεται ενέργειες που ξεκινούν από απλές διαδικασίες καθαρισμού δεδομένων και φτάνουν μέχρι

τη διατήρηση της συνέπειας των συνεισφορών και των ρόλων. Ας εμβαθύνουμε στις συγκεκριμένες ενέργειες που έχουν σχεδιαστεί για τη βελτίωση και τη βελτιστοποίηση του τοπίου δεδομένων.

Καθαρισμός δεδομένων:

GET `/api/curator/curatesimple`: Εκτελεί μια απλή διαδικασία καθαρισμού δεδομένων, εξαλείφοντας διπλά διαστήματα και ειδικά σύμβολα από καθορισμένους πίνακες και στήλες.

Διαγραφή συνεισφορών για ανύπαρκτα άτομα:

GET `/api/curator/contributions/delete/ForNonExistent/people`: Αφαιρεί συνεισφορές που σχετίζονται με διαγραμμένα ή ανύπαρκτα άτομα, διασφαλίζοντας την ακεραιότητα των δεδομένων.

Διαγραφή συνεισφορών για ανύπαρκτες παραγωγές:

GET `/api/curator/contributions/delete/ForNonExistent/Productions`: Αφαιρεί συνεισφορές που σχετίζονται με διαγραμμένες ή ανύπαρκτες παραγωγές.

Levenshtein Distance: Απλοποίηση ρόλων με χρήση της απόστασης Levenshtein:

GET `/api/curator/roles/simplify/LevenshteinDistance`: Προσπαθεί να απλοποιήσει τους ρόλους χρησιμοποιώντας ένα προσαρμοσμένο λεξικό. Σημείωση: Αυτή η λειτουργία βρίσκεται σε εξέλιξη και πρέπει να βελτιωθεί σε μελλοντικές εκδόσεις.

5.4.9 EventsController

Ο EventsController χρησιμεύει ως ο βασικός κόμβος για το χειρισμό των συμβάντων εντός της εφαρμογής. Τα γεγονότα αντιπροσωπεύουν συμβάντα που σχετίζονται με παραγωγές και παραστάσεις. Αυτός ο ελεγκτής παρέχει μια σειρά λειτουργιών που επιτρέπουν στους χρήστες να αλληλεπιδρούν αποτελεσματικά με τα δεδομένα συμβάντων. Από την ανάκτηση λεπτομερών πληροφοριών σχετικά με τα συμβάντα έως τη δημιουργία νέων, την ενημέρωση των λεπτομερειών τιμολόγησης και τη διαχείριση της ιδιοκτησίας των συμβάντων, ο EventsController συμβάλλει καθοριστικά στη διασφάλιση της απρόσκοπτης διαχείρισης των συμβάντων.

Κατάλογος όλων των εκδηλώσεων:

GET `/api/events`: Ανάκτηση μιας λίστας όλων των συμβάντων στο σύστημα, με δυνατότητα φιλτραρίσματος διεκδικούμενων ή μη διεκδικούμενων συμβάντων και σελιδοποίηση.

Δημιουργία συμβάντος:

POST `/api/events`: Δημιουργία μιας νέας εκδήλωσης παρέχοντας βασικές λεπτομέρειες, όπως η ημερομηνία της εκδήλωσης, το εύρος τιμών, η σχετική παραγωγή, ο χώρος διεξαγωγής και το αναγνωριστικό του συστήματος.

Αναζήτηση συμβάντος με ID Περιγραφή:

GET `/api/events/{id}`: Λήψη λεπτομερών πληροφοριών σχετικά με ένα συγκεκριμένο συμβάν χρησιμοποιώντας το μοναδικό αναγνωριστικό του.

Αναζήτηση εκδηλώσεων με βάση το αναγνωριστικό προσώπου:

GET `/api/events/person/{id}`: Ανάκτηση μιας λίστας γεγονότων που σχετίζονται με ένα συγκεκριμένο άτομο, τα οποία προσδιορίζονται από το μοναδικό αναγνωριστικό τους.

Αναζήτηση εκδηλώσεων με βάση το αναγνωριστικό παραγωγών:

GET `/api/events/productions/{id}`: Ανάκτηση μιας λίστας συμβάντων που συνδέονται με μια συγκεκριμένη παραγωγή χρησιμοποιώντας το μοναδικό αναγνωριστικό της.

Δημιουργία πολλαπλών συμβάντων:

POST `/api/events/range`: Ανάρτηση μιας λίστας νέων εκδηλώσεων μαζικά, διευκολύνοντας την αποτελεσματική προσθήκη πολλών εκδηλώσεων ταυτόχρονα.

Ενημέρωση τιμής συμβάντος:

PUT `/api/events/update/price`: Ενημερώνει το εύρος τιμών μιας συγκεκριμένης εκδήλωσης παρέχοντας το μοναδικό αναγνωριστικό της εκδήλωσης και το νέο εύρος τιμών.

Διεκδίκηση συμβάντος:

POST `/api/events/claim-event/{id}`: Επιτρέπει στους χρήστες να διεκδικήσουν την κυριότητα ενός συγκεκριμένου συμβάντος παρέχοντας το μοναδικό αναγνωριστικό του συμβάντος.

Ενημέρωση λεπτομερειών συμβάντος:

PUT `/api/events/update`: Ενημέρωση διαφόρων λεπτομερειών ενός συγκεκριμένου συμβάντος, διασφαλίζοντας ότι οι πληροφορίες του συμβάντος παραμένουν ακριβείς και ενημερωμένες.

5.4.10 LogsController

Σε αυτή την ενότητα, θα ασχοληθούμε με τον LogsController, ο οποίος αποτελεί ένα κρίσιμο εργαλείο για τους διαχειριστές. Ο ελεγκτής παρέχει μια μοναδική μέθοδο που επιτρέπει την πρόσβαση στα αρχεία καταγραφής που είναι αποθηκευμένα στη βάση δεδομένων της εφαρμογής. Αυτά τα αρχεία καταγραφής προσφέρουν πολύτιμες πληροφορίες και είναι προσβάσιμα αποκλειστικά στους διαχειριστές για σκοπούς παρακολούθησης και ανάλυσης.

Λήψη αρχείων καταγραφής (μόνο για διαχειριστές):

GET `/api/logs`: Ανάκτηση αρχείων καταγραφής από τη βάση δεδομένων, η οποία περιορίζεται στους διαχειριστές για σκοπούς παρακολούθησης και ανάλυσης.

5.4.11 OrganizersController

Σε αυτή την ενότητα εξερευνάται ο OrganizersController, ο οποίος περιλαμβάνει τρεις διαφορετικές μεθόδους προσαρμοσμένες στη διαχείριση των διοργανωτών.

POST /api/organizers:

Περιγραφή: Αυτή η μέθοδος διευκολύνει τη δημιουργία ενός νέου διοργανωτή, δεχόμενη διάφορες παραμέτρους όπως όνομα, διεύθυνση, πόλη, ταχυδρομικός κώδικας, email, τηλέφωνο, μεταξύ άλλων.

Χρήση: Οι διαχειριστές μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να προσθέσουν διοργανωτές στη βάση δεδομένων.

GET /api/organizers:

Περιγραφή: Σχεδιασμένη για σκοπούς ανάκτησης, αυτή η μέθοδος επιτρέπει στους χρήστες να ανακτήσουν όλους τους διοργανωτές που είναι αποθηκευμένοι στη βάση δεδομένων. Υποστηρίζει σελιδοποίηση και ταξινόμηση σε αλφαβητική σειρά.

POST /api/organizers/range:

Περιγραφή: Ιδανική για μαζικές ενέργειες, αυτή η μέθοδος επιτρέπει τη δημιουργία μιας σειράς νέων διοργανωτών στη βάση δεδομένων.

Χρήση: Οι διαχειριστές που επιθυμούν να προσθέσουν αποτελεσματικά πολλούς διοργανωτές μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να απλοποιήσουν την εν λόγω διαδικασία.

5.4.12 PeopleController

Αυτή η ενότητα εξετάζει τον PeopleController, ένα ολοκληρωμένο σύνολο έντεκα τελικών σημείων που εξυπηρετούν διάφορες λειτουργίες που σχετίζονται με τα άτομα (υθοποιούς/καλλιτέχνες) στην εφαρμογή.

GET /api/people/{id}:

Περιγραφή: Ανακτά λεπτομερείς πληροφορίες για ένα άτομο με βάση το μοναδικό αναγνωριστικό του.

DELETE /api/people/{id}:

Περιγραφή: Επιτρέπει στους διαχειριστές να διαγράφουν ένα άτομο από τη βάση δεδομένων χρησιμοποιώντας το μοναδικό αναγνωριστικό του.

GET /api/people:

Περιγραφή: Υποστηρίζει επιλογές σελιδοποίησης και φιλτραρίσματος, επιτρέποντας στους χρήστες να ανακτήσουν μια λίστα ατόμων. Οι παράμετροι περιλαμβάνουν showAvailable, αλφαβητική σειρά, ρόλο, ηλικία, ύψος, βάρος, χρώμα ματιών, χρώμα μαλλιών και γνώση γλώσσας.

Χρήση: Οι χρήστες, μπορούν να προσαρμόζουν τα ερωτήματά τους για να λαμβάνουν συγκεκριμένα σύνολα ατόμων με βάση διάφορα κριτήρια.

POST /api/people:

Περιγραφή: Δημιουργεί ένα νέο άτομο στη βάση δεδομένων, δεχόμενος στοιχεία όπως όνομα, ηλικία, ρόλο κ.λπ.

Χρήση: Οι διαχειριστές μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να προσθέσουν νέα άτομα στην εφαρμογή.

POST /api/people/range:

Περιγραφή: Επιτρέπει τη δημιουργία πολλών νέων προσώπων ταυτόχρονα.

Χρήση: Χρήσιμο για διαχειριστές που επιθυμούν να προσθέσουν αποτελεσματικά πολλά άτομα στη βάση δεδομένων.

GET /api/people/role/{role}:

Περιγραφή: Ανακτά άτομα με βάση έναν καθορισμένο ρόλο.

GET /api/people/initials/{letters}:

Περιγραφή: Επιστρέφει τα άτομα των οποίων τα αρχικά αρχίζουν με τα καθορισμένα γράμματα.

GET /api/people/{id}/productions:

Περιγραφή: Επιστρέφει όλες τις παραγωγές που σχετίζονται με ένα συγκεκριμένο άτομο.

GET /api/people/{id}/photos:

Περιγραφή: Ανακτά όλες τις φωτογραφίες που σχετίζονται με ένα συγκεκριμένο άτομο (μη χρήστη).

GET /api/people/photos:

Περιγραφή: Αντλεί όλες τις εικόνες από τη βάση δεδομένων που σχετίζονται με πρόσωπα.

5.4.13 ProductionsController:

Αυτή η ενότητα εξετάζει τον ProductionsController, ένα τμήμα της εφαρμογής που είναι υπεύθυνο για τη διαχείριση των παραγωγών. Ο ελεγκτής προσφέρει λειτουργίες για την αποτελεσματική δημιουργία, και ανάκτηση παραγωγών.

POST /api/productions:

Περιγραφή: Επιτρέπει στους διαχειριστές χρήστες να δημιουργήσουν μια νέα παραγωγή παρέχοντας λεπτομέρειες όπως τίτλο, περιγραφή, ημερομηνία και άλλα.

GET /api/productions:

Περιγραφή: Οι χρήστες μπορούν να ανακτήσουν έναν πλήρη κατάλογο των παραγωγών με δυνατότητα σελιδοποίησης των αποτελεσμάτων.

POST /api/productions/range:

Περιγραφή: Επιτρέπει τη δημιουργία πολλαπλών παραγωγών ταυτόχρονα.

Χρήση: Οι διαχειριστές μπορούν να προσθέσουν αποτελεσματικά πολλές παραγωγές στη βάση δεδομένων χρησιμοποιώντας αυτό το τελικό σημείο.

GET /api/productions/{id}:

Description: Ανακτά λεπτομερείς πληροφορίες σχετικά με μια συγκεκριμένη παραγωγή με βάση το μοναδικό αναγνωριστικό της.

Χρήση: Οι χρήστες, μπορούν να λάβουν συγκεκριμένες λεπτομέρειες σχετικά με μια παραγωγή παρέχοντας το μοναδικό αναγνωριστικό της.

5.4.14 RolesController

Αυτή η ενότητα εστιάζεται στον RolesController, ένα ουσιώδες συστατικό που είναι υπεύθυνο για τη διαχείριση των ρόλων των υθοποιών μέσα στην εφαρμογή. Ο ελεγκτής προσφέρει λειτουργίες για τη δημιουργία και την ανάκτηση.

POST /api/roles:

Περιγραφή: Επιτρέπει στους χρήστες να δημιουργήσουν ένα νέο ρόλο παρέχοντας το όνομα του ρόλου.

Χρήση: Οι διαχειριστές, μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να προσθέσουν νέους ρόλους στην εφαρμογή.

GET /api/roles:

Περιγραφή: Παρέχει μια λίστα όλων των ρόλων στη βάση δεδομένων.

Χρήση: Οι χρήστες μπορούν να ανακτήσουν μια ολοκληρωμένη λίστα με τους ρόλους που είναι διαθέσιμοι αυτή τη στιγμή στην εφαρμογή.

POST /api/roles/range:

Περιγραφή: Επιτρέπει τη δημιουργία πολλαπλών ρόλων ταυτόχρονα.

Χρήση: Οι διαχειριστές μπορούν να προσθέσουν αποτελεσματικά πολλούς ρόλους στη βάση δεδομένων χρησιμοποιώντας αυτό το τελικό σημείο.

5.4.15 ShowsController

Αυτή η ενότητα περιγράφει τον ShowsController, ένα εξειδικευμένο στοιχείο που έχει σχεδιαστεί για να ενοποιεί πληροφορίες από διάφορες οντότητες μέσα στην εφαρμογή, ώστε να παρέχει μια ολοκληρωμένη εικόνα των παραστάσεων. Αν και αυτός ο ελεγκτής προσφέρει μόνο μία μέθοδο, η λειτουργικότητά του είναι κομβικής σημασίας για την ανάκτηση λεπτομερών πληροφοριών σχετικά με τις παραστάσεις.

GET /api/shows:

Περιγραφή: Ανακτά μια λίστα παραστάσεων με βάση τις καθορισμένες παραμέτρους του ερωτήματος, συμπεριλαμβανομένης της διεύθυνσης του χώρου, του αναγνωριστικού του χώρου, της ημερομηνίας εκδήλωσης, του ονόματος του διοργανωτή, του τίτλου της παραγωγής, της σελίδας και του μεγέθους.

Χρήση: Οι χρήστες μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να λάβουν μια προσαρμοσμένη λίστα παραστάσεων με βάση συγκεκριμένα κριτήρια, επιτρέποντας μια λεπτομερή εξερεύνηση των εκδηλώσεων εντός της εφαρμογής.

Ο ShowsController απλοποιεί τη διαδικασία πρόσβασης και οργάνωσης πληροφοριών σχετικά με τις παραστάσεις, παρέχοντας στους χρήστες ένα ευέλικτο εργαλείο για την εξερεύνηση και το φιλτράρισμα των παραστάσεων σύμφωνα με τις προτιμήσεις τους. Η υλοποίηση της προσωρινής αποθήκευσης μνήμης βελτιστοποιεί την απόδοση, καθιστώντας τη διαδικασία ανάκτησης πιο αποτελεσματική για τους χρήστες.

5.4.16 VenuesController

Αυτή η ενότητα περιγράφει τον VenuesController, ένα σημαντικό στοιχείο που είναι υπεύθυνο για τη διαχείριση των χώρων διεξαγωγής μέσα στην εφαρμογή. Ο ελεγκτής προσφέρει μια σειρά τελικών σημείων που διευκολύνουν την ανάκτηση, τη δημιουργία και την ενημέρωση των πληροφοριών των χώρων.

GET /api/venues:

Περιγραφή: Ανακτά μια φιλτραρισμένη λίστα χώρων διεξαγωγής με βάση καθορισμένες παραμέτρους, όπως σελίδα, μέγεθος, αλφαβητική σειρά, αναζήτηση διεύθυνσης, τίτλος χώρου και διαθεσιμότητα για διεκδίκηση.

Χρήση: Οι χρήστες μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να εξερευνήσουν τους χώρους διεξαγωγής με βάση διάφορα κριτήρια, απλοποιώντας την αναζήτηση κατάλληλων χώρων για εκδηλώσεις.

POST /api/venues:

Περιγραφή: Δημιουργεί ένα νέο χώρο με διάφορα χαρακτηριστικά, όπως όνομα, διεύθυνση, πόλη, ταχυδρομικός κώδικας, email, τηλέφωνο και άλλα.

Χρήση: Οι διαχειριστές χώρων ή εξουσιοδοτημένοι χρήστες μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να προσθέσουν νέους χώρους στην εφαρμογή.

GET /api/venues/{id}:

Περιγραφή: Ανακτά αναλυτικές πληροφορίες σχετικά με έναν συγκεκριμένο χώρο που προσδιορίζεται από το μοναδικό αναγνωριστικό του.

Χρήση: Οι χρήστες μπορούν να έχουν πρόσβαση σε αυτό το τελικό σημείο για να λάβουν συγκεκριμένες λεπτομέρειες σχετικά με έναν συγκεκριμένο χώρο εντός της εφαρμογής.

GET /api/venues/{venueId}/productions:

Περιγραφή: Ανακτά μια σελιδοποιημένη λίστα των παραγωγών που σχετίζονται με έναν καθορισμένο χώρο.

Χρήση: Αυτό το τελικό σημείο επιτρέπει στους χρήστες να εξερευνήσουν τις παραγωγές που φιλοξενούνται σε έναν συγκεκριμένο χώρο, παρέχοντας πολύτιμες πληροφορίες για τις εκδηλώσεις που πραγματοποιούνται εκεί.

POST /api/venues/range:

Περιγραφή: Δημιουργεί πολλαπλούς χώρους διεξαγωγής στην εφαρμογή με ένα μόνο αίτημα, παρέχοντας μια βολική λειτουργία μαζικής προσθήκης.

Χρήση: Οι διαχειριστές ή οι εξουσιοδοτημένοι χρήστες μπορούν να χρησιμοποιήσουν αυτό το τελικό σημείο για να προσθέσουν αποτελεσματικά πολλούς χώρους διεξαγωγής στο σύστημα.

POST /api/venues/claim-venue/{id}:

Περιγραφή: Επιτρέπει στους χρήστες να διεκδικήσουν την κυριότητα ενός συγκεκριμένου χώρου.

PUT /api/venues/update:

Περιγραφή: Ενημερώνει τις πληροφορίες για έναν συγκεκριμένο χώρο, παρέχοντας έναν μηχανισμό για τη διατήρηση των στοιχείων του χώρου σε ισχύ.

Χρήση: Εξουσιοδοτημένοι χρήστες μπορούν να χρησιμοποιούν αυτό το τελικό σημείο για να τροποποιούν τα χαρακτηριστικά ενός χώρου, όπως απαιτείται.

Ο VenuesController παίζει καθοριστικό ρόλο στη διατήρηση ενός οργανωμένου και ολοκληρωμένου καταλόγου χώρων, δίνοντας στους χρήστες τη δυνατότητα να εξερευνήσουν, να συνεισφέρουν και να διεκδικήσουν την ιδιοκτησία αυτών των συστατικών στοιχείων της εφαρμογής.

5.5 Razor Pages

Σε αυτή την ενότητα, θα αναλύσουμε την υλοποίηση και τη σημασία των Razor Pages στην εφαρμογή μας. Τα Razor Pages είναι ένα χαρακτηριστικό του ASP.NET Core που διευκολύνει τη δημιουργία δυναμικών ιστοσελίδων, προωθώντας μια βελτιωμένη και συνεκτική εμπειρία χρήστη. Παρακάτω, εμβαθύνουμε στην εφαρμογή των Razor Pages και στον τρόπο με τον οποίο συμβάλλουν στη συνολική λειτουργικότητα και αισθητική της εφαρμογής.

Οι Razor Pages στο ASP.NET Core παρέχουν ένα συμπαγές και εκφραστικό μοντέλο προγραμματισμού για τη δημιουργία ιστοσελίδων με κώδικα από την πλευρά του διακομιστή πλήρως ενσωματωμένο στην HTML. Σε αντίθεση με τα παραδοσιακά πλαίσια MVC, το Razor Pages δίνει έμφαση στην απλότητα και τη συμβατικότητα, καθιστώντας το μια εξαιρετική επιλογή για σενάρια όπου προτιμάται μια απλή προσέγγιση με επίκεντρο τη σελίδα.

Στην εφαρμογή, οι Razor Pages παίζουν καθοριστικό ρόλο στο χειρισμό και την επικοινωνία της κατάστασης των συναλλαγών των χρηστών. Αυτή η ενότητα εμβαθύνει στη συγκεκριμένη υλοποίηση τριών βασικών Razor Pages: Cancel.cshtml, Success.cshtml και EmailVerification.cshtml. Αυτές οι σελίδες παίζουν καθοριστικό ρόλο στην παροχή μιας προσαρμοσμένης και φιλικής προς τον χρήστη εμπειρίας με βάση το αποτέλεσμα των διαδικασιών συναλλαγής.

Cancel.cshtml: Σελίδα ανακατεύθυνσης ακυρωμένης συναλλαγής.

Η σελίδα Cancel.cshtml έχει σχεδιαστεί για να διαχειρίζεται σενάρια όπου η συναλλαγή ενός χρήστη έχει ακυρωθεί.

Success.cshtml: Σελίδα ανακατεύθυνσης ολοκληρωμένης συναλλαγής.

Η σελίδα Success.cshtml είναι διαθέσιμη στους χρήστες των οποίων οι συναλλαγές ολοκληρώθηκαν με επιτυχία.

EmailVerification.cshtml: Δυναμικός χειρισμός επαλήθευσης ηλεκτρονικού ταχυδρομείου

Η σελίδα EmailVerification.cshtml εμφανίζει δυναμικά διαφορετικά στοιχεία ανάλογα με την παράμετρο κατάστασης που λαμβάνεται.

Οι σελίδες Razor με απεικόνιση υπό προϋποθέσεις χρησιμοποιούνται για την προσαρμογή των στοιχείων εμφάνισης ανάλογα με την παράμετρο κατάστασης.

Οι χρήστες που υποβάλλονται στη διαδικασία επαλήθευσης ηλεκτρονικού ταχυδρομείου λαμβάνουν μια προσαρμοσμένη οπτική αναπαράσταση ανάλογα με την κατάσταση επαλήθευσης, εξασφαλίζοντας σαφήνεια και καθοδηγώντας τους στα απαραίτητα βήματα.

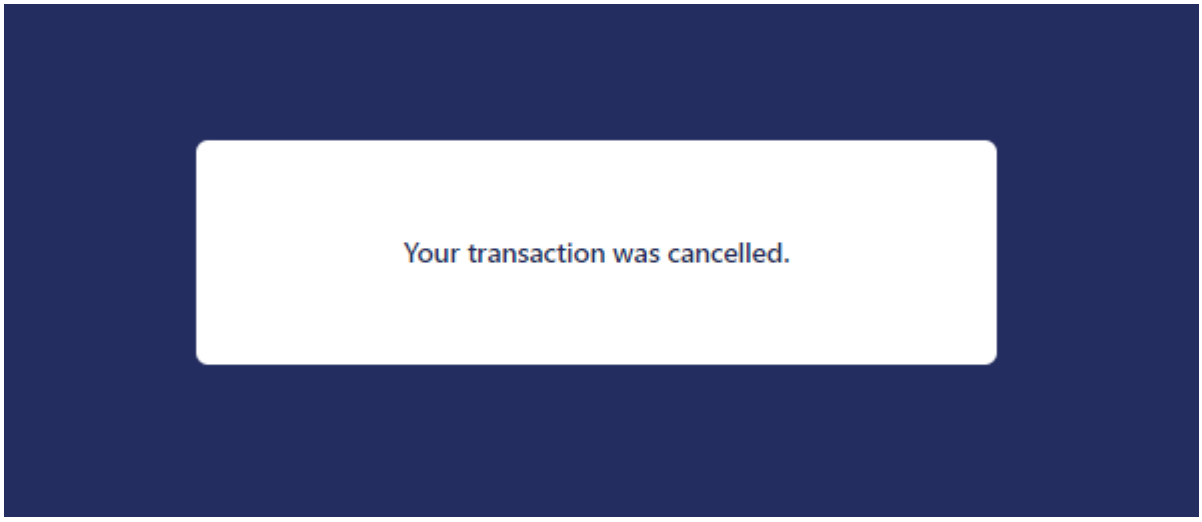
Πιθανά status query parameters για αυτήν την σελίδα:

alreadyVerified: Σελίδα εμφάνισης που εμφανίζεται αν ο χρήστης έχει ήδη επαλήθευση το email του.

success: Σελίδα εμφάνισης μετά από ολοκληρωμένη επαλήθευση email.

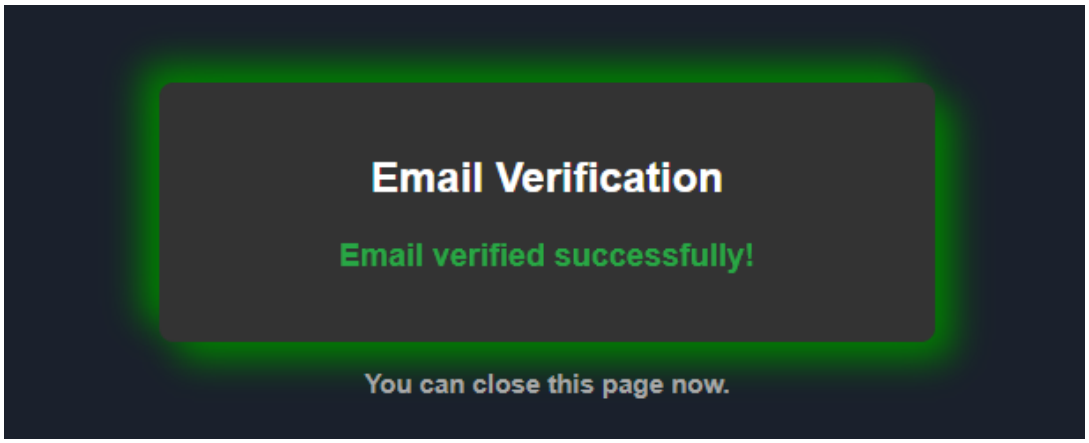
failed: Σελίδα εμφάνισης μετά από ανεπιτυχή επαλήθευση.

Οι σελίδες Razor χρησιμοποιούνται για τη δημιουργία μιας οπτικά ευχάριστης σελίδας, η οποία υποδεικνύει το αποτέλεσμα της εκτέλεσης κάποιας πράξης.

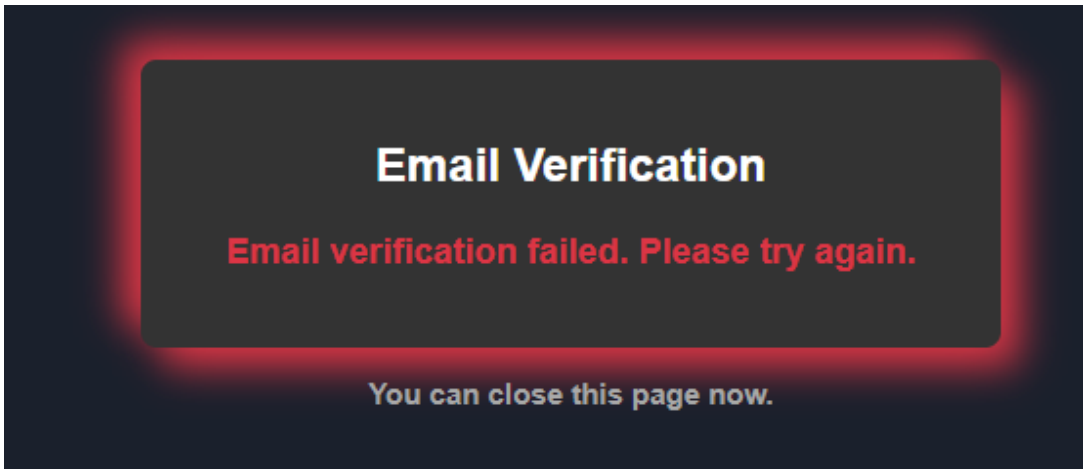


Screenshot 5.1: Cancel.cshtml

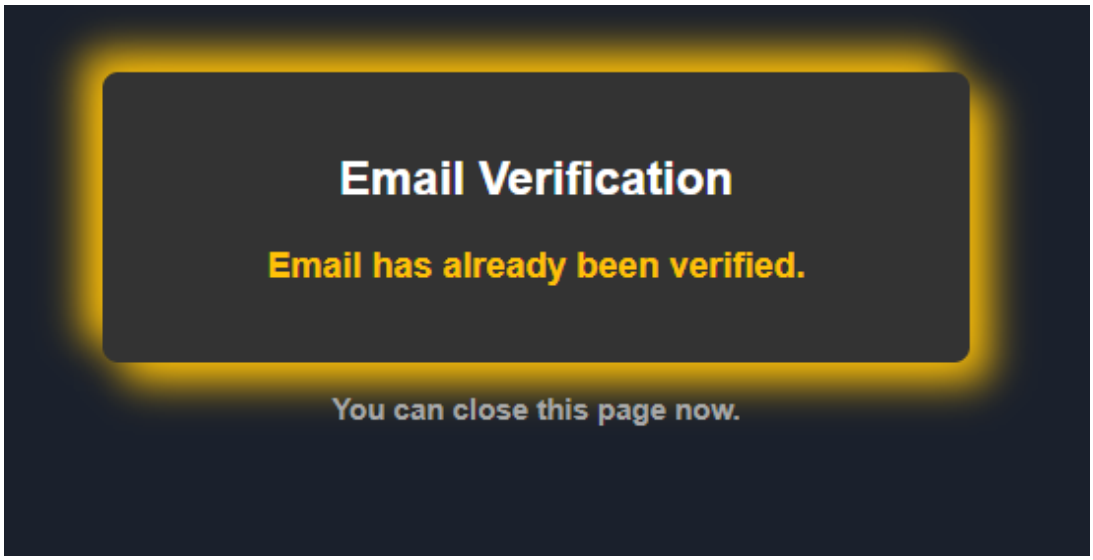
Screenshot 5.2: Success.cshtml



Screenshot 5.3: EmailVerification.cshtml με παράμετρο status success



Screenshot 5.4: EmailVerification.cshtml με παράμετρο status failed



Screenshot 5.5: EmailVerification.cshtml με παράμετρο status already-verified

5.6 Swagger

Σε αυτή την ενότητα, διερευνάται η ενσωμάτωση του Swagger στην εφαρμογή μας, διαφωτίζοντας τη σημασία του και τις συγκεκριμένες επιλογές του Swagger που υλοποιήθηκαν για τον εξορθολογισμό της τεκμηρίωσης και των δοκιμών API.

Το Swagger είναι ένα ισχυρό εργαλείο που απλοποιεί τη διαδικασία ανάπτυξης API παρέχοντας έναν τυποποιημένο τρόπο περιγραφής, τεκμηρίωσης και κατανάλωσης RESTful υπηρεσιών ιστού. Στην εφαρμογή μας, το Swagger χρησιμοποιείται για να βελτιώσει την προσβασιμότητα και την κατανόηση των τελικών σημείων API.

5.6.1 Ενσωμάτωση του Bearer Token στο Swagger UI

Τα bearer tokens αποτελούν θεμελιώδη πτυχή της διασφάλισης των API, διασφαλίζοντας ότι μόνο εξουσιοδοτημένοι χρήστες μπορούν να έχουν πρόσβαση σε προστατευμένους πόρους. Με την ενσωμάτωση της υποστήριξης των Bearer Token στο Swagger UI, ενισχύουμε την πτυχή της τεκμηρίωσης ασφάλειας, καθιστώντας σαφές στους προγραμματιστές και τους χρήστες τον τρόπο με τον οποίο γίνεται ο έλεγχος ταυτότητας εντός του API μας.

Επισκόπηση Υλοποίησης:

Η υλοποίηση περιλαμβάνει τη δημιουργία μιας κλάσης, ConfigureSwaggerOptions, η οποία υλοποιεί τη διεπαφή IConfigureOptions<SwaggerGenOptions>. Αυτή η κλάση διαμορφώνει τις επιλογές δημιουργίας Swagger, ορίζοντας συγκεκριμένα τον τρόπο τεκμηρίωσης των Bearer Tokens εντός της προδιαγραφής Swagger.

```
using Microsoft.Extensions.Options;
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.SwaggerGen;

namespace Theatrical.Api.Swagger;

public class ConfigureSwaggerOptions : IConfigureOptions<SwaggerGenOptions>
{
    public void Configure(SwaggerGenOptions options)
    {
        options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
        {
            In = ParameterLocation.Header,
            Description = "Please provide a valid token",
            Name = "Authorization",
            Type = SecuritySchemeType.Http,
            BearerFormat = "JWT",
            Scheme = "Bearer"
        });

        options.AddSecurityRequirement(new OpenApiSecurityRequirement
        {
            {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
                        Type = ReferenceType.SecurityScheme,
                        Id = "Bearer"
                    }
                },
                Array.Empty<string>()
```

```
            }
        });
    }

}
```

Κώδικας 5.1: Υλοποίηση του JWT στο Swagger UI

Επεξήγηση:

AddSecurityDefinition: Προσδιορίζει το σχήμα ασφαλείας Bearer Token με παραμέτρους όπως τοποθεσία (κεφαλίδα), περιγραφή, όνομα, τύπος (HTTP), μορφή Bearer και σχήμα.

AddSecurityRequirement: Καθορίζει ότι όλες οι λειτουργίες εντός του Swagger UI απαιτούν το Bearer Token ως μέτρο ασφαλείας.

Οφέλη:

Η ενσωμάτωση της διαμόρφωσης του Bearer Token στο Swagger UI εξυπηρετεί πολλαπλούς σκοπούς:

Διαφάνεια ασφάλειας: Επικοινωνεί με σαφήνεια με τον μηχανισμό ελέγχου ταυτότητας (Bearer Token) που απαιτείται για την πρόσβαση στο API.

Διαδραστική τεκμηρίωση: Επιτρέπει στους χρήστες να δοκιμάζουν τα τελικά σημεία με τον σωστό έλεγχο ταυτότητας απευθείας από το περιβάλλον εργασίας Swagger UI.

Προσανατολισμός προγραμματιστή: Βοηθά τους προγραμματιστές να κατανοήσουν την αναμενόμενη μορφή ελέγχου ταυτότητας κατά την αλληλεπίδραση με το API.

Εφαρμόζοντας την υποστήριξη Bearer Token στο Swagger UI, ευθυγραμμίζεται η τεκμηρίωση με τις βέλτιστες πρακτικές του τομέα, προωθώντας ένα ασφαλές και φιλικό προς τους προγραμματιστές περιβάλλον API.

5.6.2 Μετατροπή αριθμού Enum σε συμβολοσειρά

Τα Enums είναι ισχυρές κατασκευές στον προγραμματισμό, παρέχοντας έναν τρόπο αναπαράστασης ενός συνόλου ονομαστικών ολοκληρωτικών σταθερών. Ωστόσο, όταν πρόκειται για τεκμηρίωση API, η παρουσίαση αυτών των enums σε αριθμητική μορφή μπορεί να είναι λιγότερο διαισθητική για τους χρήστες. Το φίλτρο σχήματος Enum που συζητείται εδώ εξυπηρετεί τον σκοπό της μετατροπής των αναπαραστάσεων enum από αριθμούς σε συμβολοσειρές, ενισχύοντας τη σαφήνεια της τεκμηρίωσής μας στο Swagger.

Επισκόπηση υλοποίησης:

Η υλοποίηση περιλαμβάνει τη δημιουργία μιας κλάσης με όνομα EnumSchemaFilter, η οποία υλοποιεί τη διεπαφή ISchemaFilter. Αυτή η κλάση επικεντρώνεται στην εφαρμογή αλλαγών σχήματος ειδικά για enums, διασφαλίζοντας ότι το Swagger UI εμφανίζει τις τιμές enum ως συμβολοσειρές και όχι ως αριθμητικές αναπαραστάσεις.

```
using Microsoft.OpenApi.Any;
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.SwaggerGen;
using Theatrical.Data.enums;

namespace Theatrical.Api.Swagger;

public class EnumSchemaFilter : ISchemaFilter
{
    public void Apply(OpenApiSchema schema, SchemaFilterContext context)
    {
        if (context.Type.isEnum)
        {
            schema.Type = "string";
            schema.Enum.Clear();
            foreach (var name in Enum.GetNames(context.Type))
            {
                schema.Enum.Add(new OpenApiString(name));
            }
        }
    }
}
```

Κώδικας 5.2: Μετατροπή enums σε αλφαριθμητική εξήγηση στο Swagger UI

Επεξήγηση:

Τύπος Enum ελέγχου: Το φίλτρο ελέγχει αν ο παρεχόμενος τύπος είναι το enum του στόχου.

Τροποποίηση σχήματος: Αλλάζει τον τύπο σχήματος σε "string", υποδεικνύοντας ότι οι τιμές enum πρέπει να αντιμετωπίζονται ως συμβολοσειρές.

Μετασχηματισμός τιμών enum: Καθαρίζει τις υπάρχουσες τιμές enum και συμπληρώνει το σχήμα με αναπαραστάσεις OpenApiString των ονομάτων enum.

Οφέλη:

Το φίλτρο Enum Schema Filter προσφέρει πολλά πλεονεκτήματα:

Βελτιωμένη αναγνωσιμότητα:

Παρουσιάζει τις τιμές enum σε μορφή συμβολοσειράς που διαβάζεται από τον άνθρωπο, καθιστώντας την τεκμηρίωση πιο προσιτή.

Φιλικό προς το χρήστη UI Swagger:

Βελτιώνει την εμπειρία του Swagger UI, εμφανίζοντας τα enums σε μορφή εύκολα κατανοητή από τους προγραμματιστές.

Συνεπής τεκμηρίωση:

Εξασφαλίζει συνέπεια στον τρόπο παρουσίασης των τιμών enum, συμβάλλοντας σε μια τυποποιημένη και σαφή τεκμηρίωση API.

5.7 Γενική Ροή μίας μεθόδου endpoint

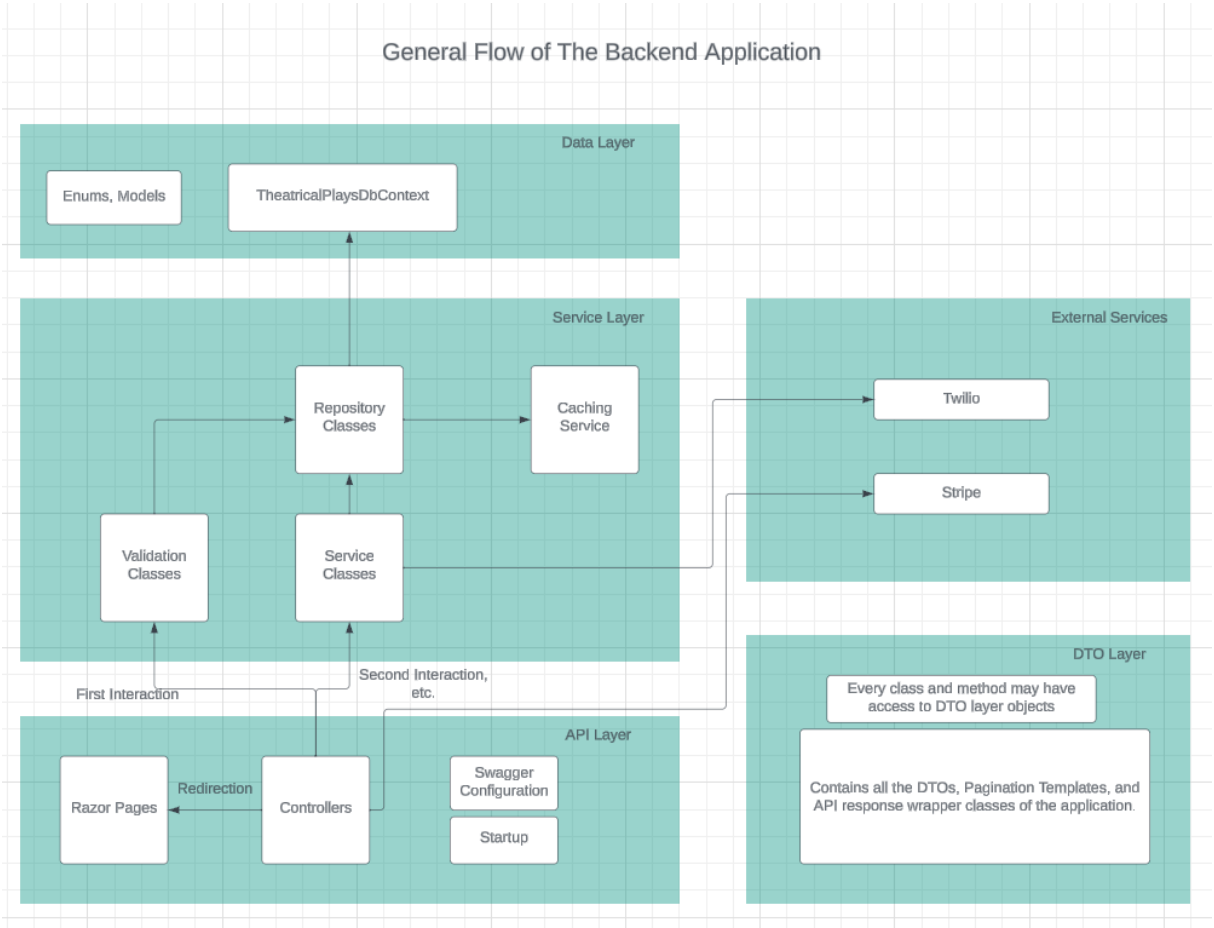
Όταν καλείται μια μέθοδος σε έναν ελεγκτή, ξεκινά μια σειρά αλληλεπιδράσεων με άλλα επίπεδα της εφαρμογής. Το πρώτο σημείο επαφής είναι συνήθως το συστατικό _validation που σχετίζεται με τη σχετική κλάση ή υπηρεσία. Αυτό το βήμα επικύρωσης διασφαλίζει ότι τα εισερχόμενα δεδομένα ή αιτήματα συμμορφώνονται με τους καθορισμένους κανόνες και περιορισμούς, παρέχοντας ένα αρχικό επίπεδο ασφάλειας και ακεραιότητας.

Μετά την επιτυχή επικύρωση, η μέθοδος του ελεγκτή επικοινωνεί στη συνέχεια με μια συγκεκριμένη για τη λειτουργικότητά της κλάση υπηρεσίας. Αυτή η κλάση υπηρεσίας είναι υπεύθυνη για την ενορχήστρωση των βασικών ενεργειών ή της επιχειρηματικής λογικής που σχετίζονται με τη συγκεκριμένη λειτουργία. Μπορεί να περιλαμβάνει αλληλεπιδράσεις με αποθετήρια, εξωτερικές υπηρεσίες ή άλλα στοιχεία εντός του επιπέδου υπηρεσιών.

Με την ενθυλάκωση της βασικής λογικής σε μια κλάση υπηρεσίας, ο ελεγκτής ακολουθεί τις αρχές του διαχωρισμού των προβλημάτων, προωθώντας τη συντηρησιμότητα και την επεκτασιμότητα. Μόλις η κλάση υπηρεσίας εκτελέσει τις απαραίτητες λειτουργίες, ο ελεγκτής χειρίζεται την απόκριση, είτε πρόκειται για τη δημιουργία μιας κατάλληλης απόκρισης HTTP, είτε για την ανακατεύθυνση σε μια συγκεκριμένη προβολή, είτε για την ενεργοποίηση περαιτέρω ενεργειών εντός της εφαρμογής.

Συνοπτικά, η ροή μιας μεθόδου ελεγκτή περιλαμβάνει την αρχική επικύρωση, την ανάθεση σε μια σχετική κλάση υπηρεσίας για την εκτέλεση της επιχειρησιακής λογικής και τον χειρισμό των αποτελεσμάτων που προκύπτουν, ώστε να παρέχεται μια απρόσκοπτη και ελεγχόμενη αλληλεπίδραση με τις λειτουργίες της εφαρμογής.

Ακολουθεί το γενικό διάγραμμα ροής της εφαρμογής:



Screenshot 5.1: Γενικό Διάγραμμα Ροής της εφαρμογής

Εξήγηση Διαγράμματος:

Service Layer:

Repository Classes: Περιέχει κλάσεις που αλληλεπιδρούν με το πλαίσιο της βάσης δεδομένων μέσω του Fluent API. Τα αποθετήρια δεν εκτελούν ελέγχους null.

Caching Service: Υπηρεσία προσωρινής αποθήκευσης: Χρησιμοποιεί μια μέθοδο προσωρινής αποθήκευσης μνήμης για την εφαρμογή. Τα αποθετήρια μπορούν μόνο να χρησιμοποιούν την κλάση caching service.

Service Classes: Περιέχει μια συλλογή από κλάσεις υπηρεσιών που εκτελούν κάποια εργασία, εσωτερικά ή στη βάση δεδομένων χρησιμοποιώντας τις κατάλληλες κλάσεις αποθετηρίου.

Validation Classes: Περιέχει μια συλλογή κλάσεων που επικυρώνουν δεδομένα κατά την ανάκτηση/προσθήκη/επικαιροποίηση/διαγραφή καταχωρήσεων. Οι έλεγχοι null υλοποιούνται εδώ.

API Layer:

Controllers: (UserController για παράδειγμα), Περιέχουν κλάσεις οι οποίες αλληλεπιδρούν με τις κλάσεις επικύρωσης και υπηρεσιών για να εκτελέσουν κάποια ενέργεια. Σημείωση, ο StripeController είναι ο μόνος που έχει πρόσβαση στην εξωτερική εφαρμογή Stripe.

Razor Pages: Περιέχονται όλες η σελίδες razor της εφαρμογής.

Swagger Configuration: Περιέχει τις ρυθμίσεις παραμετροποίησης του swagger ui.

Startup Class: Η Startup Class αναλαμβάνει τη δημιουργία των απαραίτητων εξαρτήσεων για τη λειτουργία της εφαρμογής.

DTO Layer:

Περιέχει όλα τα DTOs, τα πρότυπα σελιδοποίησης και τις κλάσεις περιτύλιξης απόκρισης API της εφαρμογής. Κάθε κλάση μπορεί να έχει πρόσβαση σε αντικείμενα επιπέδου DTO.

External Services:

Περιέχει όλες τις εξωτερικές εφαρμογές, οι οποίες χρησιμοποιούνται για την πραγματοποίηση αγορών μέσω του Stripe και την επιβεβαίωση του τηλεφωνικού αριθμού των χρηστών.

Data Layer:

Περιέχει τα κύρια μοντέλα που χρησιμοποιήθηκαν για την κατασκευή της βάσης δεδομένων. Επίσης περιέχει τα Enums που χρησιμοποιούνται στην εφαρμογή. Επίσης περιέχει το configuration του Entity Framework.

Παράδειγμα: /api/login: Όταν ένας χρήστης θέλει να συνδεθεί, εκτελεί την κατάλληλη μέθοδο (Login) του αντίστοιχου ελεγκτή (UserController).

Η μέθοδος login καλεί την κλάση υπηρεσίας επικύρωσης για ενέργειες που σχετίζονται με τον χρήστη (IUserValidationService, UserValidationService), η οποία ελέγχει αν έχει ήδη δοθεί ένα JWT στις κεφαλίδες. Εάν βρεθεί επιστρέφει, ένα μήνυμα σφάλματος ότι ο χρήστης είναι ήδη συνδεδεμένος. (ένας τρόπος για την υλοποίηση κάποιου είδους κατάστασης σε ένα stateless REST API). Αν δεν έχει δοθεί JWT συνεχίζει την εκτέλεση, καλώντας τη μέθοδο (ValidateForLogin) από την ίδια κλάση επικύρωσης. Στη συνέχεια, η μέθοδος αυτή καλεί το κατάλληλο αποθετήριο για να προσπαθήσει να αντλήσει τον χρήστη που δόθηκε ως παράμετρος. Δημιουργεί μια αναφορά επικύρωσης, εάν ο χρήστης επιστράφηκε ως null, επιστρέφει την αναφορά επικύρωσης με την επιτυχία να έχει οριστεί σε false, με κατάλληλο μήνυμα κ.λπ. Εάν ο χρήστης βρέθηκε, τότε χρησιμοποιεί τη συνάρτηση κατακερματισμού BCrypt για να ελέγξει εάν ο παρεχόμενος κωδικός πρόσβασης είναι ο ίδιος με τον κατακερματισμένο κωδικό πρόσβασης που αποθηκεύτηκε στη βάση δεδομένων κατά την εγγραφή. Στη συνέχεια, επιστρέφει ένα αντικείμενο αναφοράς επικύρωσης με την κατάσταση επιτυχίας να έχει οριστεί σε false εάν ο έλεγχος του κωδικού πρόσβασης απέτυχε. Στη συνέχεια, ελέγχει επίσης αν ο χρήστης έχει ενεργοποιήσει το 2FA και επιστρέφει ένα αντικείμενο αναφοράς επικύρωσης με success true και ErrorCode._2FaEnabled.

Εάν περάσει και αυτός ο έλεγχος, επιστρέφει το τελικό αντικείμενο αναφοράς επικύρωσης με report.Message = "User Verified",

report.Success = true,

return (report, user); (ο επιστρεφόμενος χρήστης).

Στη συνέχεια, η μέθοδος του ελεγκτή ελέγχει αν η επικύρωση επέστρεψε true ή false και στέλνει την κατάλληλη απάντηση.

Ειδική περίπτωση για τη λογική 2FA: ελέγχει το ErrorCode του αντικειμένου report και αν είναι ErrorCode._2FaEnabled, καλεί την αντίστοιχη μέθοδο της κλάσης service, η οποία δημιουργεί έναν μοναδικό κωδικό πρόσβασης χρησιμοποιώντας το OTP.Net, με το μυστικό κλειδί του χρήστη, το οποίο αποθηκεύτηκε στη βάση δεδομένων κατά την ενεργοποίηση του 2FA, και το στέλνει πίσω στη μέθοδο controller, η οποία στη συνέχεια καλεί τη μέθοδο της υπηρεσίας email για να στείλει τον κωδικό 2fa στο email του χρήστη μέσω του οποίου χρησιμοποιεί System.Net.Mail για την αποστολή του email.

Αν η ειδική περίπτωση περάσει (ο χρήστης δεν έχει ενεργοποιήσει το 2FA στο λογαριασμό του) καλεί τη μέθοδο της κλάσης service που δημιουργεί ένα JWT με βάση το χρήστη και το στέλνει πίσω, τέλος χρησιμοποιεί τη γενικευμένη κλάση ApiResponse για να κατασκευάσει μια απάντηση API και την στέλνει πίσω στο χρήστη.

5.8 Docker

Το Docker είναι μια πλατφόρμα διαχείρισης containers που επιτρέπει στους προγραμματιστές να πακετάρουν εφαρμογές και τις εξαρτήσεις τους σε απομονωμένες μονάδες που ονομάζονται containers. Αυτά τα container μπορούν να εκτελούνται με σταθερότητα σε διάφορα περιβάλλοντα, παρέχοντας μια ελαφριά και αποτελεσματική λύση για την ανάπτυξη εφαρμογών.

Το Docker χρησιμοποιεί την τεχνολογία Containerization για να ενθυλακώσει εφαρμογές και τις εξαρτήσεις τους, συμπεριλαμβανομένων βιβλιοθηκών, εργαλείων συστήματος και περιβαλλόντων χρόνου εκτέλεσης, σε ένα ενιαίο πακέτο. Αυτό το πακέτο, γνωστό ως container, είναι συμβατό και μπορεί να εκτελεστεί με συνέπεια σε οποιοδήποτε σύστημα που υποστηρίζει το Docker.

Στο πλαίσιο της εφαρμογής το Dockerfile δημιουργείται με σκοπό να επιτρέψει την εύκολη κατασκευή του έργου σε ένα Docker image και την εκτέλεσή του χρησιμοποιώντας αυτό το image. Είναι ένα βήμα προς την κατεύθυνση της ευκολίας της ανάπτυξης, της μεταφοράς και της εκτέλεσης της εφαρμογής σε διάφορα περιβάλλοντα, εξαλείφοντας τυχόν προβλήματα που μπορεί να προκύψουν λόγω διαφορών στο περιβάλλον του λειτουργικού συστήματος ή των εξαρτήσεων.

5.9 Επίλογος

Στη σφαίρα της ανάπτυξης εφαρμογών, το επίπεδο API αποτελεί μια ζωτικής σημασίας πύλη, συντονίζοντας την απρόσκοπτη αλληλεπίδραση μεταξύ των χρηστών και του υποκείμενου συστήματος. Αυτό το κεφάλαιο εμβάθυνε στις περιπλοκές του επιπέδου API, εξερευνώντας τα θεμελιώδη στοιχεία που διαμορφώνουν τη λειτουργικότητά του.

Η εφαρμογή ξεκινά με την ουσιαστική κλάση της Program.cs. Η διαμόρφωση των υπηρεσιών, η ρύθμιση του ελέγχου ταυτότητας και της εξουσιοδότησης, ο ορισμός των συνδέσεων βάσης δεδομένων

και η ενσωμάτωση του βασικού ενδιάμεσου λογισμικού θέτουν τις βάσεις για ένα ισχυρό και ασφαλές API. Ο κώδικας εντός της κλάσης λειτουργεί ως άξονας, διασφαλίζοντας ότι η εφαρμογή αρχικοποιείται με ακρίβεια και αποτελεσματικότητα.

Στη συνέχεια συζητήθηκαν οι ελεγκτές, οι φορείς συγκεκριμένων λειτουργιών εντός της εφαρμογής. Αυτοί οι ελεγκτές στεγάζουν μια πληθώρα μεθόδων, κάθε μία από τις οποίες είναι υπεύθυνη για το χειρισμό ξεχωριστών λειτουργιών - από την εγγραφή του χρήστη μέχρι την επεξεργασία συναλλαγών και όλα τα ενδιάμεσα. Οι ελεγκτές όχι μόνο διευκολύνουν τα αιτήματα των χρηστών αλλά και ενθυλακώνουν μέρος της επιχειρησιακής λογικής της εφαρμογής, προωθώντας μια αρμονική και συντηρήσιμη αρχιτεκτονική.

Οι σελίδες Razor, με την απλότητα και την ευελιξία τους, κάνουν μια μικρή εμφάνιση στο επίπεδο API, χρησιμεύοντας ως σημεία κατάληξης του UI για συγκεκριμένες λειτουργίες. Οι σελίδες, συμπεριλαμβανομένων των Ακύρωση, Επιτυχία και EmailVerification, βελτιώνουν την εμπειρία του χρήστη παρέχοντας ειδικές διεπαφές για τα αποτελέσματα της συναλλαγής και την κατάσταση επαλήθευσης email.

Το Swagger, το εργαλείο τεκμηρίωσης και εξερεύνησης, προσφέρει τη διαφάνεια στο επίπεδο API μας. Η διαμόρφωση του Swagger, σε συνδυασμό με ένα bearer token για έλεγχο ταυτότητας, διασφαλίζει ότι οι προγραμματιστές μπορούν να κατανοήσουν και να αλληλεπιδράσουν με ευκολία με τα τελικά σημεία API. Η προσθήκη ορισμών και απαιτήσεων ασφαλείας οχυρώνει το API, διασφαλίζοντάς το από μη εξουσιοδοτημένη πρόσβαση.

Μια αξιοσημείωτη βελτίωση στο επίπεδο API είναι η υλοποίηση της μετατροπής αριθμών enums σε συμβολοσειρές. Το φίλτρο EnumSchemaFilter, το οποίο έχει πλέον γενικευτεί για όλα τα enums, επαυξάνει την τεκμηρίωση Swagger παρουσιάζοντας τις τιμές των enum ως αναγνώσιμες από τον άνθρωπο συμβολοσειρές, βελτιώνοντας την αναγνωσιμότητα και την κατανόηση.

Ολοκληρώνοντας αυτό το κεφάλαιο, η συνέργεια μεταξύ των ρυθμίσεων εκκίνησης, των ελεγκτών, των σελίδων Razor και του Swagger καθιστά το επίπεδο API ένα ισχυρό και φιλικό προς το χρήστη περιβάλλον εργασίας. Αυτό το στρώμα όχι μόνο γεφυρώνει το χάσμα μεταξύ των χρηστών και του υποκείμενου συστήματος, αλλά χρησιμεύει επίσης ως απόδειξη του σχολαστικού σχεδιασμού και της προσεγμένης αρχιτεκτονικής της εφαρμογής μας. Το στρώμα API, ένα βασικό στοιχείο του οικοσυστήματος λογισμικού, είναι έτοιμο να δώσει τη δυνατότητα στους χρήστες να ενδυναμώσουν και να διευκολύνουν την απρόσκοπτη αλληλεπίδραση με τον πλούτο των λειτουργιών της εφαρμογής

Κεφάλαιο 6ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Συμπεράσματα:

Καθώς ολοκληρώνεται η διερεύνηση της αρχιτεκτονικής και των λειτουργιών της εφαρμογής, προκύπτουν πολύτιμες πληροφορίες για τα δυνατά σημεία και τους τομείς που χρήζουν βελτίωσης. Παρατηρείται ένα ισχυρό θεμέλιο στην καλά οργανωμένη δομή και τις εκτεταμένες λειτουργίες, που αναδεικνύουν τις μελετημένες σχεδιαστικές επιλογές. Η ενσωμάτωση τεχνολογιών όπως το Swagger και το Razor Pages εμπλουτίζει την εμπειρία του χρήστη, παρέχοντας ολοκληρωμένη τεκμηρίωση.

Προτάσεις για μελλοντικές βελτιώσεις:

Βελτίωση των Data Cleaners:

Οι καθαριστές δεδομένων, που είναι αναπόσπαστο στοιχείο για τη διατήρηση της ακεραιότητας των δεδομένων, μπορεί να επωφεληθούν από περαιτέρω βελτίωση. Η λεπτομερής εξέταση των διαδικασιών καθαρισμού και η εισαγωγή πιο εξελιγμένων αλγορίθμων θα μπορούσε να βελτιώσει τη συνολική ποιότητα των δεδομένων.

Ζωντανός διακομιστής Minio:

Ενώ η τρέχουσα εγκατάσταση του Minio εξυπηρετεί το σκοπό, θα μπορούσε να εξεταστεί η υλοποίηση ενός ζωντανού διακομιστή Minio. Αυτό θα επέτρεπε στους χρήστες να φιλοξενούν τα αρχεία τους ανεξάρτητα, παρέχοντας μεγαλύτερη ευελιξία.

Ενισχυμένη επικύρωση δεδομένων:

Η ενίσχυση των μηχανισμών επικύρωσης δεδομένων κατά τη διάρκεια των διαδικασιών καταχώρισης και επιμέλειας είναι ζωτικής σημασίας. Η εφαρμογή πρόσθετων ελέγχων και επικυρώσεων μπορεί να διασφαλίσει ότι μόνο ακριβή και αξιόπιστα δεδομένα εισέρχονται στο σύστημα.

Μηχανισμός προσωρινής αποθήκευσης πέραν της προσωρινής αποθήκευσης μνήμης:

Η εξερεύνηση και η ενσωμάτωση πιο προηγμένων μηχανισμών προσωρινής αποθήκευσης θα μπορούσε να βελτιστοποιήσει περαιτέρω τις επιδόσεις. Η χρήση εξωτερικών λύσεων προσωρινής αποθήκευσης ή η εφαρμογή προσαρμοσμένων στρατηγικών προσωρινής αποθήκευσης μπορεί να εξεταστεί για την αποτελεσματική ανάκτηση δεδομένων.

Σκέψεις επεκτασιμότητας:

Καθώς η εφαρμογή εξελίσσεται, η επεκτασιμότητα γίνεται κρίσιμος παράγοντας. Μελλοντικά μπορεί να εξεταστούν στρατηγικές που θα διασφαλίζουν ότι το σύστημα μπορεί να χειριστεί απρόσκοπτα

αυξημένα φορτία και αλληλεπιδράσεις χρηστών. Αυτό μπορεί να περιλαμβάνει βελτιστοποιήσεις σε ερωτήματα βάσεων δεδομένων, διαμορφώσεις διακομιστών και εξισορρόπηση φορτίου.

Λεπτομερής τεκμηρίωση:

Η πλήρης τεκμηρίωση είναι ζωτικής σημασίας για τη συντήρηση και την επέκταση της εφαρμογής. Η σαφής και ενημερωμένη τεκμηρίωση που περιγράφει την αρχιτεκτονική του συστήματος, τα API και τις βέλτιστες πρακτικές θα ωφελήσει σημαντικά τους μελλοντικούς συνεισφέροντες και προγραμματιστές.

Εν κατακλείδι, ενώ η εφαρμογή αποτελεί απόδειξη των σημερινών δυνατοτήτων της, ο δρόμος προς τη βελτίωση είναι διαρκής. Οι προτάσεις που εντοπίστηκαν χρησιμεύουν ως σκαλοπάτια για τους μελλοντικούς προγραμματιστές ώστε να βελτιώσουν τη λειτουργικότητα, τη συντηρησιμότητα και την εμπειρία του χρήστη της εφαρμογής. Με δέσμευση για συνεχή βελτίωση και συνεργασία με την κοινότητα, η εφαρμογή μπορεί να εξελιχθεί σε μια πιο ισχυρή και ευέλικτη λύση για τους χρήστες της.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Internet Site

[1] ASP.NET Core WebAPI. https://dotnettutorials.net/course/asp-net-core-web-api-tutorials

[2] ASP.NET Official Documentation. https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0

[3] Official Minio Page. https://min.io

[4] MinIO Client SDK for .NET Quickstart Quide. https://min.io/docs/minio/linux/developers/dotnet/minio-dotnet.html

[5] Twilio Service Documentation for ASP.NET Core. https://www.twilio.com/docs

[6] Stripe Documentation. https://stripe.com/docs

[7] Otp.NET Library Github. https://github.com/kspearrin/Otp.NET?search=1

[8] BCrypt.net Library Github. https://github.com/BcryptNet/bcrypt.net

[9] Useful C# Bootcamp: https://kenslearningcurve.com/bundles/c-bootcamp-your-one-stop-resource-for-learning-the-language/

[10] Docker Documentation: https://docs.docker.com/docker-hub/