Πανεπιστήμιο Αιγαίου
Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

321-4002 – Τεχνολογία Λογισμικού

Διδάσκων: Κυριάκος Κρητικός

Project: Festival Managment System

Εργαστηριακός Συνεργάτης: Αλέξανδρος Φακής

321/2019157 Στυλιανός Νικολόπουλος
321/2020010 Γεώργιος Αναγνωστόπουλος
321/2020077 Γεώργιος Καμτσικλής

Σάμος, 17 Σεπτέμβριου 2025

Πίνακας περιεχομένων

1 Περίληψη

Η παρούσα εργασία έχει ως αντικείμενο την ανάπτυξη ενός πληροφοριακού συστήματος για τη διαχείριση μουσικών φεστιβάλ. Στόχος είναι ο σχεδιασμός και η υλοποίηση ενός backend συστήματος που θα παρέχει RESTful υπηρεσίες για τη διαχείριση χρηστών, φεστιβάλ και μουσικών εμφανίσεων, αξιοποιώντας τεχνολογίες που υποστηρίζουν συνεργατική ανάπτυξη λογισμικού, αυτοματοποίηση, έλεγχο εκδόσεων και μονάδων.

Το έργο επικεντρώνεται στη δημιουργία ενός backend πληροφοριακού συστήματος που θα υποστηρίζει την πλήρη ροή διαδικασιών, από τη δημιουργία και οργάνωση ενός φεστιβάλ έως την υποβολή, αξιολόγηση και τελική αποδοχή ή απόρριψη μουσικών εμφανίσεων. Μέσω της υλοποίησης RESTful υπηρεσιών και της αξιοποίησης βάσης δεδομένων, επιδιώκεται να προσφερθεί μια αξιόπιστη, ασφαλής και επεκτάσιμη λύση, η οποία θα διασφαλίζει τη διαφάνεια των ρόλων, την ακεραιότητα των δεδομένων και την ταχύτητα εκτέλεσης των λειτουργιών.

Το σύστημα βασίζεται σε RESTful web services που αλληλεπιδρούν με υποκείμενη βάση δεδομένων για την αποθήκευση και διαχείριση όλων των σχετικών οντοτήτων, όπως χρήστες, φεστιβάλ και εμφανίσεις (performances). Υποστηρίζει διαφορετικούς ρόλους χρηστών (Visitor, User, Artist, Organizer, Staff, Admin), στους οποίους αντιστοιχούν συγκεκριμένα δικαιώματα και λειτουργίες. Η λειτουργικότητα περιλαμβάνει, μεταξύ άλλων, τη δημιουργία και ενημέρωση φεστιβάλ, την υποβολή και αξιολόγηση εμφανίσεων, την ανάθεση ρόλων και τη διαχείριση χρηστών, ακολουθώντας προκαθορισμένες καταστάσεις (states) για κάθε φεστιβάλ και εμφάνιση. Με τον τρόπο αυτό διασφαλίζεται η ορθή ροή εργασιών, η συνεργασία μεταξύ διαφορετικών εμπλεκόμενων ρόλων και η αξιοπιστία της πληροφορίας.

2 Οργάνωση

2.1 Ρόλοι ομάδας

Η εργασία υλοποιήθηκε από τρία μέλη ομάδας, καθένα από τα οποία ανέλαβε έναν κύριο ρόλο, ενώ παράλληλα συνέβαλε και στα υπόλοιπα στάδια της ανάπτυξης ώστε να εξασφαλιστεί η ομαλή συνεργασία και η συνολική κατανόηση του έργου από όλα τα μέλη.

- **Μέλος Α:** Είχε ως βασική ευθύνη τη συλλογή και ανάλυση των απαιτήσεων του συστήματος, την τεκμηρίωση των λειτουργικών και μη λειτουργικών απαιτήσεων, καθώς και τη διαμόρφωση της σχετικής ενότητας στην αναφορά. Παράλληλα, συμμετείχε στον σχεδιασμό των use case διαγραμμάτων και συνέβαλε στην υλοποίηση μικρών τμημάτων κώδικα.
- **Μέλος Β:** Ανέλαβε κυρίως τον σχεδιασμό του συστήματος και την παραγωγή των βασικών διαγραμμάτων (use case, component, activity, sequence, ER), καθώς και την τεκμηρίωσή τους στην αναφορά. Επιπλέον, βοήθησε στη συλλογή απαιτήσεων, συμμετείχε στη συγγραφή τμημάτων του κώδικα backend και συνέβαλε στις δοκιμές του συστήματος.
- **Μέλος Γ:** Είχε ως κύριο ρόλο την υλοποίηση του backend συστήματος, την ανάπτυξη των RESTful υπηρεσιών, τη διασύνδεση με τη βάση δεδομένων και την ανάπτυξη unit tests. Παράλληλα, συνέβαλε στον σχεδιασμό της βάσης δεδομένων, στη δημιουργία της τεκμηρίωσης υλοποίησης και στη συγγραφή της τελικής αναφοράς.

Και τα τρία μέλη συνεργάστηκαν ενεργά σε όλα τα στάδια του έργου, από τη συλλογή απαιτήσεων έως τις δοκιμές και τη συγγραφή της αναφοράς, διατηρώντας κοινή παρουσία στο GitHub repository και φροντίζοντας για τη συνεχή ανασκόπηση και βελτίωση του κώδικα και της τεκμηρίωσης.

- Μέλος Α → Γεώργιος Αναγνωστόπουλος
- Μέλος Β → Γεώργιος Καμτσικλής
- Μέλος Γ → Στυλιανός Νικολόπουλος

2.2 Στάδια Ανάπτυξης

Προκειμένου να ολοκληρωθεί και να είναι λειτουργικό το έργο ακολουθήθηκαν τα παρακάτω στάδια ανάπτυξης:

1. **Ανάλυση Απαιτήσεων (15%)** – Καταγραφή και οργάνωση λειτουργικών και μη λειτουργικών απαιτήσεων.
2. **Σχεδιασμός (30%)** – Δημιουργία διαγραμμάτων για την αρχιτεκτονική, τις χρήσεις και τη συμπεριφορά του συστήματος, καθώς και για τη βάση δεδομένων.
3. **Υλοποίηση (50%)** – Ανάπτυξη του backend με RESTful υπηρεσίες, χρήση GitHub για συνεργασία και υλοποίηση unit tests.
4. **Συγγραφή Αναφοράς (5%)** – Ενοποίηση παραδοτέων και τεκμηρίωση της εργασίας.

2.3 Χρονοδιάγραμμα και Διάρκεια

Η συνολική διάρκεια ανάπτυξης της εργασίας εκτιμάται σε 252.8 ώρες. Η κατανομή του χρόνου στα στάδια ήταν περίπου η εξής:

- Ανάλυση απαιτήσεων: ~ 3 ημέρες
- Σχεδιασμός: ~ 7 ημέρες
- Υλοποίηση: ~ 20 ημέρες
- Αναφορά: ~ 1.6 ημέρες

3 Απαιτήσεις Συστήματος

Στην ενότητα αυτή παρουσιάζονται οι απαιτήσεις του συστήματος, οι οποίες καθορίζουν τις λειτουργίες που πρέπει να υποστηρίζει, καθώς και τα χαρακτηριστικά που διασφαλίζουν την ορθή και αξιόπιστη λειτουργία του. Οι απαιτήσεις διαχωρίζονται σε λειτουργικές, που αφορούν τις δυνατότητες και τις ενέργειες των χρηστών και του συστήματος, και σε μη λειτουργικές, που καθορίζουν την απόδοση, την ασφάλεια και τη δομή του συστήματος.

3.1 Λειτουργικές Απαιτήσεις

I. Διαχείριση χρηστών

1. **Εγγραφή Χρήστη (Register User):** Ένας μη εγγεγραμμένος χρήστης έχει το δικαίωμα δημιουργίας νέου λογαριασμού χρήστη. Ο πρώτος χρήστης που καταχωρείται γίνεται αυτόματα ADMIN, ενώ όλοι οι επόμενοι είναι ανενεργοί μέχρι να ενεργοποιηθούν από τον διαχειριστή.

Λειτουργίες:

- Επικύρωση username (regex).
- Επικύρωση password (μήκος, κεφαλαία, μικρά, αριθμός, ειδικός χαρακτήρας).
- Hashing password πριν αποθηκευτεί στη βάση.
- Αποθήκευση του χρήστη στη βάση.

2. **Σύνδεση Χρήστη (Login User):** Ένας χρήστης έχει την δυνατότητα να συνδεθεί στο σύστημα με username και password και να λάβει authentication token.

Λειτουργίες:

- Έλεγχος ύπαρξης χρήστη και ενεργού λογαριασμού.
- Έλεγχος σωστού κωδικού και αύξηση αποτυχημένων προσπαθειών.
- Απενεργοποίηση λογαριασμού μετά από 3 αποτυχημένες προσπάθειες.
- Γεννήτρια token με χρόνο λήξης.

3. **Αποσύνδεση Χρήστη (Logout User):** Ο χρήστης μπορεί να αποσυνδεθεί, απενεργοποιώντας όλα τα ενεργά token του.

Λειτουργίες:

- Απενεργοποίηση όλων των session tokens του χρήστη.

4. **Ενημέρωση Στοιχείων Χρήστη (Update User Info):** Ο χρήστης έχει την δυνατότητα να αλλάξει το πλήρες όνομα του ή το username. Οι admins μπορούν να ενημερώσουν άλλους χρήστες.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας χρήστη μέσω token.
- Αλλαγή full name ή username.

- Έλεγχος διαθεσιμότητας νέου username.
- Επανέκδοση token αν αλλάξει username.

5. **Αλλαγή Κωδικού Χρήστη (Update User Password):** Ο χρήστης μπορεί να αλλάξει τον κωδικό του, με επικύρωση του παλιού κωδικού και κανόνες ισχυρού password.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος παλιού κωδικού.
- Απενεργοποίηση λογαριασμού μετά από 3 αποτυχημένες προσπάθειες αλλαγής κωδικού.
- Hashing του νέου κωδικού και αποθήκευση.
- Επανέκδοση token.

6. **Διαχείριση Κατάστασης Λογαριασμού (Update Account Status):** Ο admin επιτρέπεται να ενεργοποιεί ή να απενεργοποιεί λογαριασμούς χρηστών.

Λειτουργίες:

- Επιβεβαίωση δικαιωμάτων admin.
- Αλλαγή κατάστασης χρήστη (active/inactive).
- Απενεργοποίηση token όταν ένας λογαριασμός απενεργοποιείται.

7. **Διαγραφή Χρήστη (Delete User):** Στον χρήστη επιτρέπεται να διαγράψει το δικό του λογαριασμό ή σε έναν admin να διαγράψει άλλους χρήστες.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος δικαιωμάτων admin για διαγραφή άλλου χρήστη.
- Διαγραφή όλων των tokens του χρήστη.
- Αφαίρεση του χρήστη από τη βάση.

II. Διαχείριση φεστιβάλ

8. **Δημιουργία Φεστιβάλ (Create Festival):** Ένας εξουσιοδοτημένος χρήστης μπορεί να δημιουργήσει ένα νέο φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση βασικών στοιχείων (όνομα, περιγραφή, ημερομηνίες, τοποθεσία).
- Αποθήκευση νέου φεστιβάλ στη βάση δεδομένων.

9. **Ενημέρωση Στοιχείων Φεστιβάλ (Update Festival Info):** Ένας εξουσιοδοτημένος χρήστης επιτρέπεται να ενημερώσει τις πληροφορίες ενός υπάρχοντος φεστιβάλ**.**

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή βασικών στοιχείων (όνομα, περιγραφή, ημερομηνίες).
- Διαχείριση Venue Layout (σκηνές, vendor areas, εγκαταστάσεις).
- Διαχείριση Budget (έσοδα, κόστη, logistics).
- Διαχείριση Vendor Management (food stalls, booths).
- Ενημέρωση λίστας διοργανωτών και προσωπικού.

10. **Διαγραφή Φεστιβάλ (Delete Festival):** Ένας εξουσιοδοτημένο χρήστης δύναται να διαγράψει ένα φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος δικαιωμάτων (admin/organizer).
- Αφαίρεση φεστιβάλ από τη βάση.

11. **Αναζήτηση Φεστιβάλ (Search Festival):** Οι χρήστες και οι επισκέπτες μπορούν να αναζητούν φεστιβάλ με βάση κριτήρια.

Λειτουργίες:

- Αναζήτηση βάσει ονόματος, περιγραφής, ημερομηνιών και τοποθεσίας.
- Επιστροφή αποτελεσμάτων με συνοπτικά στοιχεία (id, name, description, dates, venue).

12. **Προβολή Φεστιβάλ (View Festival):** Οι χρήστες και οι επισκέπτες επιτρέπεται να δουν λεπτομέρειες ενός φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας (προαιρετική για επισκέπτες).
- Εμφάνιση όλων των βασικών πληροφοριών για το φεστιβάλ.

13. **Προσθήκη Διοργανωτών (Add Organizers):** Στον διοργανωτή του συγκεκριμένου festival επιτρέπεται η προσθήκη χρηστών ως συνδιοργάνωνες.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος δικαιωμάτων.
- Ενημέρωση λίστας διοργανωτών.

14. **Προσθήκη Προσωπικού (Add Staff):** Οι διοργανωτές του συγκεκριμένου festival έχουν την δυνατότητα προσθήκης χρηστών στη λίστα προσωπικού του φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος δικαιωμάτων.
- Ενημέρωση λίστας προσωπικού.

15. **Έναρξη Υποβολών (Submission Start):** Ο διοργανωτής θέτει την έναρξη της περιόδου υποβολής συμμετοχών.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε “Submission Open”.

16. **Έναρξη Ανάθεσης Stage Managers (Stage Manager Assignment Start):** Ο διοργανωτής ενεργοποιεί την διαδικασία ανάθεσης stage managers.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε “Stage Manager Assignment”.

17. **Έναρξη Αξιολόγησης (Review Start):** Ο διοργανωτής επιτρέπει την έναρξη της διαδικασίας αξιολόγησης συμμετοχών.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε “Review Phase”.

18. **Έναρξη Δημιουργίας Προγράμματος (Schedule Making):** Ο διοργανωτής επιτρέπει την έναρξη δημιουργίας του προγράμματος του φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε “Schedule Making”.

19. **Έναρξη Τελικής Υποβολής (Final Submission Start):** Επιτρέπει την έναρξη τελικής υποβολής συμμετοχών.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε “Final Submission”.

20. **Διαδικασία Λήψης Απόφασης (Decision Making):** Ο διοργανωτής επιτρέπει την έναρξη και καταγραφή της τελικής απόφασης για το φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε "Decision Making".

21. **Ανακοίνωση Φεστιβάλ (Festival Announcement):** Ο διοργανωτής επιτρέπει την επίσημη ανακοίνωση του φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Αλλαγή κατάστασης φεστιβάλ σε "Announced".

III. Διαχείριση εμφανίσεων (Performances)

22. **Προσθήκη Μέλους Μπάντας (Add Band Member):** Ένας κύριος καλλιτέχνης ή admin μπορεί να προσθέσει νέο μέλος σε ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι ο χρήστης είναι κύριος καλλιτέχνης ή admin.
- Έλεγχος ότι το νέο μέλος υπάρχει στο σύστημα και δεν είναι ήδη μέλος της μπάντας.
- Προσθήκη του νέου μέλους στο performance.

23. **Διαχείριση Merchandise (Merchandise Management)**: Ένας εξουσιοδοτημένος χρήστης δύναται να προσθέσει, επεξεργαστεί ή διαγράψει αντικείμενα merchandise για ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Προσθήκη αντικειμένων με name, type, price και προαιρετικό description.
- Ενημέρωση ή διαγραφή υπάρχοντων αντικειμένων.
- Εμφάνιση λίστας αντικειμένων για το performance.

24. **Έγκριση Performance (Approve Performance)**: Ο διοργανωτής του festival μπορεί να εγκρίνει ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι ο χρήστης είναι οργανωτής.
- Έγκριση του performance και εμφάνιση του στο πρόγραμμα του festival.

25. **Ανάθεση Staff (Assign Staff)**: Ο admin μπορεί να αναθέσει staff σε ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι ο χρήστης είναι admin.
- Ανάθεση του staff στο performance.
- Ειδοποίηση του staff για την ανάθεση.

26. **Δημιουργία Performance (Create Performance)**: Ένας καλλιτέχνης επιτρέπεται να δημιουργήσει νέο performance σε ένα festival.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση υποχρεωτικών πεδίων: festivalId, name, description, genre, duration, bandMemberIds.
- Προαιρετική καταχώρηση: technicalRequirements, setlist, merchandiseItems, preferredRehearsalTimes, preferredPerformanceSlots.
- Έλεγχος μοναδικότητας ονόματος performance στο festival.
- Έλεγχος ύπαρξης των band members στο σύστημα.

27. **Οριστική Υποβολή Performance (Final Submission)**: Ο καλλιτέχνης μπορεί να οριστικοποιήσει το performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση τελικής setlist, rehearsalTimes και performanceTimeSlots.
- Απαγορεύεται η αλλαγή χωρίς έγκριση οργανωτή μετά την υποβολή.

28. **Απόρριψη Performance (Reject Performance)**: Ο διοργανωτής μπορεί να απορρίψει ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση rejectionReason (υποχρεωτικό).
- Απόκρυψη του performance από το πρόγραμμα.

29. **Αξιολόγηση Performance (Review Performance)**: Το προσωπικό δύναται να αξιολογήσει ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση score και reviewerComments.
- Σύνδεση αξιολόγησης με το performance πριν την τελική έγκριση.

30. **Υποβολή Performance (Submit Performance)**: Ο καλλιτέχνης μπορεί να κάνει απλή υποβολή ενός performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Σύνδεση υποβολής με το συγκεκριμένο performanceId.

31. **Ενημέρωση Performance (Update Performance)**: Ο καλλιτέχνης επιτρέπεται να ενημερώσει στοιχεία του performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Ενημέρωση πεδίων: name, description, genre, duration, bandMemberIds, technicalRequirements, setlist, merchandiseItems, preferredRehearsalTimes, preferredPerformanceSlots.
- Έλεγχος ότι οι αλλαγές γίνονται σύμφωνα με τους κανόνες έγκρισης του οργανωτή.

32. **Απόσυρση Performance (Withdraw Performance)**: Ο καλλιτέχνης μπορεί να αποσύρει το performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Απόκρυψη του performance από το πρόγραμμα.
- Σύνδεση απόσυρσης με το performanceId.

33. **Τεχνικές Απαιτήσεις (Technical Requirements Upload)**: Ο καλλιτέχνης επιτρέπεται να ανεβάσει αρχεία τεχνικών απαιτήσεων.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Ανέβασμα αρχείου (fileName).
- Σύνδεση αρχείου με το performance για χρήση από το staff.

3.2 Μη Λειτουργικές Απαιτήσεις

i. Απόδοση και αξιοπιστία

- Κάθε αίτημα να εκτελείται σε 5–10 δευτερόλεπτα.
- Το σύστημα να είναι αξιόπιστο, χωρίς εσωτερικά σφάλματα.
- Οι αλλαγές στη βάση να είναι συνεπείς (transactional).

ii. Διαχείριση σφαλμάτων και ασφάλεια

- Το σύστημα επιστρέφει κατάλληλα μηνύματα λάθους για λανθασμένες εισόδους.
- Το σύστημα απενεργοποιεί τον λογαριασμό μετά από 3 αποτυχημένα login ή password update.
- Error messages για invalid ή expired tokens.
- Απενεργοποίηση λογαριασμών σε περίπτωση σύγκρουσης tokens.
- Ακύρωση tokens όταν αλλάζουν στοιχεία χρήστη ή password.

iii. Πρόσβαση και εξουσιοδότηση

- Το σύστημα να δέχεται μόνο authenticated χρήστες με σωστά roles.
- Κάθε λειτουργία να εκτελείται μόνο αν ο χρήστης έχει τα κατάλληλα δικαιώματα.

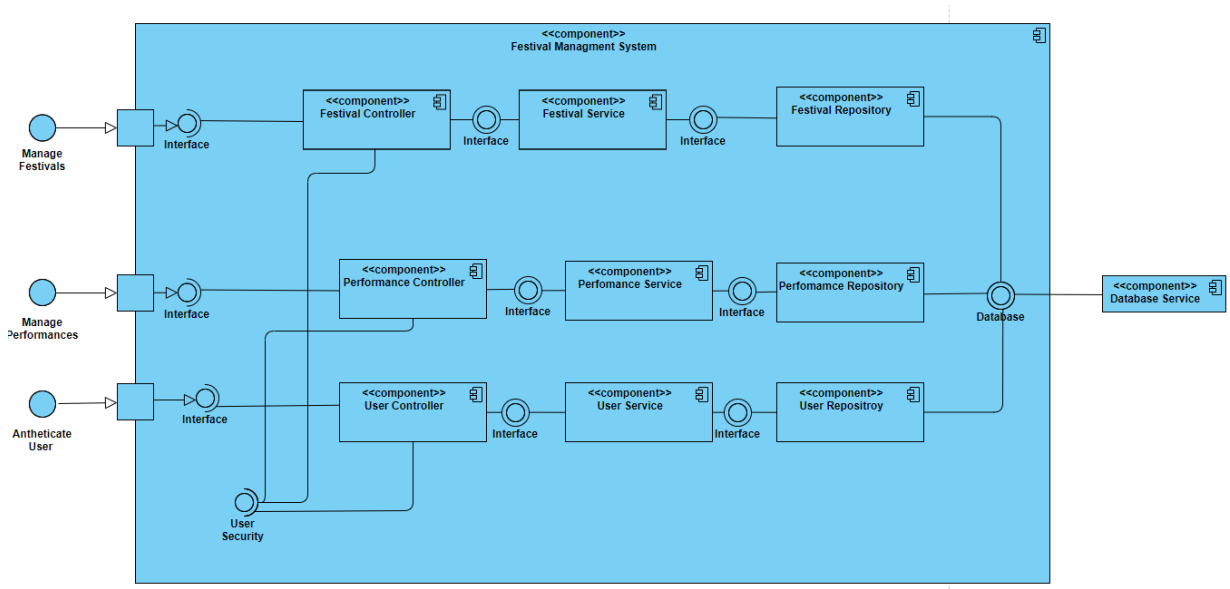
iv. Σχεδίαση και δομή συστήματος

- Το σύστημα διασφαλίζει την ασφαλή διαχείριση passwords (pattern check, strong password rules, hashed).
- Modular design: separation of concerns (Controllers, Services, DTOS, DAO).
- Επαλήθευση των tokens των χρηστών σε κάθε αίτημα.

4 Σχεδιασμός Συστήματος

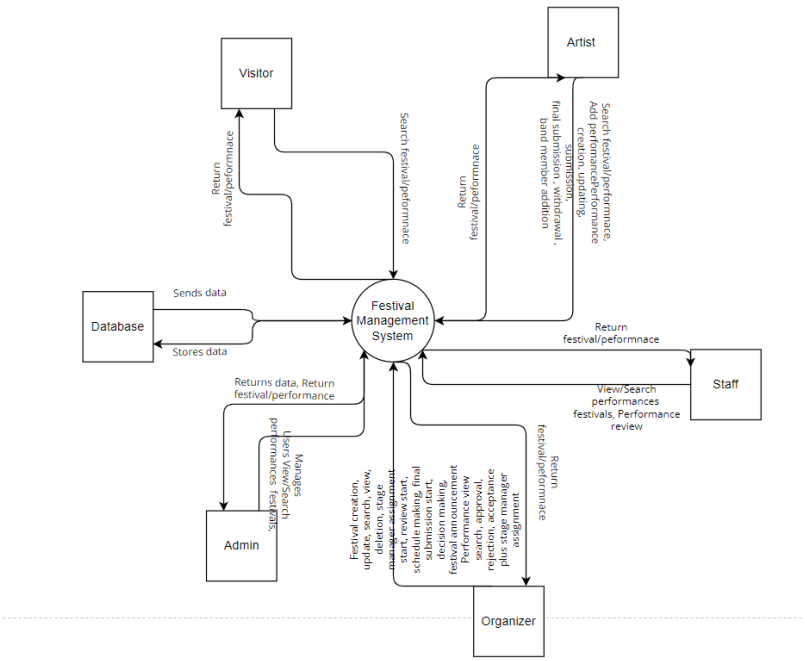
4.1 Αρχιτεκτονική και Πλαίσιο Συστήματος

Component Diagram



Το διάγραμμα παρουσιάζει την αρχιτεκτονική του Festival Management System. Το σύστημα αποτελείται από τρεις βασικές λειτουργικές ενότητες: διαχείριση φεστιβάλ, διαχείριση παραστάσεων και διαχείριση χρηστών.
Κάθε ενότητα περιλαμβάνει έναν Controller, έναν Service και έναν Repository που επικοινωνούν μεταξύ τους ακολουθώντας την αρχιτεκτονική τριών επιπέδων. Οι Controllers δέχονται αιτήματα από τα εξωτερικά interfaces, τα Services υλοποιούν την επιχειρησιακή λογική, ενώ τα Repositories συνδέονται με τη βάση δεδομένων για την αποθήκευση και ανάκτηση πληροφοριών.
Το Database Service υποστηρίζει όλες τις ενότητες, εξασφαλίζοντας ενιαία και ασφαλή πρόσβαση στη βάση δεδομένων. Επιπλέον, υπάρχει ξεχωριστό interface για τον έλεγχο ασφάλειας των χρηστών (User Security). Το διάγραμμα καταδεικνύει την οργάνωση του συστήματος σε ανεξάρτητα αλλά συνεργαζόμενα components, διευκολύνοντας την επεκτασιμότητα και τη συντήρηση.

Context Diagramm



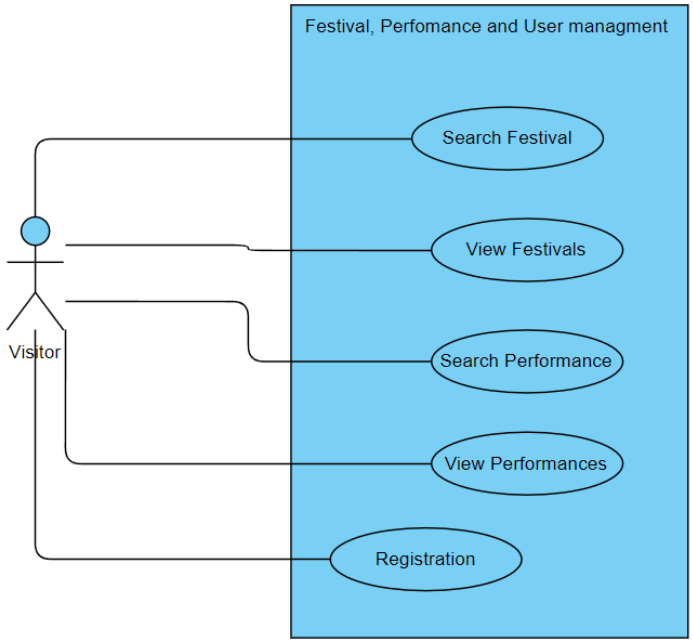
Το διάγραμμα αυτό απεικονίζει το Festival Management System σε επίπεδο context, αναδεικνύοντας τις σχέσεις του με τους εξωτερικούς χρήστες και συστήματα. Οι βασικοί actors είναι:

- Visitor: αναζητά και λαμβάνει πληροφορίες για φεστιβάλ και παραστάσεις.
- Artist: καταχωρεί στοιχεία, ενημερώνει το προφίλ του και συμμετέχει σε προγραμματισμένες παραστάσεις.
- Organizer: δημιουργεί φεστιβάλ, διαχειρίζεται παραστάσεις, εγκρίνει συμμετοχές και καθορίζει το τελικό πρόγραμμα.
- Staff: αναζητά φεστιβάλ/παραστάσεις και παρέχει αξιολογήσεις.
- Admin: έχει πλήρη δικαιώματα διαχείρισης (δημιουργία, ενημέρωση, διαγραφή, έλεγχος χρηστών και ρόλων).
- Database: αποθηκεύει και παρέχει δεδομένα προς το σύστημα.

Οι ροές δεδομένων μεταξύ των actors και του κεντρικού συστήματος περιγράφουν τις κύριες λειτουργίες, από την αναζήτηση και επιστροφή πληροφοριών έως την πλήρη διαχείριση φεστιβάλ και παραστάσεων. Το διάγραμμα τονίζει την κεντρική θέση του Festival Management System ως ενδιάμεσο φορέα επικοινωνίας μεταξύ όλων των ενδιαφερόμενων μερών.

4.2 Περιπτώσεις Χρήσης

Use Case Diagram - Visitor



Περιγραφή διαγράμματος:

Το διάγραμμα απεικονίζει τις βασικές λειτουργίες που έχει στη διάθεσή του ο επισκέπτης (Visitor) του συστήματος.

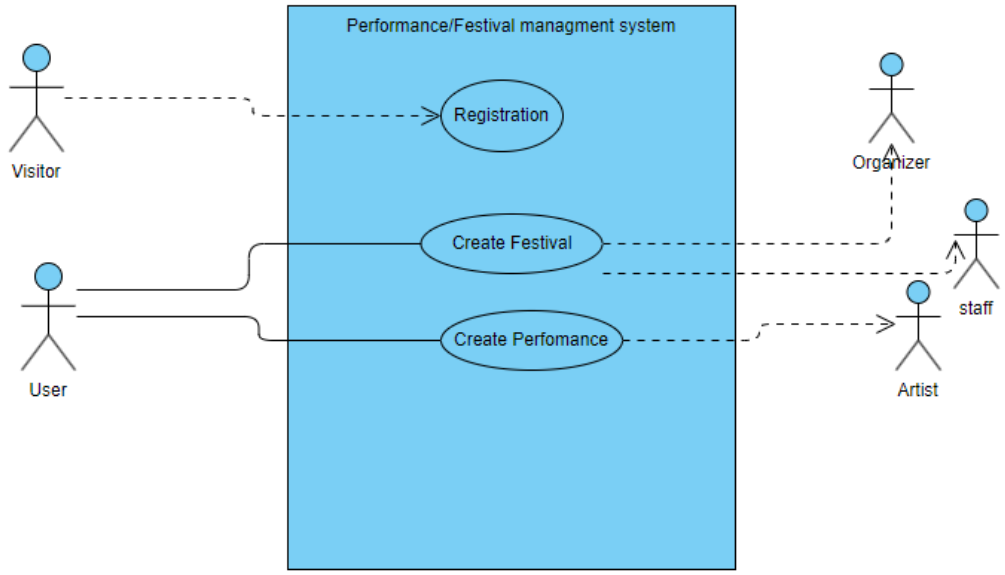
Συμμετέχοντες:

Visitor (Επισκέπτης): Ο βασικός actor που αλληλεπιδρά με το σύστημα, με στόχο την αναζήτηση και προβολή πληροφοριών για φεστιβάλ και παραστάσεις.

Χρήσεις (Use Cases):

1. Search performance: Αναζήτηση παραστάσεων.
2. View performance: Προβολή λεπτομερειών για μια παράσταση.
3. Search festival: Αναζήτηση φεστιβάλ.
4. View festival: Προβολή πληροφοριών για ένα φεστιβάλ.

Use Case Diagram – User



Περιγραφή διαγράμματος:

Το διάγραμμα παρουσιάζει τις βασικές λειτουργίες που μπορεί να εκτελέσει ένας γενικός χρήστης (User). Ο χρήστης έχει τη δυνατότητα να δημιουργεί φεστιβάλ και παραστάσεις, ενώ οι ρόλοι Organizer, Artist και Staff συνδέονται με αυτές τις διαδικασίες. Επίσης, μέσω του ρόλου Visitor υποστηρίζεται η εγγραφή στο σύστημα.

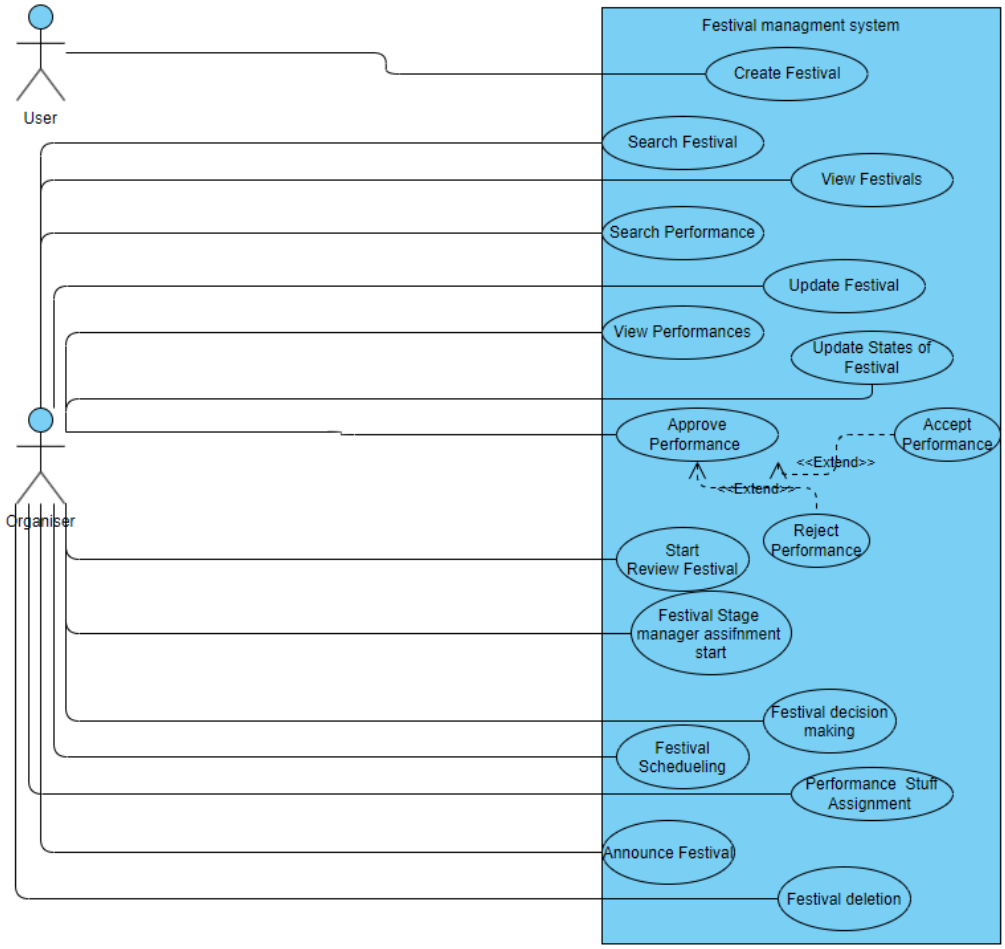
Συμμετέχοντες:

- User (Χρήστης): Δημιουργεί φεστιβάλ και παραστάσεις.
- Visitor (Επισκέπτης): Μπορεί να πραγματοποιήσει εγγραφή (Registration).
- Organizer (Διοργανωτής): Συνδέεται με τη δημιουργία φεστιβάλ.
- Artist (Καλλιτέχνης): Συνδέεται με τη δημιουργία παραστάσεων.
- Staff (Προσωπικό): Συνδέεται επίσης με τη δημιουργία παραστάσεων.

Χρήσεις (Use Cases):

1. Registration: Επιτρέπει σε έναν επισκέπτη να εγγραφεί στο σύστημα.
2. Create Festival: Ο χρήστης δημιουργεί ένα νέο φεστιβάλ, το οποίο σχετίζεται με τον διοργανωτή.
3. Create Performance: Ο χρήστης δημιουργεί μια νέα παράσταση, που σχετίζεται με καλλιτέχνες και προσωπικό.

Use Case Diagram - Organizer



Περιγραφή διαγράμματος:
Το διάγραμμα παρουσιάζει τις λειτουργίες του διοργανωτή (Organizer), ο οποίος έχει τον πλήρη έλεγχο των στοιχείων ενός φεστιβάλ και των συμμετοχών.

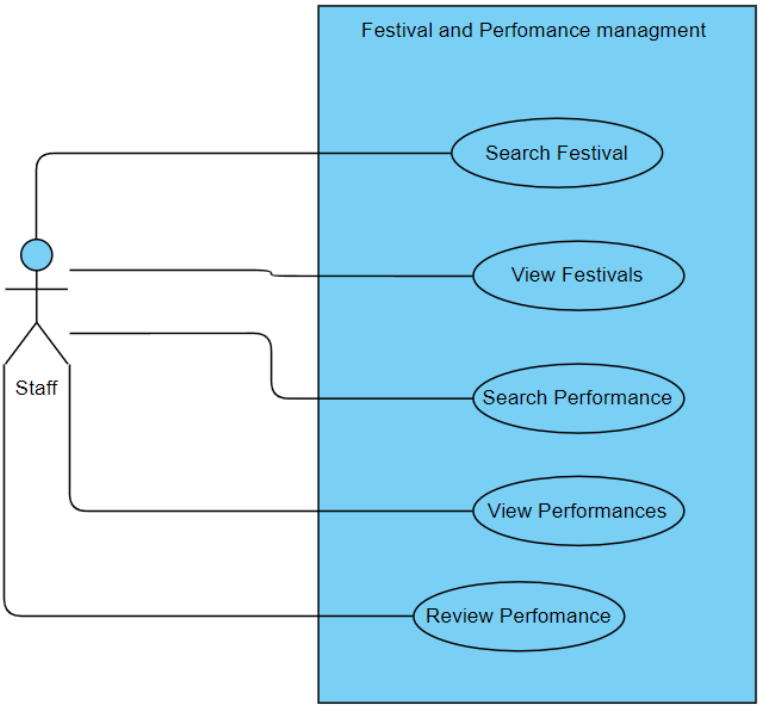
Συμμετέχοντες:

Organizer (Διοργανωτής): Actor που οργανώνει και διαχειρίζεται φεστιβάλ και performances.

Χρήσεις (Use Cases):

1. Manage festival: Διαχείριση όλων των στοιχείων ενός φεστιβάλ.
2. Approve performance: Έγκριση performance που υποβάλλουν οι καλλιτέχνες.
3. Reject performance: Απόρριψη performance που δεν πληροί τα κριτήρια.
4. Assign staff: Ανάθεση ρόλων/καθηκόντων στο προσωπικό (Staff).

Use Case Diagram – Staff



Περιγραφή διαγράμματος:
Το διάγραμμα απεικονίζει τις λειτουργίες του προσωπικού (Staff) μέσα στο σύστημα, κυρίως σε σχέση με τη διαχείριση των φεστιβάλ και των παραστάσεων.

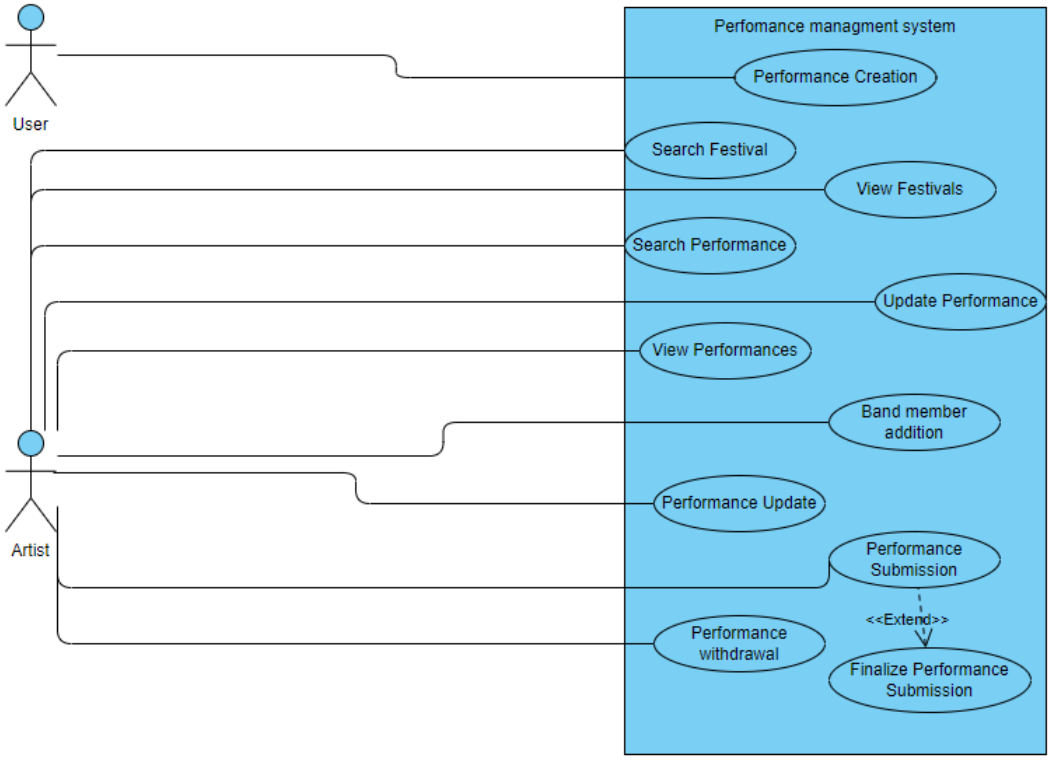
Συμμετέχοντες:

Staff (Προσωπικό): Actor που έχει αρμοδιότητες σχετικά με την οργάνωση και τροποποίηση πληροφοριών των φεστιβάλ.

Χρήσεις (Use Cases):

1. Search festival / Performance: Αναζήτηση φεστιβάλ / παραστάσεων στο σύστημα.
2. View festival / Performance: Προβολή φεστιβάλ / παραστάσεων.
3. Review festival / Performance: Αξιολόγηση φεστιβάλ / παραστάσεων.

Use Case Diagram - Artist



Περιγραφή διαγράμματος:
Το διάγραμμα παρουσιάζει τις λειτουργίες που μπορεί να εκτελέσει ο καλλιτέχνης (Artist) μέσα στο σύστημα, με επίκεντρο τη διαχείριση των performances που συμμετέχει.

Συμμετέχοντες:

Artist (Καλλιτέχνης): Actor που αλληλεπιδρά με το σύστημα προκειμένου να δημιουργήσει, να επεξεργαστεί ή να αποσύρει τα performances του.

Χρήσεις (Use Cases):

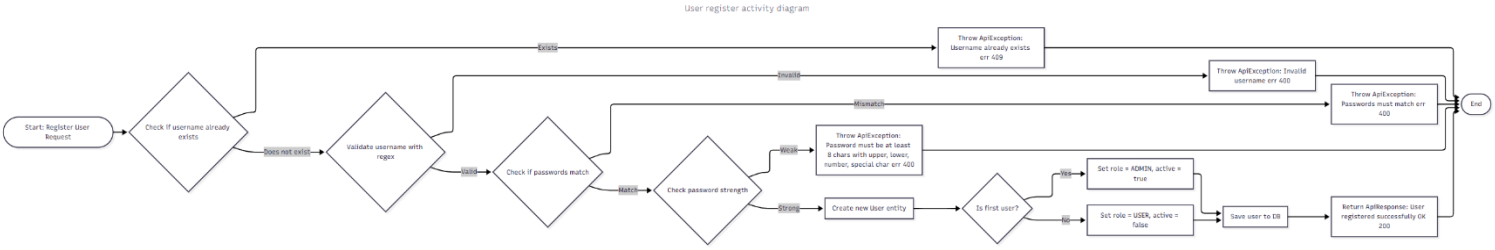
1. Create performance: Δημιουργία νέας παράστασης.
2. Update performance: Ενημέρωση/τροποποίηση στοιχείων μιας υπάρχουσας παράστασης.
3. Withdraw performance: Απόσυρση μιας παράστασης από το πρόγραμμα.
4. Add Band member: Προσθήκη μελών στην μπάντα
5. Search festival / Performance: Αναζήτηση φεστιβάλ / παραστάσεων στο σύστημα.
6. View festival / Performance: Προβολή φεστιβάλ / παραστάσεων.

4.3 Συμπεριφορά Συστήματος

4.3.1 Activity Diagramms

Τα Διαγράμματα Δραστηριότητας (Activity Diagram), απεικονίζουν τη ροή εργασιών ή ενεργειών μέσα σε μια διαδικασία του συστήματος. Χρησιμοποιούνται για να δείξουν τα βήματα, τους ελέγχους και τις εναλλακτικές πορείες εκτέλεσης. Αυτό τα καθιστά ιδανικά για την κατανόηση και τεκμηρίωση επιχειρησιακών διαδικασιών ή λειτουργιών.

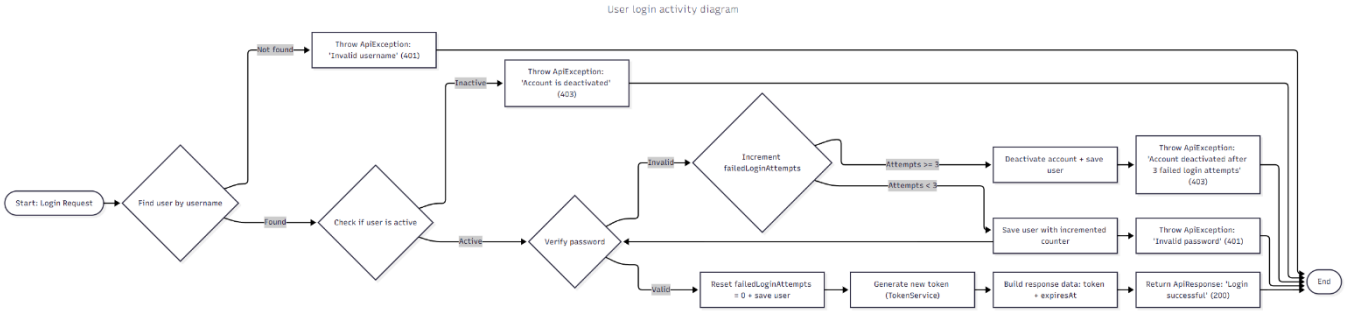
Activity Diagram – registerUser:



Το διάγραμμα δείχνει τη διαδικασία εγγραφής ενός νέου χρήστη. Αρχικά ελέγχεται εάν υπάρχει ήδη το username. Έπειτα γίνεται έλεγχος μορφής username και εγκυρότητας/ισότητας των passwords. Αν όλα είναι σωστά, δημιουργείται ο χρήστης. Ο πρώτος χρήστης γίνεται Admin και ενεργός, ενώ οι υπόλοιποι χρήστες γίνονται User και ανενεργοί. Τέλος αποθηκεύεται ο νέος χρήστης στην βάση δεδομένων και επιστρέφεται μήνυμα επιτυχίας.

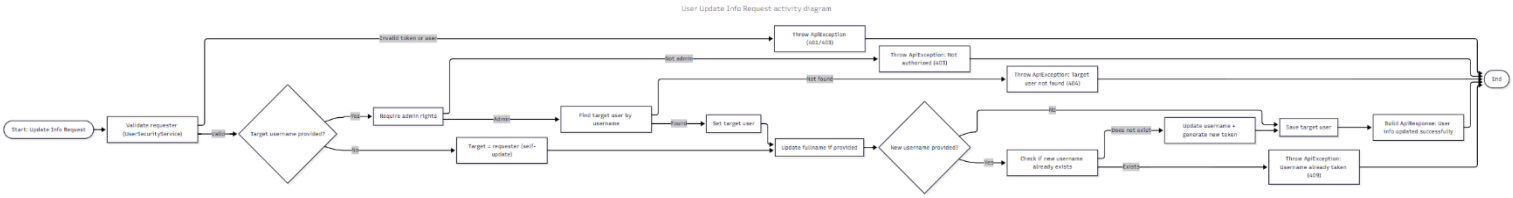
Σε περίπτωση εσφαλμένου username ή password ή ακόμη και λάθους format καλείται η εξαίρεση και η διαδικασία τερματίζεται.

Activity Diagram – userlogin:



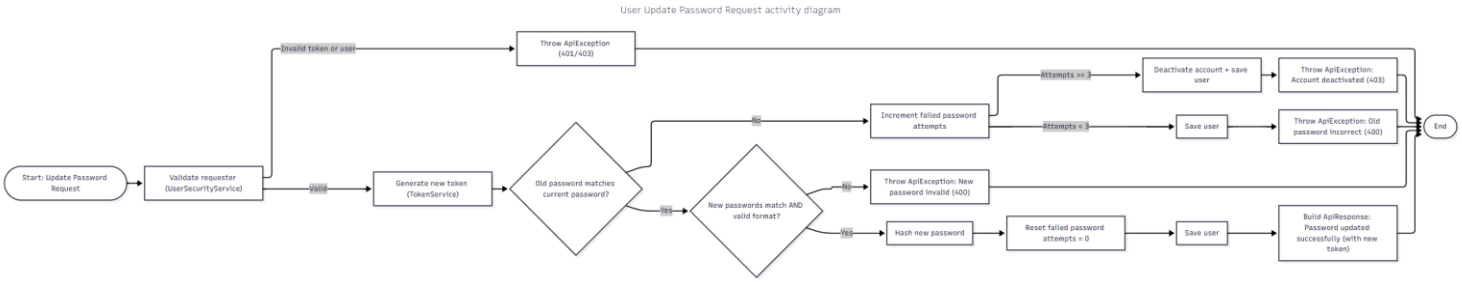
Το διάγραμμα αυτό απεικονίζει την τυπική ροή σύνδεσης. Ο χρήστης αναζητείται με βάση το username. Αν δεν υπάρχει, επιστρέφεται σφάλμα. Αν ο λογαριασμός είναι απενεργοποιημένος, απορρίπτεται η σύνδεση. Στη συνέχεια γίνεται έλεγχος του password, αν δεν ταιριάζει, επιστρέφεται σφάλμα. Αν όλα είναι σωστά, παράγεται νέο token με ημερομηνία λήξης και επιστρέφεται στο χρήστη με μήνυμα επιτυχίας

Activity Diagram – userInfo:



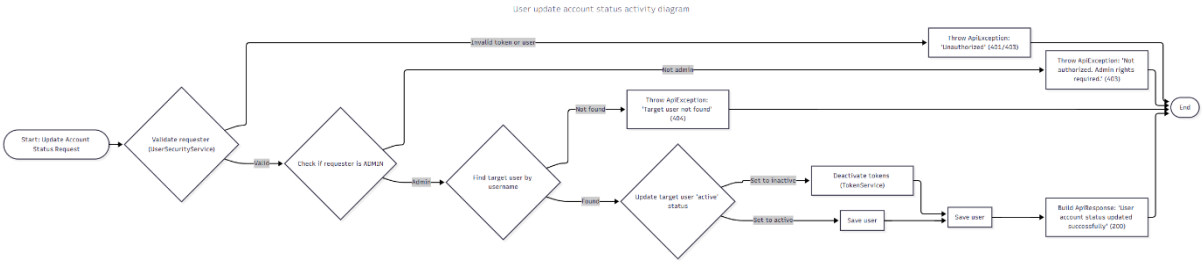
Το διάγραμμα παρουσιάζει την διαδικασία ενημέρωσης των στοιχείων χρήστη. Αρχικά γίνεται επαλήθευση του requester μέσω username και token. Αν το targetUsername είναι null, τότε μόνο ο Admin μπορεί να ενημερώσει τα στοιχεία του χρήστη, αλλιώς γίνεται ενημέρωση και από τον ίδιο τον χρήστη. Αν υπάρχει νέο full name, ενημερώνεται. Επίσης αν υπάρχει νέο username, ελέγχεται αρχικά η διαθεσιμότητα του. Εάν το username είναι διαθέσιμο αλλάζει και δημιουργείται νέο token που επιστρέφεται στην απάντηση. Αλλιώς καλείται η εξαίρεση. Στο τέλος αποθηκεύονται οι αλλαγές και επιστρέφεται μήνυμα επιτυχίας.

Activity Diagram – userPassword:



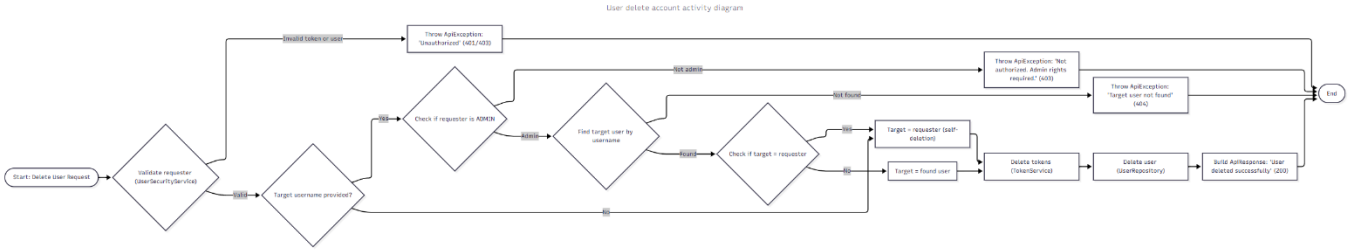
Η διαδικασία αλλαγής κωδικού πρόσβασης ξεκινά με την επαλήθευση του χρήστη που υποβάλλει το αίτημα (requester). Σε περίπτωση επιτυχίας, δημιουργείται ένα νέο token που θα επιστραφεί ως ένδειξη επιτυχημένης ολοκλήρωσης της διαδικασίας. Ακολουθεί έλεγχος του παλιού κωδικού: εάν είναι λανθασμένος, αυξάνεται ο μετρητής αποτυχημένων προσπαθειών. Σε περίπτωση που ο μετρητής φτάσει τις τρεις αποτυχημένες προσπάθειες, ο λογαριασμός απενεργοποιείται και εκκινείται σχετική εξαίρεση. Αν οι αποτυχημένες προσπάθειες είναι λιγότερες από τρεις, καλείται εξαίρεση με μήνυμα «Λάθος password». Αντίθετα, όταν το παλιό password είναι σωστό, πραγματοποιείται έλεγχος για τη συμφωνία και την εγκυρότητα των νέων κωδικών. Σε περίπτωση ασυμφωνίας ή μη έγκυρων νέων κωδικών, εκκινείται η αντίστοιχη εξαίρεση. Όταν όλα τα δεδομένα είναι σωστά, ο νέος κωδικός κρυπτογραφείται με hash, ο μετρητής αποτυχημένων προσπαθειών επανέρχεται στο μηδέν και ο χρήστης αποθηκεύεται στη βάση δεδομένων. Η διαδικασία ολοκληρώνεται με την επιστροφή μηνύματος επιτυχίας μαζί με το νέο token.

Activity Diagram – updateAccount:



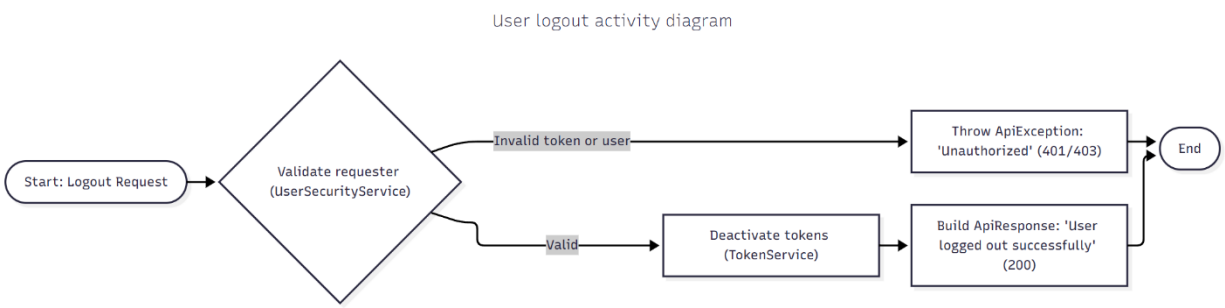
Το διάγραμμα παρουσιάζει τη διαδικασία ενημέρωσης της κατάστασης λογαριασμού χρήστη. Αρχικά ελέγχεται η εγκυρότητα του requester. Αν δεν είναι Admin ή το token άκυρο, επιστρέφεται σφάλμα. Αν ο Admin ζητά αλλαγή, αναζητείται ο χρήστης στόχος. Αν βρεθεί, ενημερώνεται η κατάσταση (active/inactive) και, αν απενεργοποιείται, απενεργοποιούνται και τα tokens. Τέλος αποθηκεύονται οι αλλαγές και επιστρέφεται μήνυμα επιτυχίας.

Activity Diagram – deleteAccount:



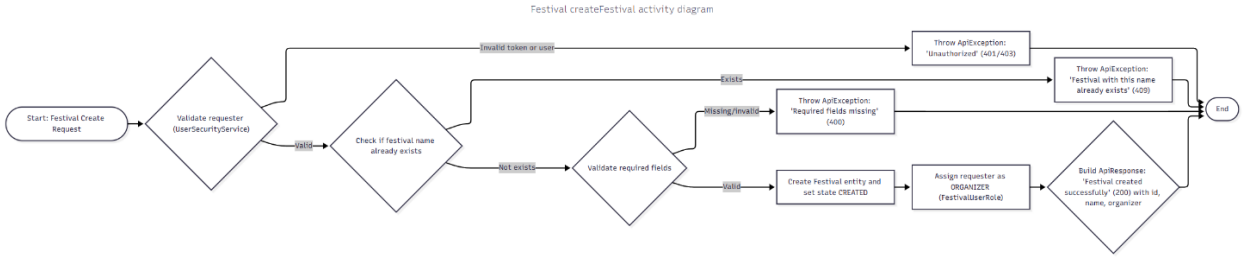
Το διάγραμμα παρουσιάζει τη διαδικασία διαγραφής χρήστη. Αρχικά ελέγχεται η εγκυρότητα του requester. Αν το token είναι άκυρο ή ο χρήστης μη εξουσιοδοτημένος, επιστρέφεται σφάλμα «Unauthorized» (401/403). Αν δεν παρέχεται target username, ο requester διαγράφει τον λογαριασμό του (self-deletion). Αν παρέχεται username, ελέγχεται αν ο requester είναι Admin. Αν δεν είναι, επιστρέφεται σφάλμα «Not authorized» (403). Αν είναι Admin, αναζητείται ο target user. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Target user not found» (404). Αν βρεθεί, καθορίζεται ο χρήστης που θα διαγραφεί. Ακολουθεί διαγραφή των tokens και του χρήστη, και επιστρέφεται μήνυμα επιτυχίας «User deleted successfully» (200).

Activity Diagram – User logout:



Το διάγραμμα παρουσιάζει τη διαδικασία αποσύνδεσης χρήστη (logout). Αρχικά ελέγχεται η εγκυρότητα του requester. Αν το token είναι άκυρο ή ο χρήστης μη εξουσιοδοτημένος, επιστρέφεται σφάλμα «Unauthorized» (401/403). Αν ο χρήστης είναι έγκυρος, απενεργοποιούνται τα tokens του (TokenService) και επιστρέφεται μήνυμα επιτυχίας «User logged out successfully» (200).

Activity Diagram – CreateFestival:

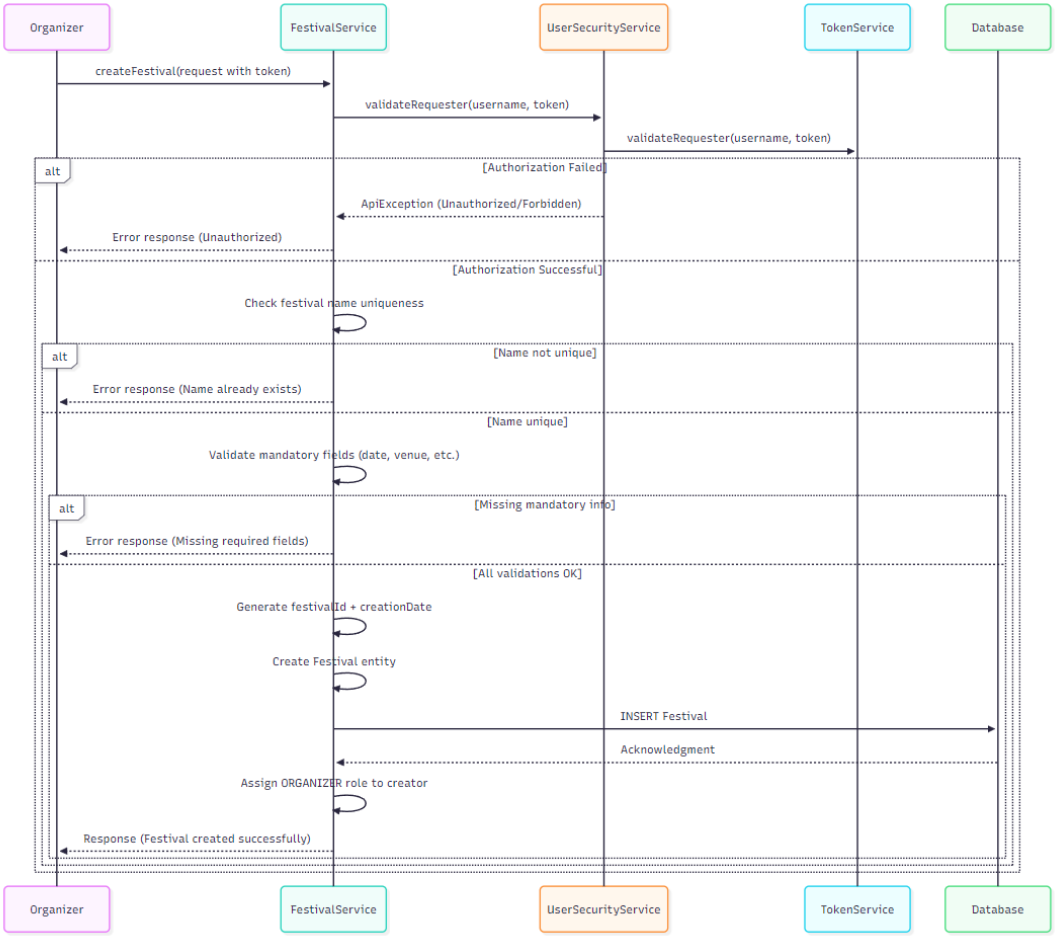


Το διάγραμμα παρουσιάζει τη διαδικασία δημιουργίας φεστιβάλ. Αρχικά ελέγχεται η εγκυρότητα του requester. Αν το token είναι άκυρο ή ο χρήστης μη εξουσιοδοτημένος, επιστρέφεται σφάλμα «Unauthorized» (401/403). Στη συνέχεια ελέγχεται αν το όνομα του φεστιβάλ υπάρχει ήδη. Αν υπάρχει, επιστρέφεται σφάλμα «Festival with this name already exists» (409). Αν όχι, επαληθεύονται τα υποχρεωτικά πεδία. Αν λείπουν ή είναι άκυρα, επιστρέφεται σφάλμα «Required fields missing» (400). Αν όλα είναι έγκυρα, δημιουργείται το φεστιβάλ με κατάσταση CREATED, ο requester ανατίθεται ως ORGANIZER, και επιστρέφεται μήνυμα επιτυχίας «Festival created successfully» (200) με id, όνομα και organizer.

Activity Diagram – CreateFestival:

4.3.2 Sequence Diagramms

sequence diagram – festival creation

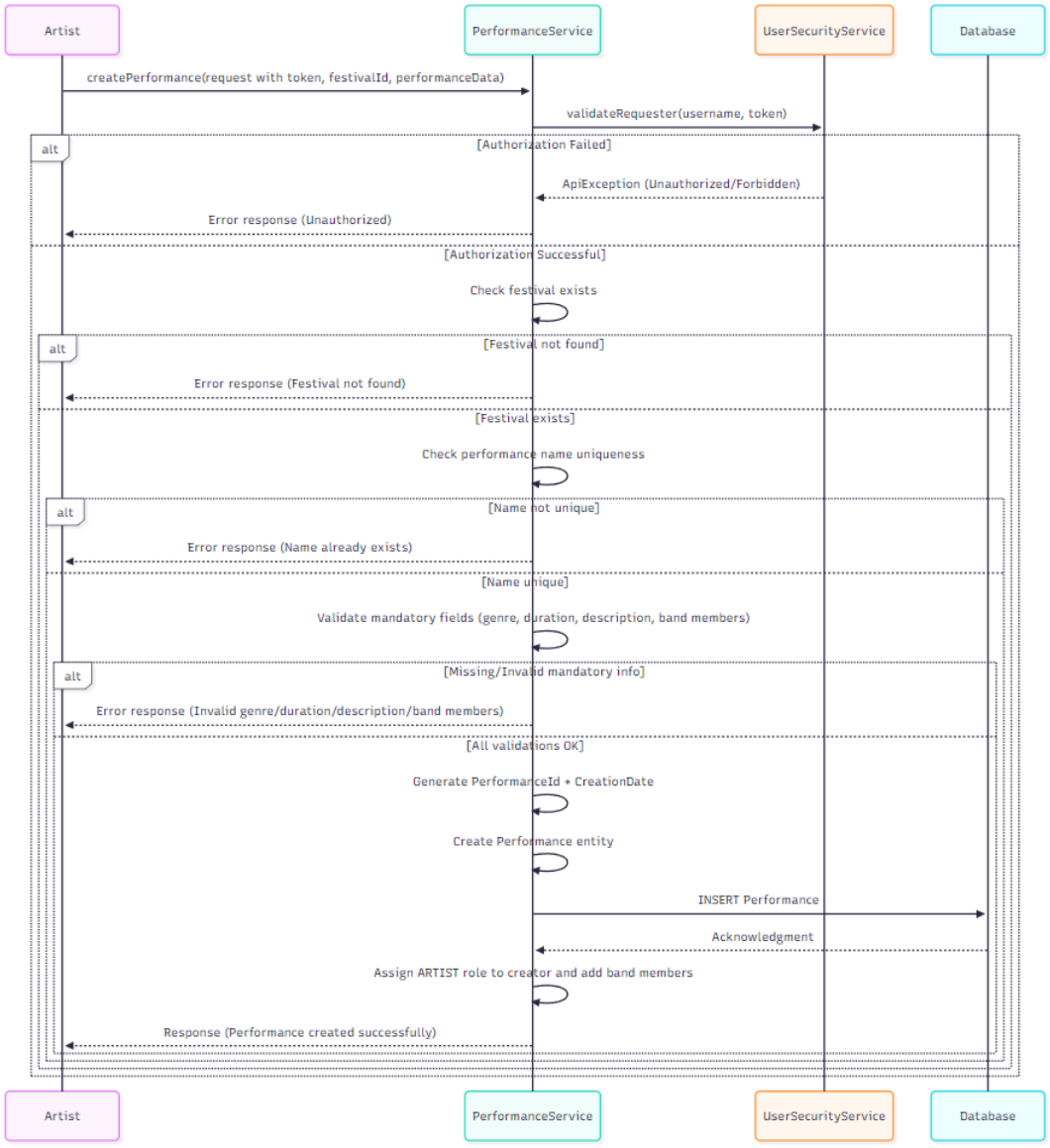


Το παραπάνω Sequence Diagram απεικονίζει τη διαδικασία δημιουργίας ενός νέου φεστιβάλ από έναν διοργανωτή μέσω του συστήματος. Αρχικά, ο διοργανωτής στέλνει αίτημα δημιουργίας φεστιβάλ μαζί με το διακριτικό ταυτοποίησης (token). Το FestivalService προωθεί το αίτημα στο UserSecurityService, το οποίο με τη σειρά του χρησιμοποιεί το TokenService για να επικυρώσει την ταυτότητα του αιτούντος. Αν η επικύρωση αποτύχει, επιστρέφεται μήνυμα σφάλματος μη εξουσιοδότησης (Unauthorized/Forbidden). Εφόσον η επικύρωση είναι επιτυχής, το FestivalService ελέγχει αν το όνομα του φεστιβάλ είναι μοναδικό. Σε περίπτωση που το όνομα υπάρχει ήδη, επιστρέφεται σχετικό μήνυμα σφάλματος. Αν το όνομα είναι μοναδικό, γίνεται επαλήθευση των υποχρεωτικών πεδίων (ημερομηνία, τοποθεσία κ.λπ.). Αν λείπουν απαιτούμενες πληροφορίες, επιστρέφεται αντίστοιχο μήνυμα σφάλματος.

Όταν όλα τα βήματα επαλήθευσης ολοκληρωθούν με επιτυχία, το σύστημα δημιουργεί τα αναγνωριστικά και την ημερομηνία δημιουργίας, κατασκευάζει την οντότητα Festival και την καταχωρεί στη βάση δεδομένων. Στη συνέχεια, ανατίθεται ο ρόλος ORGANIZER στον δημιουργό και αποστέλλεται επιβεβαίωση επιτυχούς δημιουργίας του φεστιβάλ.

Με αυτόν τον τρόπο το διάγραμμα καταγράφει όλη τη ροή της διαδικασίας, από την αρχική επικύρωση ταυτότητας έως την τελική δημιουργία και αποθήκευση της οντότητας του φεστιβάλ.

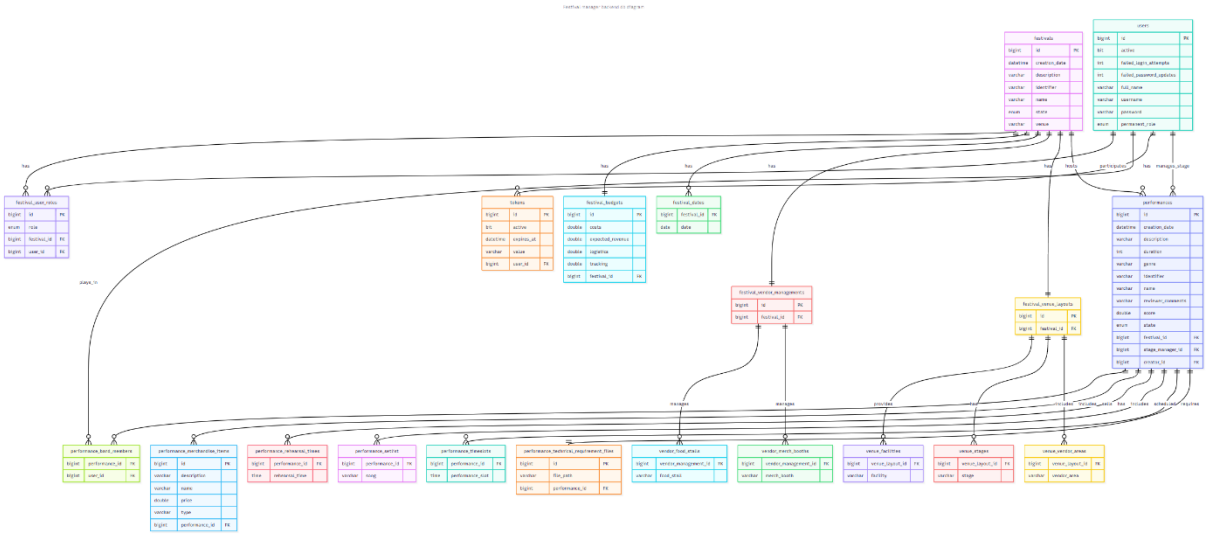
Sequence diagram - performance creation



Το διάγραμμα ακολουθίας παρουσιάζει τη διαδικασία δημιουργίας μίας νέας performance από έναν καλλιτέχνη (Artist). Αρχικά, ο χρήστης στέλνει αίτημα στο PerformanceService μαζί με τα απαραίτητα στοιχεία (token, festivalId, performanceData). Το αίτημα περνάει από έλεγχο αυθεντικοποίησης μέσω του UserSecurityService· σε περίπτωση αποτυχίας, επιστρέφεται μήνυμα σφάλματος, ενώ σε περίπτωση επιτυχίας συνεχίζεται η διαδικασία. Στη συνέχεια, το σύστημα ελέγχει αν υπάρχει το αντίστοιχο φεστιβάλ, αν το όνομα της performance είναι μοναδικό και αν όλα τα υποχρεωτικά πεδία (είδος, διάρκεια, περιγραφή, μέλη συγκροτήματος) είναι έγκυρα. Αν κάποιος από αυτούς τους ελέγχους αποτύχει, επιστρέφεται κατάλληλο μήνυμα λάθους στον χρήστη. Εφόσον όλες οι επικυρώσεις ολοκληρωθούν επιτυχώς, δημιουργείται η performance entity, αποθηκεύεται στη Database, και αποδίδεται στον δημιουργό ο ρόλος ARTIST μαζί με τα μέλη του συγκροτήματος. Τέλος, ο καλλιτέχνης λαμβάνει επιβεβαίωση ότι η performance δημιουργήθηκε με επιτυχία.

4.4 Οντότητες Συστήματος

ER / database diagram



Το παραπάνω διάγραμμα απεικονίζει τις βασικές οντότητες και τις μεταξύ τους συσχετίσεις για το σύστημα Festival & Performance Management. Μέσω αυτού ορίζεται η λογική δομή της βάσης δεδομένων που θα υποστηρίξει τη λειτουργικότητα του συστήματος.

- User: Αποτελεί την κεντρική οντότητα του συστήματος. Κάθε χρήστης διαθέτει μοναδικό username και αποθηκεύονται στοιχεία όπως κωδικός πρόσβασης, πλήρες όνομα και ρόλος. Οι ρόλοι διαχωρίζονται σε Visitor, Artist, Staff και Organizer, καθορίζοντας τα δικαιώματα και τις δυνατότητες κάθε χρήστη.
- Festival: Περιλαμβάνει τις πληροφορίες ενός φεστιβάλ, όπως τίτλο, ημερομηνία, τοποθεσία και κατάσταση (ενεργό/ανενεργό). Ένα festival μπορεί να περιλαμβάνει πολλές παραστάσεις (performances) και συνδέεται άμεσα με τον χρήστη που το έχει δημιουργήσει (Organizer).
- Performance: Αντιπροσωπεύει μια καλλιτεχνική συμμετοχή σε ένα festival. Περιλαμβάνει στοιχεία όπως τίτλο, περιγραφή, διάρκεια, κατάσταση έγκρισης και συμμετέχοντες καλλιτέχνες. Κάθε performance ανήκει σε ένα μόνο festival, αλλά μπορεί να συνδέεται με περισσότερους από έναν καλλιτέχνες.
- Artist: Εξειδικευμένος τύπος χρήστη που έχει τη δυνατότητα να δημιουργεί, να υποβάλλει και να ενημερώνει performances. Οι καλλιτέχνες συνδέονται με ένα ή περισσότερα performances, ενδεχομένως σε διαφορετικά φεστιβάλ.
- Staff: Οντότητα που αναπαριστά μέλη της ομάδας υποστήριξης ενός festival (π.χ. τεχνικοί, stage managers). Συνδέονται με συγκεκριμένα festivals και performances και έχουν ρόλο στην αξιολόγηση και οργάνωση τους.

Συσχετίσεις:

- Festival – Performance (1:N): Κάθε festival μπορεί να περιλαμβάνει πολλές παραστάσεις.
- Performance – Artist (M:N): Κάθε performance μπορεί να έχει πολλούς καλλιτέχνες και κάθε καλλιτέχνης μπορεί να συμμετέχει σε πολλά performances. Η σχέση αυτή υλοποιείται μέσω ενδιάμεσου πίνακα.
- Festival – Staff (M:N): Κάθε festival μπορεί να διαθέτει πολλούς υπαλλήλους και κάθε μέλος του staff μπορεί να υποστηρίζει περισσότερα από ένα festivals.
- User – Role (1:N): Κάθε χρήστης έχει έναν ρόλο, ο οποίος καθορίζει τα δικαιώματα του στο σύστημα.

Συνολικά, το διάγραμμα εξασφαλίζει την ορθή αναπαράσταση των βασικών λειτουργικών σχέσεων του συστήματος, επιτρέποντας την οργάνωση δεδομένων με τρόπο που υποστηρίζει την αναζήτηση, προβολή και διαχείριση χρηστών, φεστιβάλ και παραστάσεων. Η χρήση ενδιάμεσων πινάκων για τις συσχετίσεις «many-to-many» διασφαλίζει τη σωστή κανονικοποίηση της βάσης δεδομένων και τη συνέπεια των δεδομένων.

5 Υλοποίηση Συστήματος

5.1 GitHub

Η ανάπτυξη του έργου πραγματοποιήθηκε μέσω GitHub, σε ένα ιδιωτικό αποθετήριο με τίτλο **Festival-Managment-Backend**.

Το αποθετήριο είναι διαθέσιμο στη διεύθυνση:
https://github.com/stelios1361/Festival-Managment-Backend

Στο έργο συνέβαλαν οι ακόλουθοι συνεργάτες:

1. Νικολόπουλος Στυλιανός
2. Αναγνωστόπουλος Γεώργιος
3. Καμτσικλής Γεώργιος

Η χρήση pull requests και code reviews συνέβαλε στη διατήρηση της ποιότητας του κώδικα και στη συνεργατική ανάπτυξη.

5.2 Τεκμηρίωση εφαρμογής

Το σύστημα έχει υλοποιηθεί χρησιμοποιώντας τεχνολογίες και εργαλεία που εξασφαλίζουν αποδοτικότητα, modularity και επεκτασιμότητα. Το project είναι βασισμένο σε Spring Boot με Maven για διαχείριση εξαρτήσεων και build. Η εφαρμογή έχει αναπτυχθεί στο NetBeans IDE. Η βάση δεδομένων υλοποιείται σε MySQL, που τρέχει μέσω XAMPP για την εύκολη διαχείριση του τοπικού server και των βάσεων δεδομένων. Για την επικοινωνία με το API και τη δοκιμή των endpoints χρησιμοποιήθηκε το Postman. Ο κώδικας βρίσκεται σε GitHub repository, ώστε να είναι δυνατή η συνεργασία και η παρακολούθηση των αλλαγών.

Η αρχιτεκτονική του συστήματος ακολουθεί το κλασικό pattern Controller → Service → Repository, με καθαρή διάκριση των επιπέδων ευθύνης:

- Τα Controllers διαχειρίζονται τα αιτήματα των χρηστών και την απόκριση.
- Τα Services υλοποιούν την επιχειρησιακή λογική.
- Τα Repositories / DAO αλληλεπιδρούν με τη βάση δεδομένων.

Ο οργανισμός του κώδικα είναι ομαδοποιημένος σε φακέλους με βάση το domain (Users, Festival, Performance), ώστε να είναι εύκολα επαναχρησιμοποιήσιμος και κατανοητός. Το σύστημα υποστηρίζει RESTful endpoints, για παράδειγμα:

- /api/users/... για διαχείριση χρηστών (εγγραφή, login, update, διαγραφή).
- /api/festival/... για διαχείριση φεστιβάλ (δημιουργία, ενημέρωση, αλλαγή κατάστασης).
- /api/performance/... για διαχείριση εμφανίσεων (δημιουργία, update, υποβολή, έγκριση).

Η ασφάλεια διασφαλίζεται με χρήση JWT tokens για authorisation, τα οποία ελέγχονται σε κάθε αίτημα ώστε να επιτρέπεται μόνο η πρόσβαση σε authenticated χρήστες με τα κατάλληλα roles. Το modular design και η ομαλή διαχείριση των dependencies επιτρέπουν εύκολη συντήρηση και επέκταση του συστήματος.

5.3 Τεκμηρίωση δοκιμών

5.3.1 UnitTests

Έγινε επιλογή των συγκεκριμένων κλάσεων για δοκιμές καθώς θεωρήσαμε πως είναι οι πιο σημαντικές από το σύνολο κλάσεων που απαρτίζουν το σύστημα.

UnitTests – PasswordServiceTest

Το PasswordServiceTest ελέγχει τη λειτουργία του PasswordService. Hashing και verification με BCrypt:

- **testHashAndMatch_success:**
  Δοκιμάζει ότι ένα raw password, αφού γίνει hash, ταιριάζει σωστά με τον έλεγχο matches. Το αναμενόμενο αποτέλεσμα πρέπει να είναι true.

```
    @Test
    void testHashAndMatch_success() {
        String rawPassword = "mySecret123";
        String hashedPassword = passwordService.hash(rawPassword);

        System.out.println(x: "Running testHashAndMatch_success:");
        System.out.println("Raw password: " + rawPassword);
        System.out.println("Hashed password: " + hashedPassword);

        assertNotNull(actual: hashedPassword, message:"Hashed password should not be null");
        assertTrue(condition: passwordService.matches(rawPassword, hashedPassword),
                message:"PasswordService should match raw password with hashed password");

        System.out.println(x: "testHashAndMatch_success completed successfully\n");
    }
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.festivalmanager.security.PasswordServiceTest
=== PasswordServiceTest setup completed ===
Running testHashAndMatch_success:
Raw password: mySecret123
Hashed password: $2a$10$ey6vbL6idFb0abibQFpr3ONU6j8zKjZSj8CImVrVDkbogsqLDwbWy
testHashAndMatch_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.470 s -- in com.festivalmanager.security.PasswordServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testMatchFails_forWrongPassword:**
  Δοκιμάζει ότι δύο διαφορετικά passwords δεν ταιριάζουν, ακόμη κι αν το ένα είναι hashed. Αναμενόμενο αποτέλεσμα θα είναι false.

```
    @Test
    void testMatchFails_forWrongPassword() {
        String rawPassword = "password1";
        String hashedPassword = passwordService.hash(rawPassword:"password2");

        System.out.println(x: "Running testMatchFails_forWrongPassword:");
        System.out.println("Raw password: " + rawPassword);
        System.out.println("Hashed password: " + hashedPassword);

        assertFalse(condition: passwordService.matches(rawPassword, hashedPassword),
                message:"PasswordService should return false for non-matching passwords");

        System.out.println(x: "testMatchFails_forWrongPassword completed successfully\n");
    }
}
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.festivalmanager.security.PasswordServiceTest
=== PasswordServiceTest setup completed ===
Running testMatchFails_forWrongPassword:
Raw password: password1
Hashed password: $2a$10$htm98cY9AaQWI9wdiSYqHu/WGI3Zlaznz8ad3zuq7KBLXpNfX7iMi
testMatchFails_forWrongPassword completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.360 s -- in com.festivalmanager.security.PasswordServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

UnitTests – UserSecurityServiceTest:

Το UserSecurityServiceTest ελέγχει τη λειτουργία του UserSecurityService.validateRequester, δηλαδή αν ένας χρήστης μπορεί να θεωρηθεί έγκυρος αιτών, με βάση την ύπαρξή του, την κατάστασή του (active) και το token του.

- **testValidateRequester_success:**
  Ο χρήστης υπάρχει, είναι ενεργός και το token είναι έγκυρο. Αναμενόμενο αποτέλεσμα να επιστρέφεται ο χρήστης.

```
@Test
void testValidateRequester_success() {
    User user = new User();
    user.setUsername(username: "alice");
    user.setActive(active: true);

    when(methodCall: userRepository.findByUsername(username: "alice")).thenReturn(t: Optional.of(value: user));
    when(methodCall: tokenService.validateToken(value: "token123", requestingUser: user)).thenReturn(t: true);

    System.out.println(x: "Running testValidateRequester_success");

    User result = securityService.validateRequester(requesterUsername: "alice", token: "token123");

    assertEquals(expected: "alice", actual: result.getUsername(), message: "Validated user should match requested username");
    verify(mock: tokenService).validateToken(value: "token123", requestingUser: user);

    System.out.println(x: "testValidateRequester_success completed successfully\n");
}
```

```
Running testValidateRequester_success
testValidateRequester_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.677 s -- in com.festivalmanager.security.UserSecurityServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testValidateRequester_userNotFound:**
  Ο χρήστης δεν υπάρχει στη βάση. Αναμενόμενο αποτέλεσμα ApiException με HttpStatus.UNAUTHORIZED.

```
@Test
void testValidateRequester_userNotFound() {
    when(methodCall: userRepository.findByUsername(username: "bob")).thenReturn(t: Optional.empty());

    System.out.println(x: "Running testValidateRequester_userNotFound");

    ApiException ex = assertThrows(expectedType: ApiException.class, () ->
            securityService.validateRequester(requesterUsername: "bob", token: "token123")
    );

    assertEquals(expected: HttpStatus.UNAUTHORIZED, actual: ex.getStatus(), message: "Nonexistent user should throw UNAUTHORIZED");

    System.out.println(x: "testValidateRequester_userNotFound completed successfully\n");
}
```

```
Running testValidateRequester_userNotFound
testValidateRequester_userNotFound completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.772 s -- in com.festivalmanager.security.UserSecurityServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testValidateRequester_userInactive:**
  Ο χρήστης υπάρχει αλλά είναι απενεργοποιημένος. Αναμενόμενο αποτέλεσμα ApiException με HttpStatus.FORBIDDEN.

```
    @Test
    void testValidateRequester_userInactive() {
        User user = new User();
        user.setUsername(username: "charlie");
        user.setActive(active: false);

        when(methodCall: userRepository.findByUsername(username: "charlie")).thenReturn(t: Optional.of(value: user));

        System.out.println(x: "Running testValidateRequester_userInactive");

        ApiException ex = assertThrows(expectedType: ApiException.class, () ->
                securityService.validateRequester(requesterUsername: "charlie", token: "token123")
        );

        assertEquals(expected: HttpStatus.FORBIDDEN, actual: ex.getStatus(), message: "Inactive user should throw FORBIDDEN");

        System.out.println(x: "testValidateRequester_userInactive completed successfully\n");
    }
```

```
Running testValidateRequester_userInactive
testValidateRequester_userInactive completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.713 s -- in com.festivalmanager.security.UserSecurityServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testValidateRequester_tokenInvalid**
  Ο χρήστης υπάρχει και είναι ενεργός, αλλά το token δεν είναι έγκυρο. Αναμενόμενο αποτέλεσμα ApiException με HttpStatus.UNAUTHORIZED.

```
@Test
void testValidateRequester_tokenInvalid() {
    User user = new User();
    user.setUsername(username: "dave");
    user.setActive(active: true);

    when(methodCall: userRepository.findByUsername(username: "dave")).thenReturn(t: Optional.of(value: user));
    doThrow(new ApiException(message: "Invalid token", status: HttpStatus.UNAUTHORIZED))
            .when(t: tokenService).validateToken(value: "badToken", requestingUser: user);

    System.out.println(x: "Running testValidateRequester_tokenInvalid");

    ApiException ex = assertThrows(expectedType: ApiException.class, () ->
            securityService.validateRequester(requesterUsername: "dave", token: "badToken")
    );

    assertEquals(expected: HttpStatus.UNAUTHORIZED, actual: ex.getStatus(), message: "Invalid token should throw UNAUTHORIZED");

    System.out.println(x: "testValidateRequester_tokenInvalid completed successfully\n");
}
```

```
Running testValidateRequester_tokenInvalid
testValidateRequester_tokenInvalid completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.791 s -- in com.festivalmanager.security.UserSecurityServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

UnitTests – TokenServiceTest:

Το TokenServiceTest ελέγχει τη λειτουργία του TokenService, δηλαδή τη διαχείριση tokens για τους χρήστες (δημιουργία, απενεργοποίηση, διαγραφή και έλεγχο εγκυρότητας).

- **testGenerateToken:**
  Δημιουργεί νέο token για έναν χρήστη. Απενεργοποιεί τυχόν παλιά tokens και αποθηκεύει το νέο. Αναμενόμενο αποτέλεσμα το νέο token έχει τιμή (value), είναι ενεργό και ανήκει στον χρήστη.

```
@Test
void testGenerateToken() {
    User user = new User();
    Token oldToken = new Token();
    oldToken.setActive(active: true);

    when(methodCall: tokenRepository.findAllByUser(user)).thenReturn(t: Collections.singletonList(o: oldToken));
    when(methodCall: tokenRepository.saveAll(entities: anyList())).thenReturn(t: Collections.singletonList(o: oldToken));
    when(methodCall: tokenRepository.save(entity: any(type: Token.class))).thenAnswer(i -> i.getArgument(i: 0));

    System.out.println(x: "Running testGenerateToken");

    Token newToken = tokenService.generateToken(user);

    assertNotNull(actual: newToken.getValue(), message: "Generated token value should not be null");
    assertTrue(condition: newToken.isActive(), message: "Generated token should be active");
    assertEquals(expected: user, actual: newToken.getUser(), message: "Generated token should belong to the correct user");

    verify(mock: tokenRepository).saveAll(entities: anyList());
    verify(mock: tokenRepository).save(entity: newToken);

    System.out.println(x: "testGenerateToken completed successfully\n");
}
```

```
Running testGenerateToken
testGenerateToken completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.899 s -- in com.festivalmanager.service.TokenServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testDeactivateTokens:**
  Απενεργοποιεί όλα τα tokens ενός χρήστη. Αναμενόμενο αποτέλεσμα όλα τα tokens γίνονται inactive και αποθηκεύονται ξανά.

```
@Test
void testDeactivateTokens() {
    User user = new User();
    Token token = new Token();
    token.setActive(active: true);

    when(methodCall: tokenRepository.findAllByUser(user)).thenReturn(t: Collections.singletonList(o: token));
    when(methodCall: tokenRepository.saveAll(entities: anyList())).thenReturn(t: Collections.singletonList(o: token));

    System.out.println(x: "Running testDeactivateTokens");

    tokenService.deactivateTokens(user);

    assertFalse(condition: token.isActive(), message: "Token should be deactivated");
    verify(mock: tokenRepository).saveAll(entities: anyList());

    System.out.println(x: "testDeactivateTokens completed successfully\n");
}
```

```
=== TokenServiceTest setup completed ===

Running testDeactivateTokens
testDeactivateTokens completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.677 s -- in com.festivalmanager.service.TokenServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testDeleteTokens:**
  Διαγράφει όλα τα tokens ενός χρήστη από το repository. Αναμενόμενο αποτέλεσμα καλείται το deleteByUser(user).

```java
@Test
void testDeleteTokens() {
    User user = new User();

    System.out.println(x: "Running testDeleteTokens");

    tokenService.deleteTokens(user);

    verify(mock: tokenRepository).deleteByUser(user);

    System.out.println(x: "testDeleteTokens completed successfully\n");
}
```

```
=== TokenServiceTest setup completed ===

Running testDeleteTokens
testDeleteTokens completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.985 s -- in com.festivalmanager.service.TokenServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testValidateToken_success:**
  Ελέγχει ότι ένα ενεργό, μη ληγμένο token που αντιστοιχεί στον χρήστη είναι έγκυρο. Αναμενόμενο αποτέλεσμα να επιστρέφει true.

```
@Test
void testValidateToken_success() {
    User user = new User();
    user.setUsername(username: "alice");
    user.setActive(active: true);

    Token token = new Token();
    token.setValue(value: "token123");
    token.setUser(user);
    token.setActive(active: true);
    token.setExpiresAt(expiresAt: LocalDateTime.now().plusHours(hours: 1));

    when(methodCall: tokenRepository.findByValue(value: "token123")).thenReturn(t: Optional.of(value: token));

    System.out.println(x: "Running testValidateToken_success");

    boolean result = tokenService.validateToken(value: "token123", requestingUser: user);

    assertTrue(condition: result, message:"Token should be valid");

    System.out.println(x: "testValidateToken_success completed successfully\n");
}
```

```
=== TokenServiceTest setup completed ===

Running testValidateToken_success
testValidateToken_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.831 s -- in com.festivalmanager.service.TokenServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testValidateToken_expired:**
  Ελέγχει ότι ένα token που έχει λήξει απορρίπτεται. Το αναμενόμενο αποτέλεσμα είναι ApiException με HttpStatus.UNAUTHORIZED και το token γίνεται inactive.

```
@Test
void testValidateToken_expired() {
    User user = new User();
    Token token = new Token();
    token.setValue(value: "expiredToken");
    token.setUser(user);
    token.setActive(active: true);
    token.setExpiresAt(expiresAt: LocalDateTime.now().minusMinutes(minutes:1));

    when(methodCall: tokenRepository.findByValue(value: "expiredToken")).thenReturn(t: Optional.of(value: token));
    when(methodCall: tokenRepository.saveAndFlush(entity: token)).thenReturn(t: token);

    System.out.println(x: "Running testValidateToken_expired");

    ApiException ex = assertThrows(expectedType:ApiException.class, ()
            -> tokenService.validateToken(value: "expiredToken", requestingUser: user)
    );

    assertEquals(expected:HttpStatus.UNAUTHORIZED, actual: ex.getStatus(), message:"Expired token should throw UNAUTHORIZED");
    assertFalse(condition: token.isActive(), message:"Expired token should be deactivated");

    System.out.println(x: "testValidateToken_expired completed successfully\n");
}
```

```
=== TokenServiceTest setup completed ===

Running testValidateToken_expired
testValidateToken_expired completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.737 s -- in com.festivalmanager.service.TokenServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

UnitTests – UserServiceTest:

Το UserServiceTest ελέγχει τη βασική λειτουργικότητα του UserService, δηλαδή τη διαχείριση χρηστών (εγγραφή, login, αλλαγή κωδικού, διαγραφή).

- **testRegisterUser_success:**
  Δημιουργία νέου χρήστη με σωστά στοιχεία. Αν είναι ο πρώτος χρήστης γίνεται admin. Αναμενόμενο αποτέλεσμα, επιστροφή OK και αποθήκευση του χρήστη.

```java
@Test
void testRegisterUser_success() {
    RegisterRequest req = new RegisterRequest();
    req.setUsername(username: "alice1");
    req.setPassword1(password1: "Strong@123");
    req.setPassword2(password2: "Strong@123");
    req.setFullname(fullname: "Alice Example");

    when(methodCall: userRepository.existsByUsername(username: "alice1")).thenReturn(t: false);
    when(methodCall: userRepository.count()).thenReturn(t: 0L); // first user → admin
    when(methodCall: passwordService.hash(rawPassword: "Strong@123")).thenReturn(t: "hashed");

    System.out.println(x: "Running testRegisterUser_success");

    var response = userService.registerUser(request: req);

    assertEquals(expected: HttpStatus.OK.value(), actual: response.getStatus());
    verify(mock: userRepository).save(entity: any(type: User.class));

    System.out.println(x: "testRegisterUser_success completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testRegisterUser_success
testRegisterUser_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.787 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testRegisterUser_usernameAlreadyExists:**
  Έλεγχος ότι δεν μπορεί να δημιουργηθεί χρήστης με username που υπάρχει ήδη. Αναμενόμενο αποτέλεσμα , ApiException με CONFLICT.

```
@Test
void testRegisterUser_usernameAlreadyExists() {
    RegisterRequest req = new RegisterRequest();
    req.setUsername(username: "bob");

    when(methodCall: userRepository.existsByUsername(username: "bob")).thenReturn(t: true);

    System.out.println(x: "Running testRegisterUser_usernameAlreadyExists");

    ApiException ex = assertThrows(expectedType: ApiException.class, () -> userService.registerUser(request:req));

    assertEquals(expected: HttpStatus.CONFLICT, actual: ex.getStatus());

    System.out.println(x: "testRegisterUser_usernameAlreadyExists completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testRegisterUser_usernameAlreadyExists
testRegisterUser_usernameAlreadyExists completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.087 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testLoginUser_success:**
  Επιτυχές login με σωστό username και password. Αναμενόμενο αποτέλεσμα, επιστροφή OK και δημιουργία νέου token.

```
@Test
void testLoginUser_success() {
    User user = new User();
    user.setUsername(username: "carol");
    user.setPassword(password: "hashedPw");
    user.setActive(active: true);

    LoginRequest req = new LoginRequest();
    req.setUsername(username: "carol");
    req.setPassword(password: "pw123");

    when(methodCall: userRepository.findByUsername(username: "carol")).thenReturn(t: Optional.of(value: user));
    when(methodCall: passwordService.matches(rawPassword: "pw123", hashedPassword: "hashedPw")).thenReturn(t: true);
    when(methodCall: tokenService.generateToken(user)).thenReturn(new com.festivalmanager.model.Token());

    System.out.println(x: "Running testLoginUser_success");

    var response = userService.loginUser(request:req);

    assertEquals(expected: HttpStatus.OK.value(), actual: response.getStatus());
    verify(mock: tokenService).generateToken(user);

    System.out.println(x: "testLoginUser_success completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testLoginUser_success
testLoginUser_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.797 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testLoginUser_deactivateAfter3FailedAttempts:**
  Έλεγχος ότι αν ένας χρήστης αποτύχει 3 φορές στο login, απενεργοποιείται. Αναμενόμενο αποτέλεσμα, ApiException με FORBIDDEN και active = false.

```
@Test
void testLoginUser_deactivateAfter3FailedAttempts() {
    User user = new User();
    user.setUsername(username: "dave");
    user.setPassword(password: "hashedPw");
    user.setActive(active: true);
    user.setFailedLoginAttempts(failedLoginAttempts:2);

    LoginRequest req = new LoginRequest();
    req.setUsername(username: "dave");
    req.setPassword(password: "wrong");

    when(methodCall: userRepository.findByUsername(username: "dave")).thenReturn(t: Optional.of(value: user));
    when(methodCall: passwordService.matches(rawPassword:"wrong", hashedPassword: "hashedPw")).thenReturn(t: false);

    System.out.println(x: "Running testLoginUser_deactivateAfter3FailedAttempts");

    ApiException ex = assertThrows(expectedType: ApiException.class, () -> userService.loginUser(request:req));

    assertEquals(expected: HttpStatus.FORBIDDEN, actual: ex.getStatus());
    assertFalse(condition: user.isActive(), message:"User should be deactivated after 3 failed attempts");

    System.out.println(x: "testLoginUser_deactivateAfter3FailedAttempts completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testLoginUser_deactivateAfter3FailedAttempts
testLoginUser_deactivateAfter3FailedAttempts completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.930 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testUpdateUserPassword_success:**
  Επιτυχής αλλαγή κωδικού όταν το παλιό password είναι σωστό. Αναμενόμενο αποτέλεσμα, επιστροφή OK και αποθήκευση νέου hashed password.

```
@Test
void testUpdateUserPassword_success() {
    User user = new User();
    user.setUsername(username: "emma");
    user.setPassword(password: "hashedOld");
    user.setActive(active: true);

    UpdatePasswordRequest req = new UpdatePasswordRequest();
    req.setRequesterUsername(requesterUsername: "emma");
    req.setToken(token: "tkn");
    req.setOldPassword(oldPassword:"old");
    req.setNewPassword1(newPassword1: "New@1234");
    req.setNewPassword2(newPassword2: "New@1234");

    when(methodCall: userSecurityService.validateRequester(requesterUsername: "emma", token: "tkn")).thenReturn(t: user);
    when(methodCall: passwordService.matches(rawPassword:"old", hashedPassword: "hashedOld")).thenReturn(t: true);
    when(methodCall: passwordService.hash(rawPassword:"New@1234")).thenReturn(t: "hashedNew");
    when(methodCall: tokenService.generateToken(user)).thenReturn(new com.festivalmanager.model.Token());

    System.out.println(x: "Running testUpdateUserPassword_success");

    var response = userService.updateUserPassword(request:req);

    assertEquals(expected: HttpStatus.OK.value(), actual: response.getStatus());
    assertEquals(expected: "hashedNew", actual: user.getPassword());

    System.out.println(x: "testUpdateUserPassword_success completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testUpdateUserPassword_success
testUpdateUserPassword_success completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.816 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testDeleteUser_selfDelete:**
  Ο χρήστης διαγράφει τον εαυτό του. Αναμενόμενο αποτέλεσμα, διαγραφή tokens και χρήστη από τη βάση.

```
@Test
void testDeleteUser_selfDelete() {
    User user = new User();
    user.setUsername(username: "frank");

    DeleteUserRequest req = new DeleteUserRequest();
    req.setRequesterUsername(requesterUsername: "frank");
    req.setToken(token: "tkn");

    when(methodCall: userSecurityService.validateRequester(requesterUsername: "frank", token: "tkn")).thenReturn(t: user);

    System.out.println(x: "Running testDeleteUser_selfDelete");

    userService.deleteUser(request:req);

    verify(mock: tokenService).deleteTokens(user);
    verify(mock: userRepository).delete(entity: user);

    System.out.println(x: "testDeleteUser_selfDelete completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testDeleteUser_selfDelete
testDeleteUser_selfDelete completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.807 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testRegisterUser_invalidUsernamePattern:**
  Έλεγχος ότι username που δεν πληροί τους κανόνες (π.χ. ξεκινά με αριθμό) απορρίπτεται. Αναμενόμενο αποτέλεσμα, ApiException με BAD_REQUEST.

```
@Test
void testRegisterUser_invalidUsernamePattern() {
    RegisterRequest req = new RegisterRequest();
    req.setUsername(username: "123bad"); // invalid: starts with digit
    req.setPassword1(password1: "Strong@123");
    req.setPassword2(password2: "Strong@123");
    req.setFullname(fullname: "Invalid Name");

    when(methodCall: userRepository.existsByUsername(username: "123bad")).thenReturn(t: false);
    when(methodCall: userRepository.count()).thenReturn(t: 1L);

    System.out.println(x: "Running testRegisterUser_invalidUsernamePattern");

    ApiException ex = assertThrows(expectedType: ApiException.class, () -> userService.registerUser(request:req));

    assertEquals(expected: HttpStatus.BAD_REQUEST, actual: ex.getStatus());
    assertTrue(condition: ex.getMessage().contains(s: "Invalid username"));

    System.out.println(x: "testRegisterUser_invalidUsernamePattern completed successfully\n");
}
```

```
=== UserServiceTest setup completed ===

Running testRegisterUser_invalidUsernamePattern
testRegisterUser_invalidUsernamePattern completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.963 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

- **testRegisterUser_invalidPasswordPattern:**
  Έλεγχος ότι password που δεν πληροί τους κανόνες (π.χ. πολύ αδύναμο) απορρίπτεται. Αναμενόμενο αποτέλεσμα, ApiException με BAD_REQUEST.

```
@Test
void testRegisterUser_invalidPasswordPattern() {
    RegisterRequest req = new RegisterRequest();
    req.setUsername(username: "validName");
    req.setPassword1(password1: "weak");   // invalid: too short, no uppercase/special
    req.setPassword2(password2: "weak");
    req.setFullname(fullname: "Weak Password User");

    when(methodCall: userRepository.existsByUsername(username: "validName")).thenReturn(t: false);
    when(methodCall: userRepository.count()).thenReturn(t: 1L);

    System.out.println(x: "Running testRegisterUser_invalidPasswordPattern");

    ApiException ex = assertThrows(expectedType: ApiException.class, () -> userService.registerUser(request:req));

    assertEquals(expected: HttpStatus.BAD_REQUEST, actual: ex.getStatus());
    assertTrue(condition: ex.getMessage().contains(s: "Password must be at least 8 characters"));

    System.out.println(x: "testRegisterUser_invalidPasswordPattern completed successfully\n");
}
```

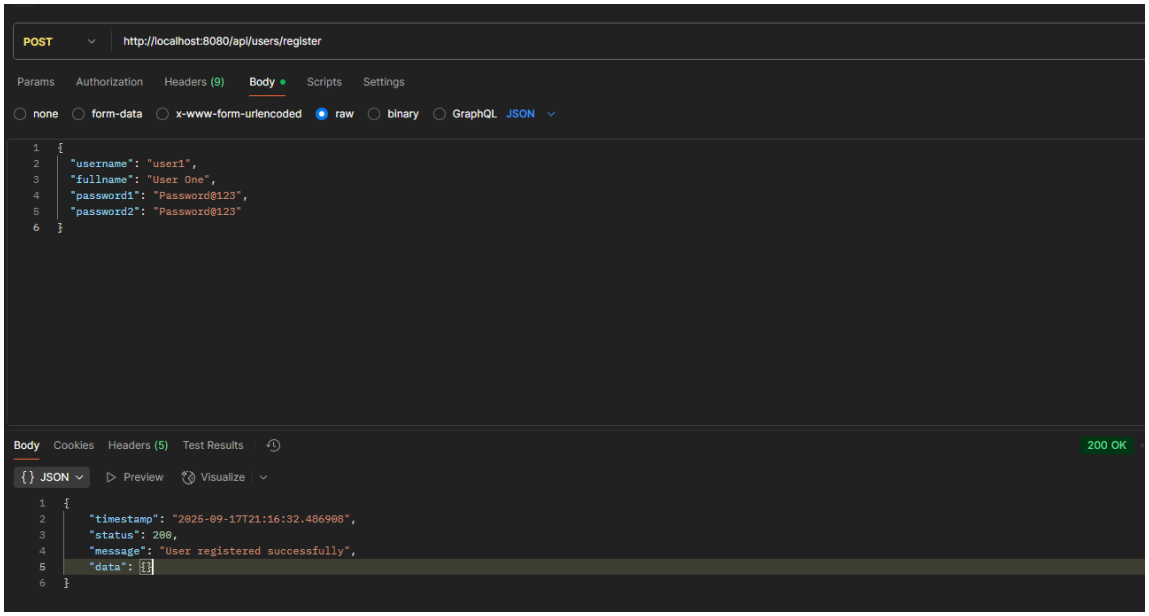
```
=== UserServiceTest setup completed ===

Running testRegisterUser_invalidPasswordPattern
testRegisterUser_invalidPasswordPattern completed successfully

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.180 s -- in com.festivalmanager.service.UserServiceTest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

5.3.2 Postman Testing

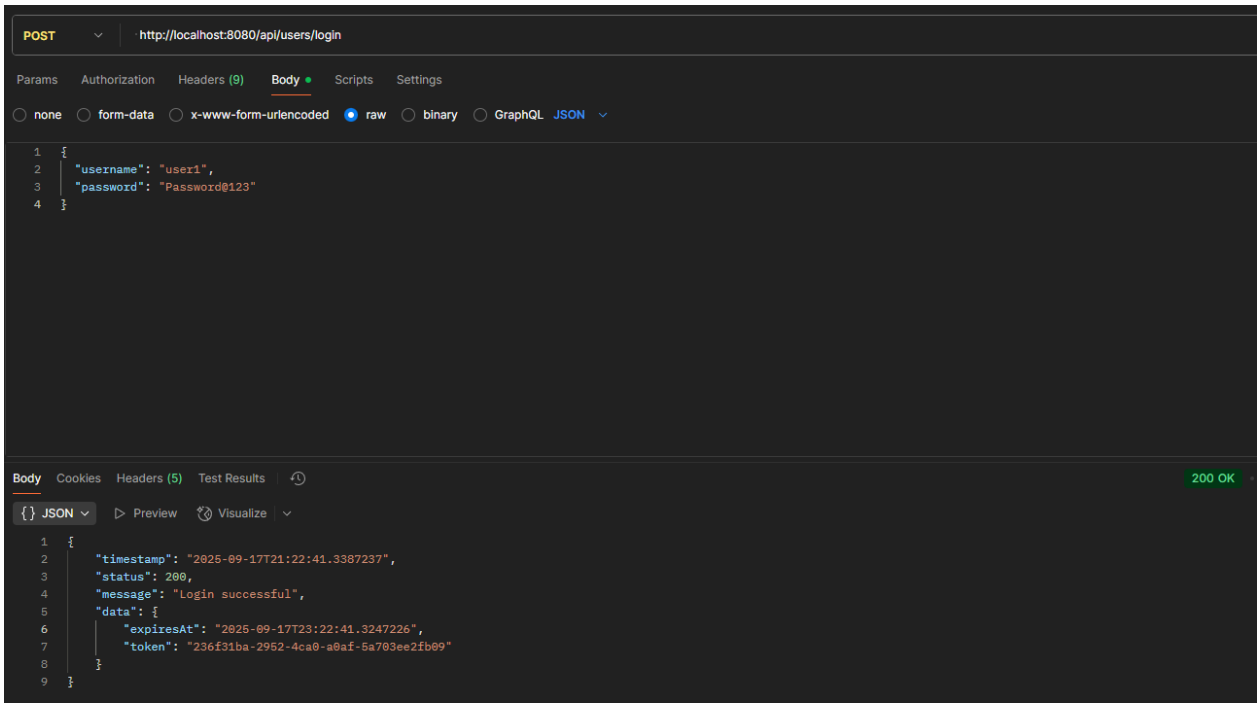
Εκτελούμε μερικές δοκιμές του συστήματος μέσω του Postman, επιβεβαιώνοντας την ορθή λειτουργία της εφαρμογής.

Χρήστης / Admin

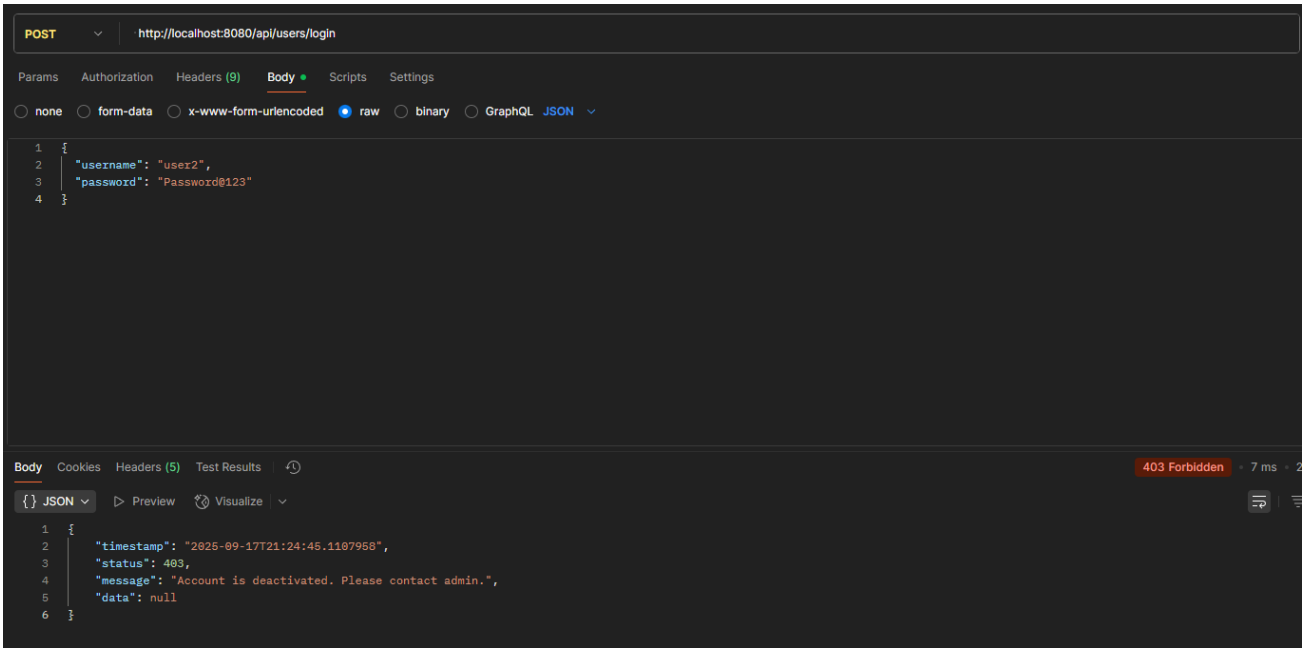
- Εκτέλεση ενέργειας register από τον χρήστη.



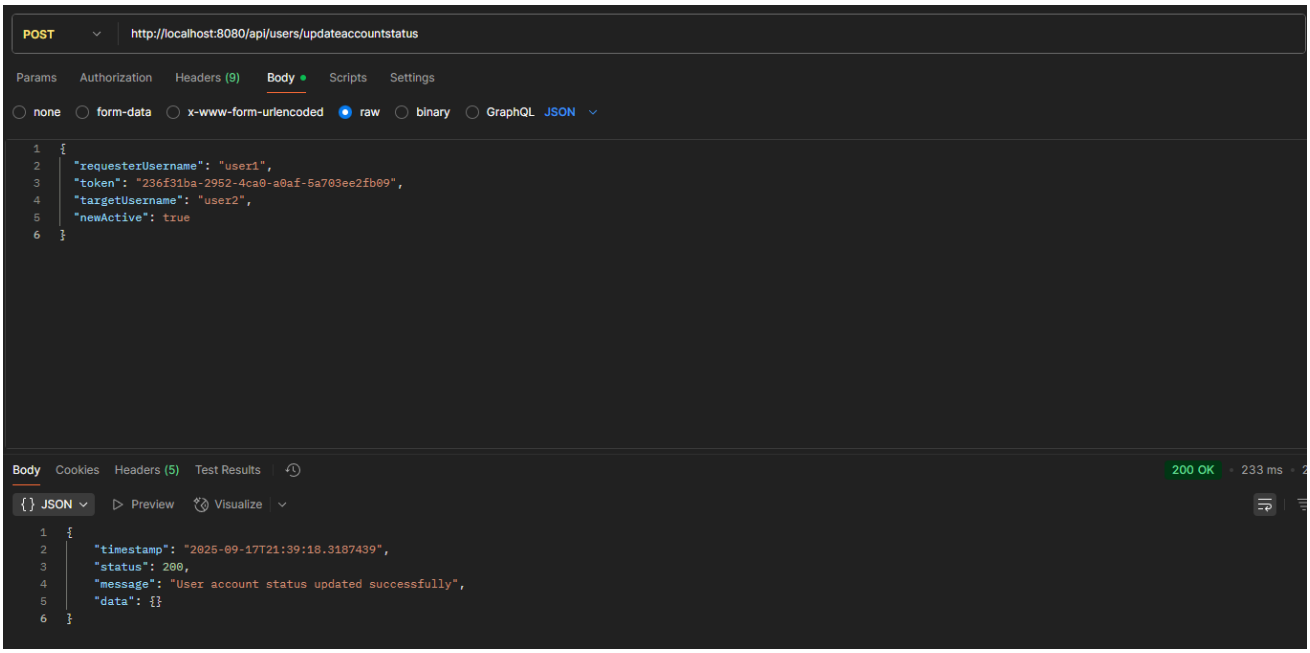
- Εκτελούμε login με τα στοιχεία του πρώτου χρήστη ο οποίος γίνεται και ο admin του συστήματος.



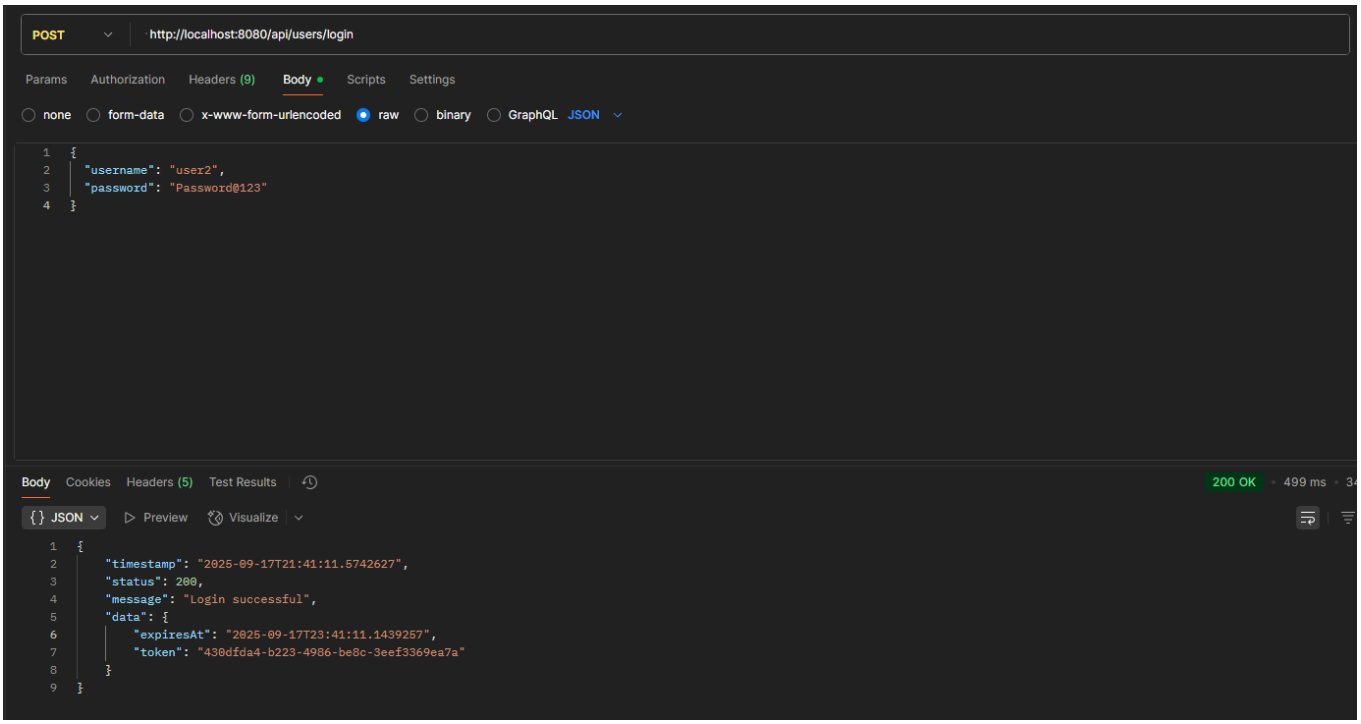
- Ο δεύτερος χρήστη δεν μπορεί να κάνει login γιατί ο λογαριασμός τους δεν είναι ενεργοποιημένος από τον admin.



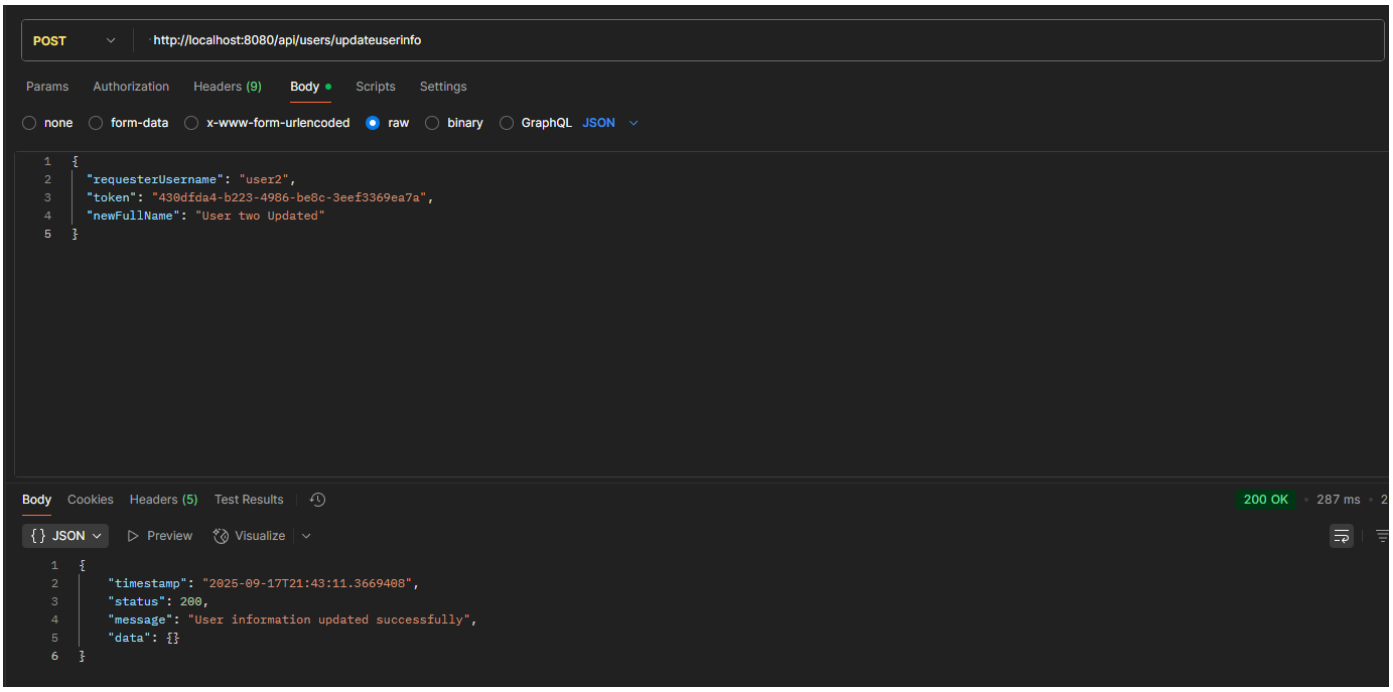
- Βλέπουμε ότι ο χρήστης 2 έχει ενεργοποιηθεί.



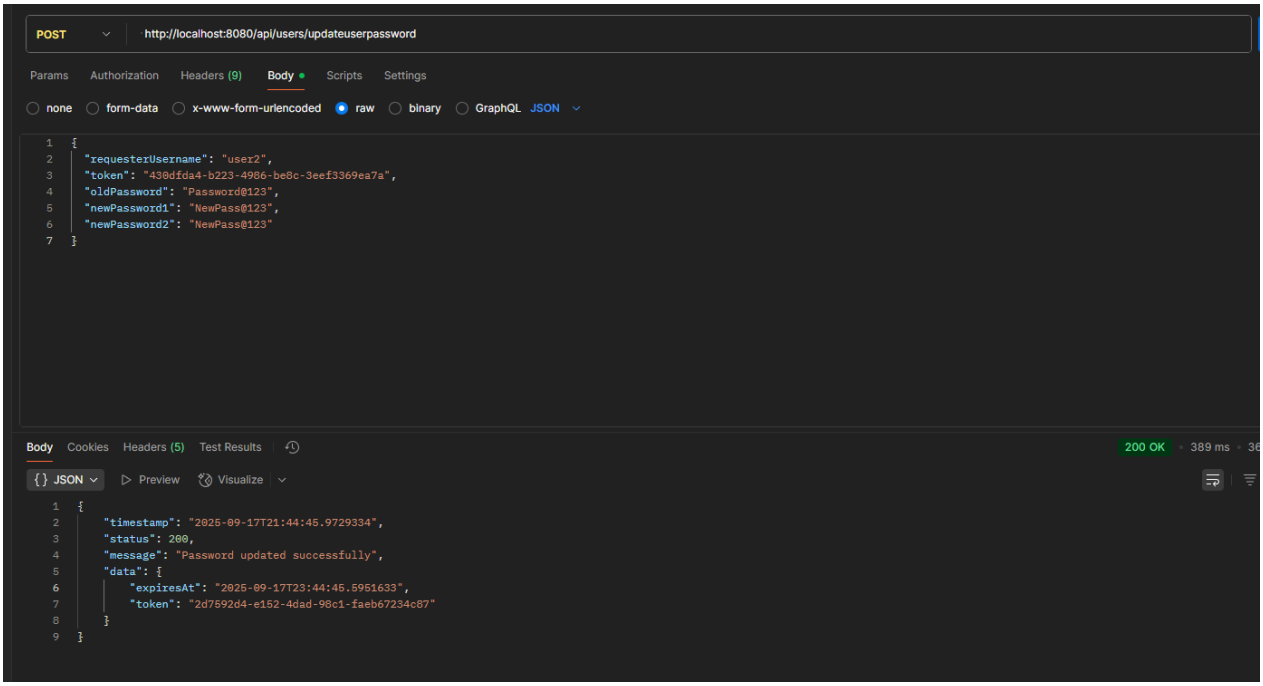
- Ο χρήστης 2 συνδέεται κανονικά και παίρνει το τοκεν του με την ημερομηνία λήξης του.



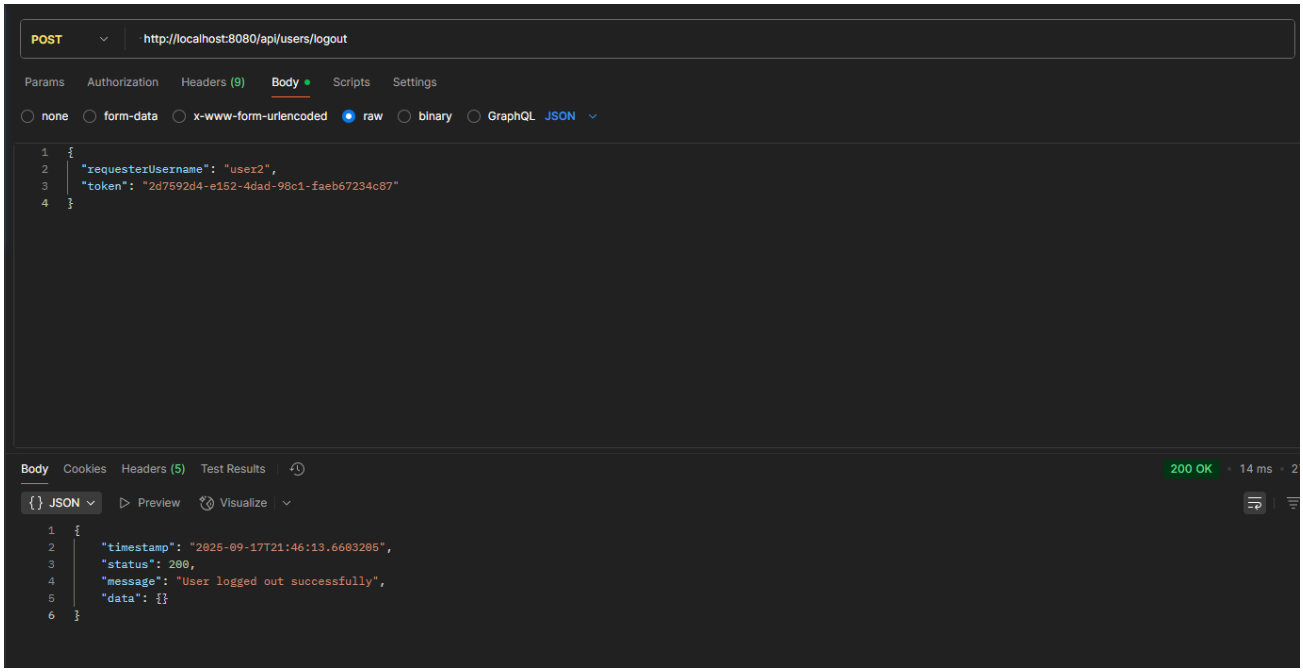
- Κάνουμε αλλαγή του ονόματος του δεύτερου χρήστη.



- Κάνοντας αλλαγή του κωδικού με τα κατάλληλα στοιχεία , παίρνουμε και καινούργιο τοκεν.

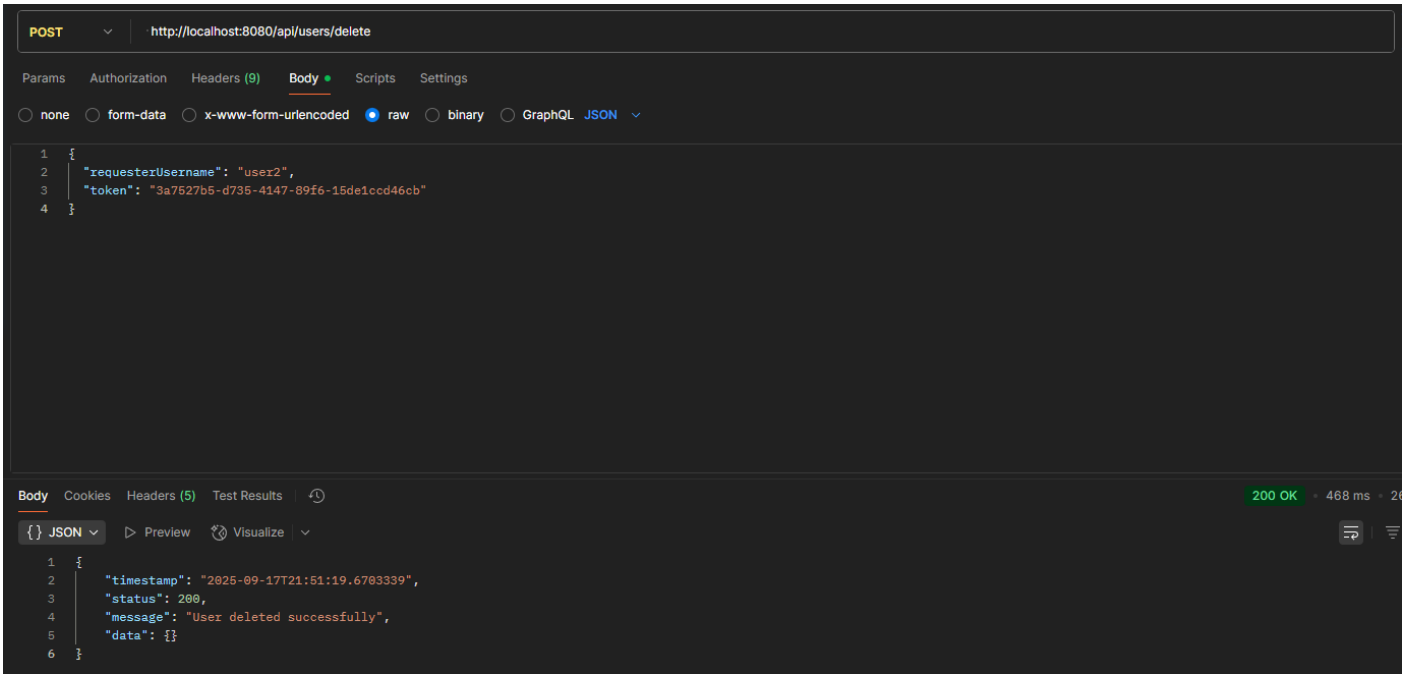


- Βλέπουμε ότι ο χρήστης 2 κάνει επιτυχώς logout.



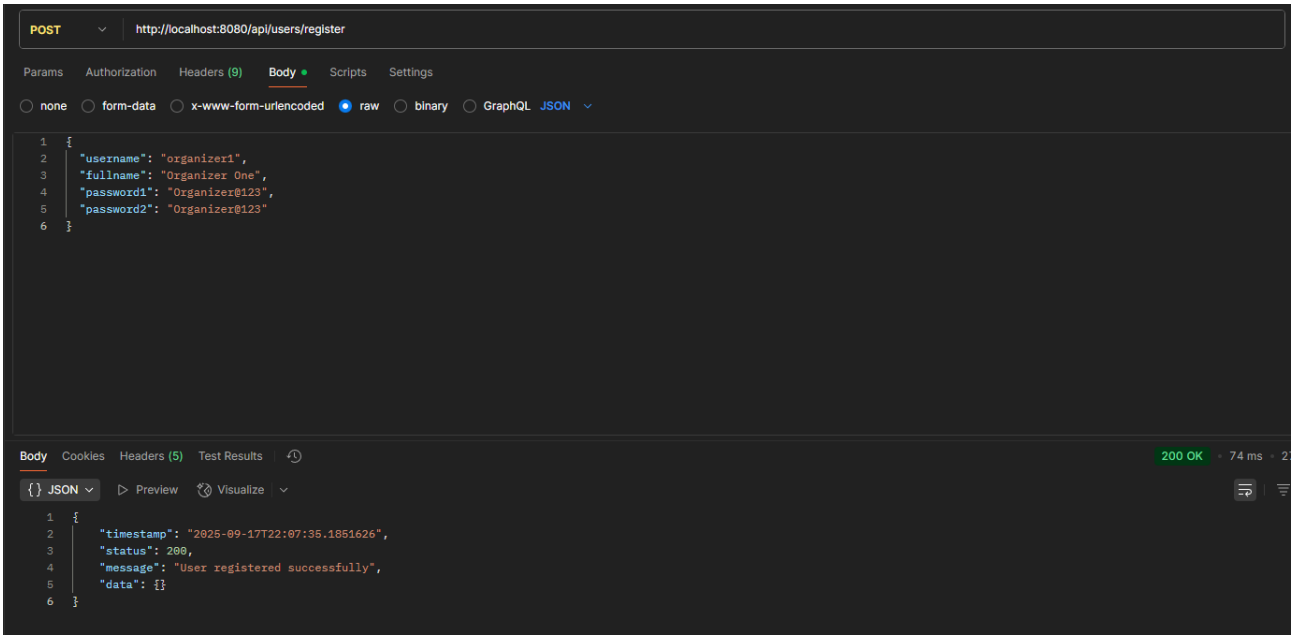
- Ταυτόχρονα με το logout το τοκεν απενεργοποιείται.

- Τέλος βλέπουμε ότι ο χρήστης 2 μπορεί να διαγράψει τον εαυτό του.

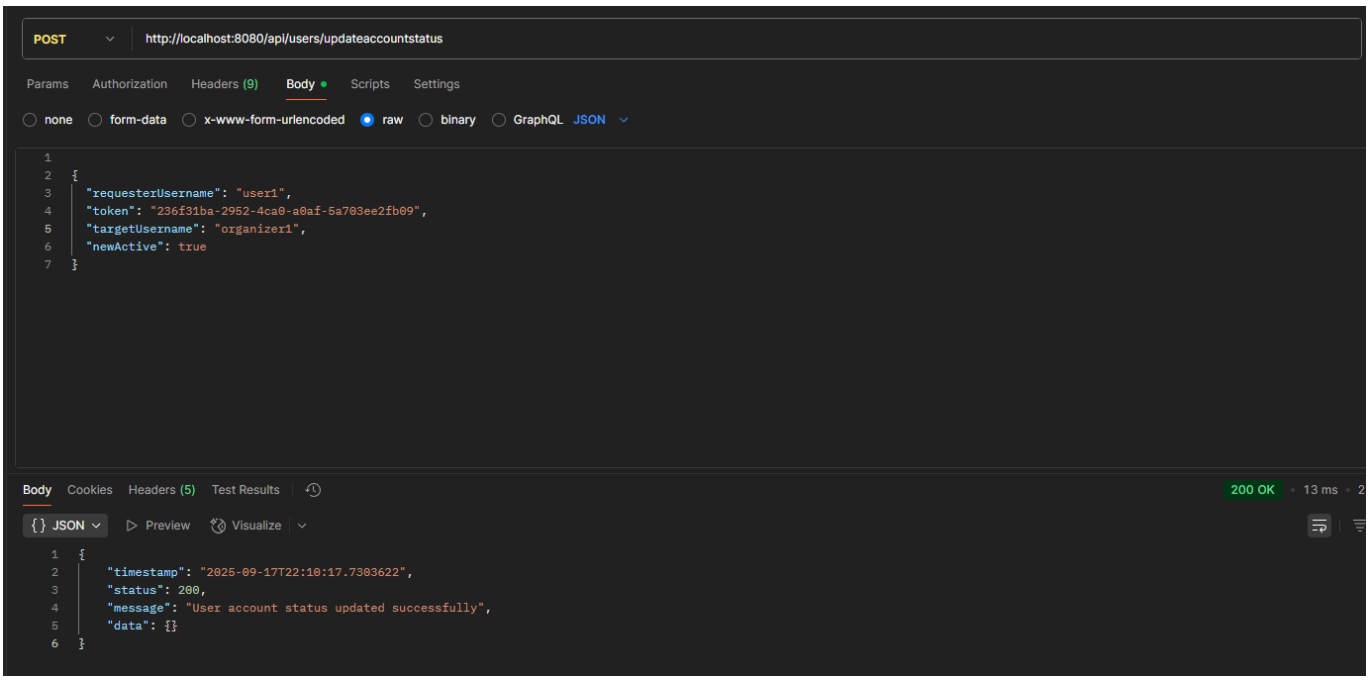


Festival creation and organizer assignment

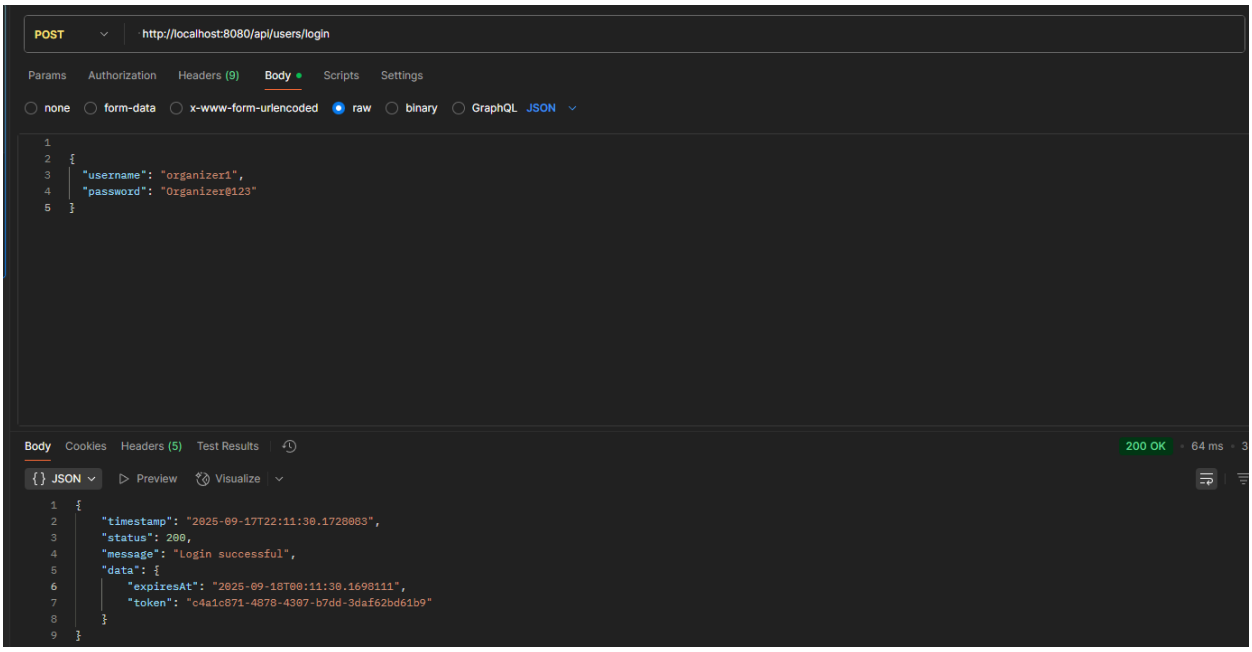
- Αρχικά δημιουργούμε έναν χρήστη ακόμα με νέα στοιχεία.



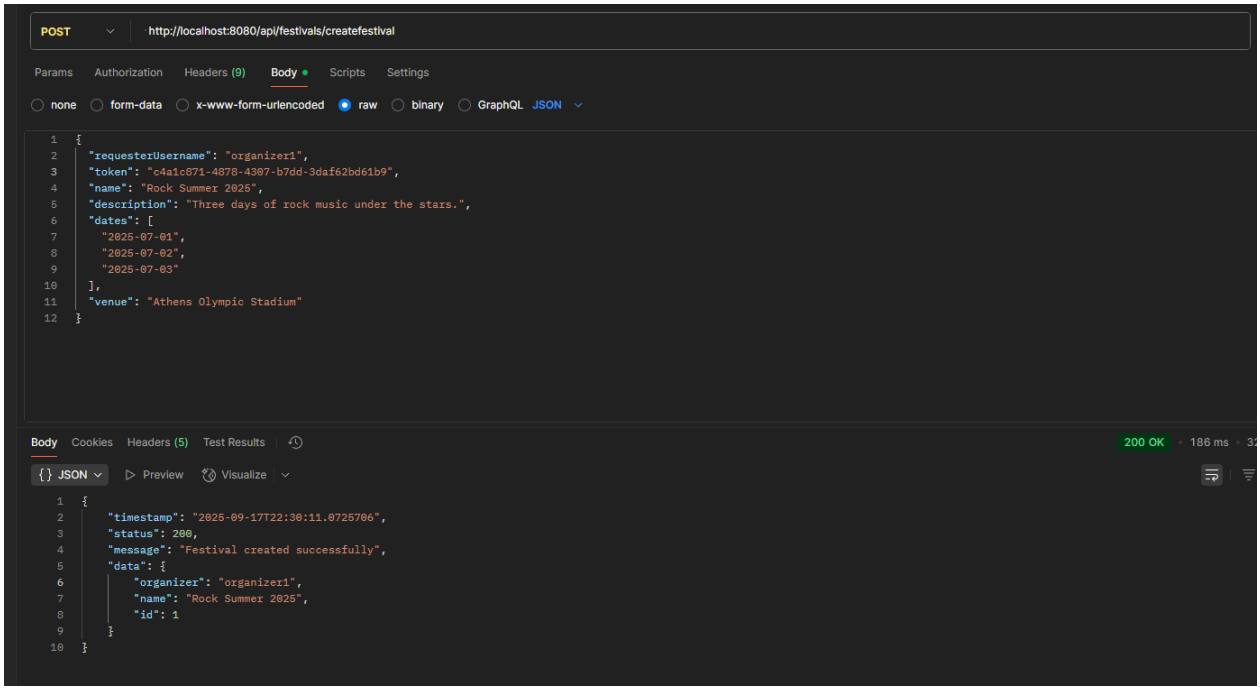
- Κάνουμε activate τον νέο χρήστη.



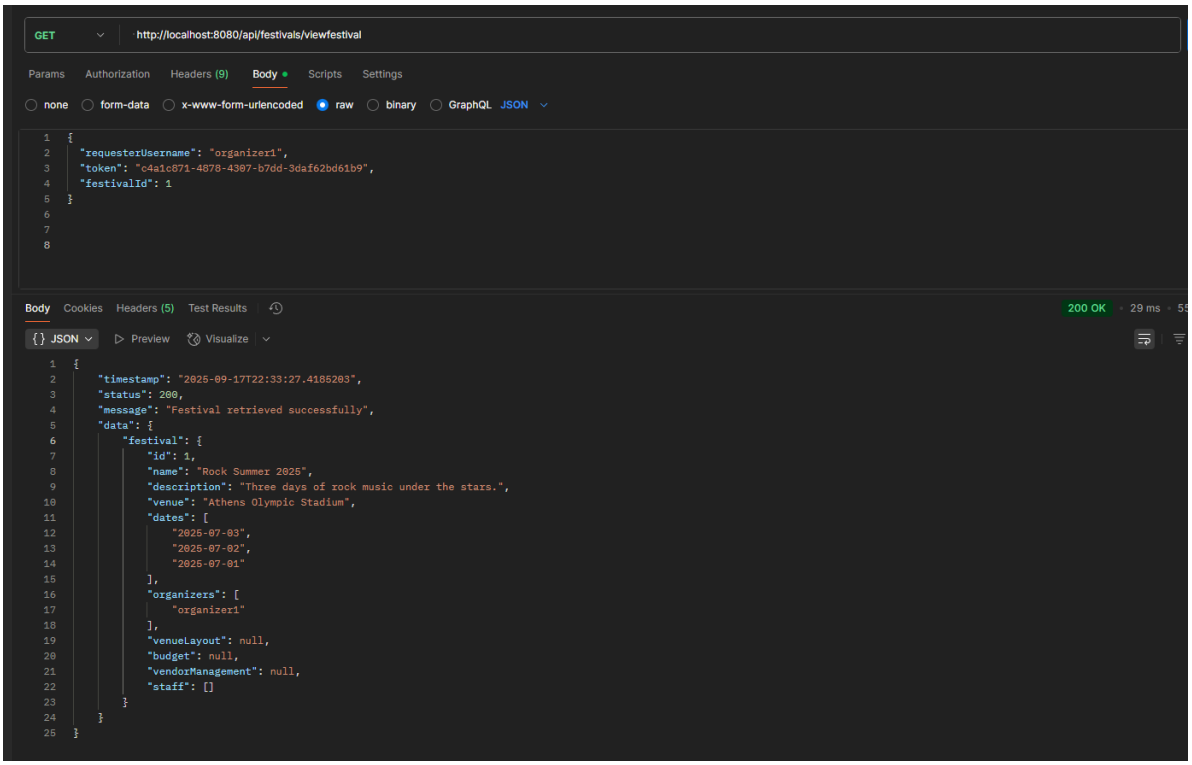
- Κάνουμε σύνδεση με τα χαρακτηριστικά του νέου χρήστη (organizer 1)



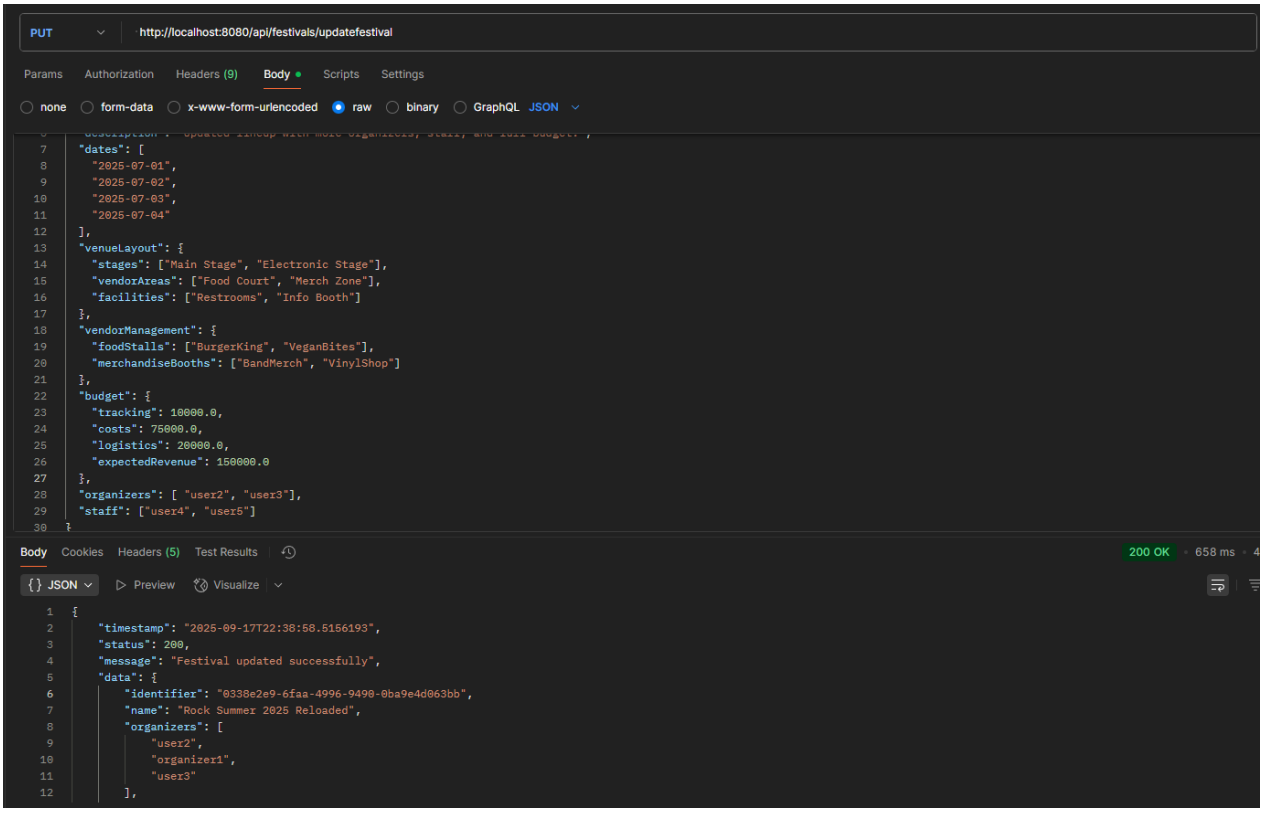
- Δημιουργία φεστιβάλ δίνοντας τα σωστά στοιχεία του φεστιβάλ.



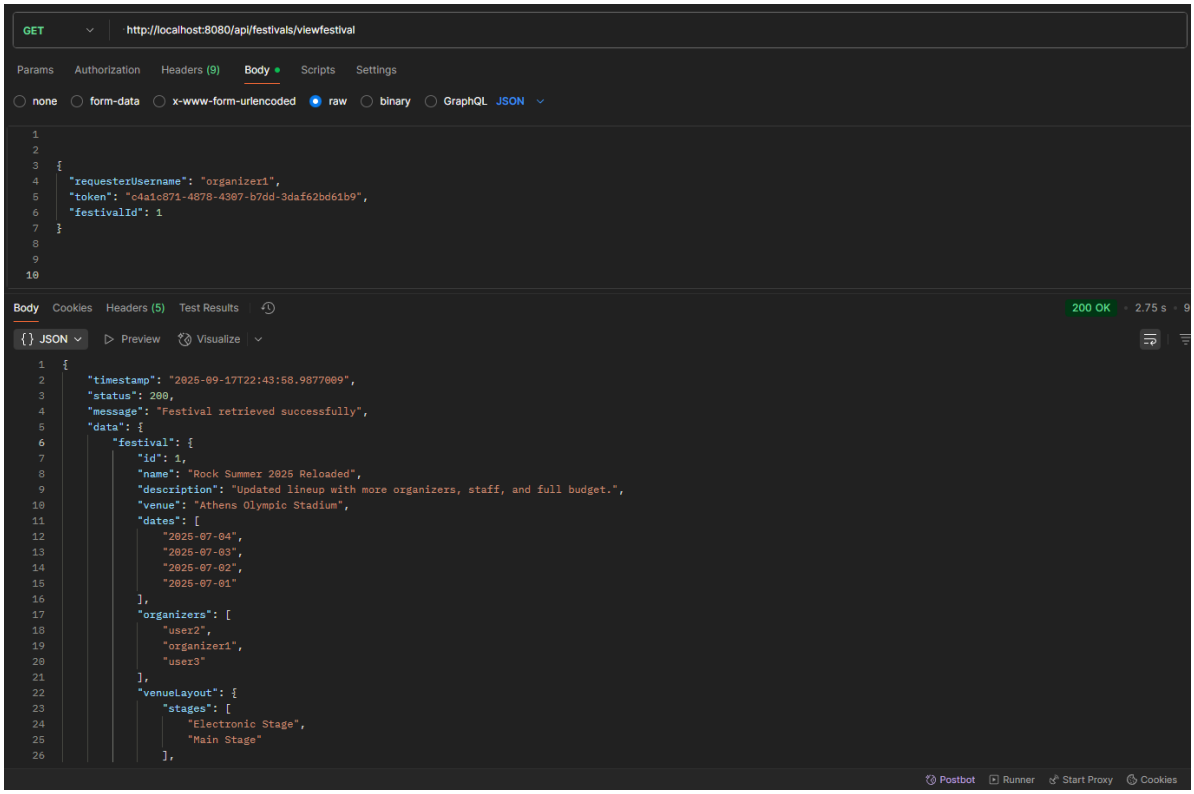
- Βλέπουμε τα στοιχεία του φεστιβάλ με το κατάλληλο κάλεσμα τον στοιχείων.

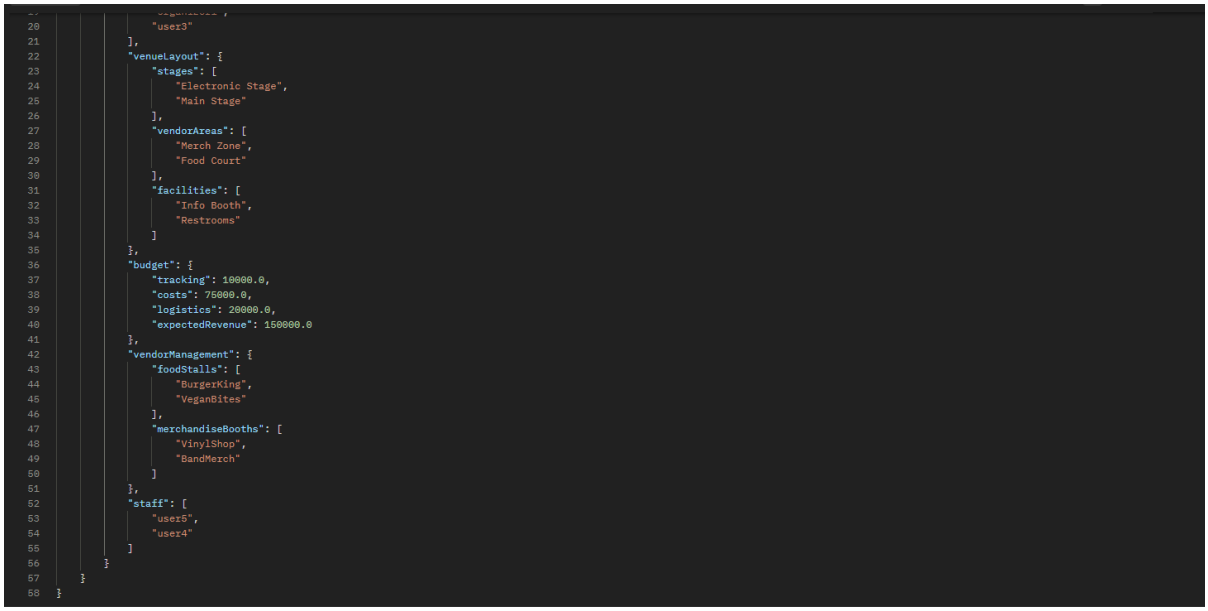


- Πραγματοποιείται αλλαγή στα στοιχεία του φεστιβάλ.



- Εκτελώντας GET στο συγκεκριμένο URL μας επιστρέφονται τα στοιχεία του συγκεκριμένου festival με festivaid: 1.





6 Συμπεράσματα

6.1 Εμπειρία που αποκτήθηκε

Η υλοποίηση του Festival Management System παρείχε πολύτιμη εμπειρία στον σχεδιασμό και την ανάπτυξη ενός σύνθετου πληροφοριακού συστήματος που περιλαμβάνει πολλαπλούς χρήστες, διαφορετικούς ρόλους και διασύνδεση με βάση δεδομένων. Η εργασία επέτρεψε την εξοικείωση με τη μοντελοποίηση μέσω UML διαγραμμάτων, την κατανόηση της αρχιτεκτονικής τριών επιπέδων (Controller–Service–Repository) και τη σημασία του καθαρού διαχωρισμού ευθυνών για την ευκολότερη συντήρηση και επεκτασιμότητα του λογισμικού. Παράλληλα, έγινε σαφές πόσο κρίσιμη είναι η σωστή αποτύπωση των απαιτήσεων, ώστε να μπορούν να μεταφραστούν σε λειτουργικές και μη λειτουργικές προδιαγραφές του συστήματος.

6.2 Προβλήματα και δυσκολίες

Κατά την ανάπτυξη προέκυψαν διάφορες δυσκολίες, κυρίως στην ορθή μοντελοποίηση της επικοινωνίας μεταξύ των components και στη σαφή αποτύπωση των ροών δεδομένων ανάμεσα στους χρήστες και το σύστημα. Επιπλέον, προκλήσεις παρουσιάστηκαν στην αποφυγή επικάλυψης αρμοδιοτήτων μεταξύ διαφορετικών ρόλων χρηστών, όπως του Admin και του Organizer, καθώς και στην ενσωμάτωση των κατάλληλων μηχανισμών ασφάλειας για την αυθεντικοποίηση χρηστών. Τέλος, η επιλογή των κατάλληλων μεθοδολογιών και συμβάσεων UML αποτέλεσε εμπόδιο που ξεπεράστηκε μέσα από συνεχή ανατροφοδότηση και έλεγχο συνέπειας των διαγραμμάτων.

6.3 Βέλτιστες πρακτικές υλοποίησης

Κατά τη διάρκεια του έργου ακολουθήθηκαν ορισμένες βέλτιστες πρακτικές, οι οποίες βελτίωσαν σημαντικά την ποιότητα της τελικής υλοποίησης. Συγκεκριμένα, εφαρμόστηκε σαφής διαχωρισμός ευθυνών μέσω της αρχιτεκτονικής MVC (Model–View–Controller), γεγονός που ενίσχυσε τη modularity και τη δυνατότητα επαναχρησιμοποίησης του κώδικα. Επίσης, η τεκμηρίωση μέσω UML διαγραμμάτων διευκόλυνε την κατανόηση και επικοινωνία των απαιτήσεων και του σχεδιασμού μεταξύ των μελών της ομάδας. Τέλος, η έμφαση στην ασφάλεια χρηστών και δεδομένων αποτέλεσε καθοριστική πρακτική, η οποία μπορεί να εφαρμοστεί και σε μελλοντικά έργα για τη διασφάλιση της αξιοπιστίας και της εμπιστοσύνης των τελικών χρηστών.