



Πανεπιστήμιο Αιγαίου
**Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών
Συστημάτων**

321-4002 – Τεχνολογία Λογισμικού

Διδάσκων: Κυριάκος Κρητικός

Project: Festival Management System

Εργαστηριακός Συνεργάτης: Αλέξανδρος Φακής

321/2019157 Στυλιανός Νικολόπουλος

321/2020010 Γεώργιος Αναγνωστόπουλος

321/2020077 Γεώργιος Καμτσικλής

Σάμος, 21 Σεπτέμβριου 2025



Πίνακας περιεχομένων

1	Περίληψη	3
2	Οργάνωση	4
2.1	Ρόλοι ομάδας	4
2.2	Στάδια Ανάπτυξης	4
2.3	Χρονοδιάγραμμα και Διάρκεια	5
3	Απαιτήσεις Συστήματος	6
3.1	Λειτουργικές Απαιτήσεις	6
3.2	Μη Λειτουργικές Απαιτήσεις	13
4	Σχεδιασμός Συστήματος	14
4.1	Αρχιτεκτονική και Πλαίσιο Συστήματος	14
4.2	Περιπτώσεις Χρήσης	16
4.3	Συμπεριφορά Συστήματος	22
4.3.1	Activity Diagrams	22
4.3.2	Sequence Diagrams	31
4.4	Οντότητες Συστήματος	46
5	Υλοποίηση Συστήματος	49
5.1	GitHub	49
5.2	Τεκμηρίωση εφαρμογής	49
5.3	Τεκμηρίωση δοκιμών	50
5.3.1	UnitTests	50
5.3.2	Postman Testing	61
6	Συμπεράσματα	71
6.1	Εμπειρία που αποκτήθηκε	71
6.2	Προβλήματα και δυσκολίες	71
6.3	Βέλτιστες πρακτικές υλοποίησης	71



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.

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 μια τριμελής ομάδα, όπου το κάθε μέλος 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.



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμπισικλής

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

- 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:

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

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

Λειτουργίες:

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

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

Λειτουργίες:

- Απενεργοποίηση όλων των session tokens του χρήστη.

- 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.



- Άλλαγή full name ή username.
- Έλεγχος διαθεσιμότητας νέου username.
- Επανέκδοση token αν αλλάξει username.

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

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.
- Άλλαγή 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).
- Έλεγχος μοναδικότητας ονόματος festival.
- Αποθήκευση νέου φεστιβάλ στη βάση δεδομένων.



- Ανάθεση ρόλου χρήστη σε organizer για το συγκεκριμένο φεστιβάλ.

9. Ενημέρωση Στοιχείων Φεστιβάλ (Update Festival Info): Στον διοργανωτή του συγκεκριμένου festival επιτρέπεται να ενημερώσει τις πληροφορίες του.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Επιβεβαίωση ότι ο εξουσιοδοτημένος χρήστης είναι organizer στο συγκεκριμένο φεστιβάλ.
- Άλλασή 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.
- Ανάθεση ρόλων χρήστη σε Organizer/Staff για το συγκεκριμένο φεστιβάλ.

10. Διαγραφή Φεστιβάλ (Delete Festival): Στον διοργανωτή του συγκεκριμένου festival δύναται να διαγράψει ένα φεστιβάλ.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Επιβεβαίωση ότι ο εξουσιοδοτημένος χρήστης είναι organizer στο συγκεκριμένο φεστιβάλ.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “Created”.
- Αφαίρεση φεστιβάλ από τη βάση.

11. Αναζήτηση Φεστιβάλ (Search Festival): Οι χρήστες και οι επισκέπτες μπορούν να αναζητούν φεστιβάλ βάσει κριτηρίων.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας (προαιρετική για επισκέπτες).
- Αναζήτηση βάσει ονόματος, περιγραφής, ημερομηνιών και τοποθεσίας.
- Επιστροφή αποτελεσμάτων με βάση τον ρόλο του χρήστη. Για τους επισκέπτες επιστρέφονται μόνο τα ανακοινωμένα φεστιβάλ.

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας (προαιρετική για επισκέπτες).
- Εμφάνιση πληροφοριών για το φεστιβάλ ανάλογα με τον ρόλο χρήστη. Για τους επισκέπτες εμφανίζονται μόνο τα βασικά στοιχεία του φεστιβάλ (όνομα, περιγραφή, ημερομηνίες, τοποθεσία).

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμπικλής

- Έλεγχος δικαιωμάτων.
- Ενημέρωση λίστας διοργανωτών.
- Ανάθεση ρόλου χρήστη σε Organizer για το συγκεκριμένο φεστιβάλ.

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.
- Ανάθεση ρόλου χρήστη σε Staff για το συγκεκριμένο φεστιβάλ.

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

Λειτουργίες:

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

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “Submission”.
- Άλλαγή κατάστασης φεστιβάλ σε “Assignment”.

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “Assignment”.
- Άλλαγή κατάστασης φεστιβάλ σε “Review”.

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “Review”.
- Άλλαγή κατάστασης φεστιβάλ σε “Scheduling”.



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμπικλής

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “ Scheduling ”.
- Άλλαγή 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.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “ Final_Submission ”.
- Αυτόματη απόρριψη των εμφανίσεων που έχουν γίνει αποδεκτές αλλά δεν έχουν οριστικοποιηθεί.
- Άλλαγή κατάστασης φεστιβάλ σε “Decision”.

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

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “ Decision ”.
- Έλεγχος αν όλα τα δευτερεύοντα στοιχεία του φεστιβάλ (Venue Layout, Budget, Vendor Management, λίστες διοργανωτών και προσωπικού) έχουν καταχωρηθεί.
- Άλλαγή κατάστασης φεστιβάλ σε “Announced”.
- Οριστική κατάσταση φεστιβάλ. Δεν πραγματοποιούνται αλλαγές στις πληροφορίες

III. Διαχείριση εμφανίσεων (Performances)

22. Δημιουργία Performance (Create Performance): Ένας εξουσιοδοτημένος χρήστης επιτρέπεται να δημιουργήσει νέο performance σε ένα festival.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση υποχρεωτικών πεδίων: name, description, genre, duration.
- Έλεγχος μοναδικότητας ονόματος performance στο φεστιβάλ.
- Αποθήκευση νέου performance στην βάση δεδομένων.
- Ανάθεση ρόλου χρήστη σε Artist για το συγκεκριμένο φεστιβάλ.



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμπικλής

23. Ενημέρωση Performance (Update Performance): O Artist επιτρέπεται να ενημερώσει στοιχεία του performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Ενημέρωση πεδίων: name, description, genre, duration, bandMembers, technicalRequirements, setlist, merchandiseItems, preferredRehearsalTimes, preferredPerformanceSlots.

24. Προσθήκη Μέλους Μπάντας (Add Band Member): O Artist επιτρέπεται να προσθέσει νέο μέλος σε ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι το νέο μέλος υπάρχει στο σύστημα και δεν είναι ήδη μέλος της μπάντας.
- Προσθήκη του νέου μέλους στο performance.
- Ανάθεση ρόλου χρήστη σε Artist για το συγκεκριμένο φεστιβάλ.

25. Υποβολή Performance (Submit Performance): O Artist επιτρέπεται να κάνει υποβολή του συγκεκριμένου performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι έχουν καταχωρηθεί όλα τα στοιχεία ενός performance.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “submit”.
- Άλλαγή κατάστασης performance σε “submited”.

26. Απόσυρση Performance (Withdraw Performance): O/oi Artist/s επιτρέπεται να αποσύρει το performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του performance είναι πριν την κατάσταση “submited”.
- Διαγραφή του performance από την βάση δεδομένων.

27. Ανάθεση Staff (Assign Staff): O organizer του συγκεκριμένου φεστιβάλ στο οποίο απευθύνεται το performance αναθέτει σε έναν από τους Staff του, την διαχείριση του συγκεκριμένου Performance .

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος ότι ο χρήστης είναι staff.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “assigment”.
- Ανάθεση του staff στο performance.



28. Αξιολόγηση Performance (Review Performance): Ο Staff στον οποίο έχει ανατεθεί το συγκεκριμένο performance δύναται να το αξιολογήσει.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “review”.
- Καταχώρηση score και reviewerComments.
- Άλλαγή κατάστασης performance σε “reviewed”.

29. Έγκριση Performance (Approve Performance): Ο organizer του festival μπορεί να εγκρίνει ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “scheduling”.
- Άλλαγή κατάστασης performance σε “approved”.

30. Απόρριψη Performance (Reject Performance): Ο organizer μπορεί να απορρίψει ένα performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση rejectionReason (υποχρεωτικό).
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “scheduling / decision”.
- Άλλαγή κατάστασης performance σε “rejected”.

31. Οριστική Υποβολή Performance (Final Submission): Ο Artist μπορεί να οριστικοποιήσει το performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Καταχώρηση τελικής setlist, rehearsalTimes και performanceTimeSlots.
- Συμφόρηση ανάλογα με τις κρητικές (reviews) του αντίστοιχου Staff.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “final_submission”.
- Οριστική κατάσταση πληροφοριών Performance. Δεν πραγματοποιούνται αλλαγές.

32. Αναζήτηση Performance (Search performance): Οι χρήστες και οι επισκέπτες μπορούν να αναζητούν performance βάσει κριτήριων.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας (προαιρετική για επισκέπτες).
- Αναζήτηση βάσει ονόματος, ειδους και καλλιτέχνη.
- Επιστροφή αποτελεσμάτων με βάση τον ρόλο του χρήστη. Για τους επισκέπτες επιστρέφονται μόνο τα προγραμματισμένα performances.



33. Προβολή Performance (View Performance): Οι χρήστες και οι επισκέπτες επιτρέπεται να δουν λεπτομέρειες ενός performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας (προαιρετική για επισκέπτες).
- Εμφάνιση πληροφοριών για το performance ανάλογα με τον ρόλο χρήστη. Για τους επισκέπτες εμφανίζονται μόνο τα βασικά στοιχεία του performance (όνομα, περιγραφή, είδος, διάρκεια, καλλιτέχνες).

34. Έγκριση performance (Performance Acceptance): Ο organizer εγκρίνει το performance.

Λειτουργίες:

- Επιβεβαίωση ταυτότητας μέσω token.
- Έλεγχος αν η κατάσταση του φεστιβάλ είναι “Decision”.
- Άλλαγή κατάστασης performance σε “Scheduled”.
- Οριστική κατάσταση performance.

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

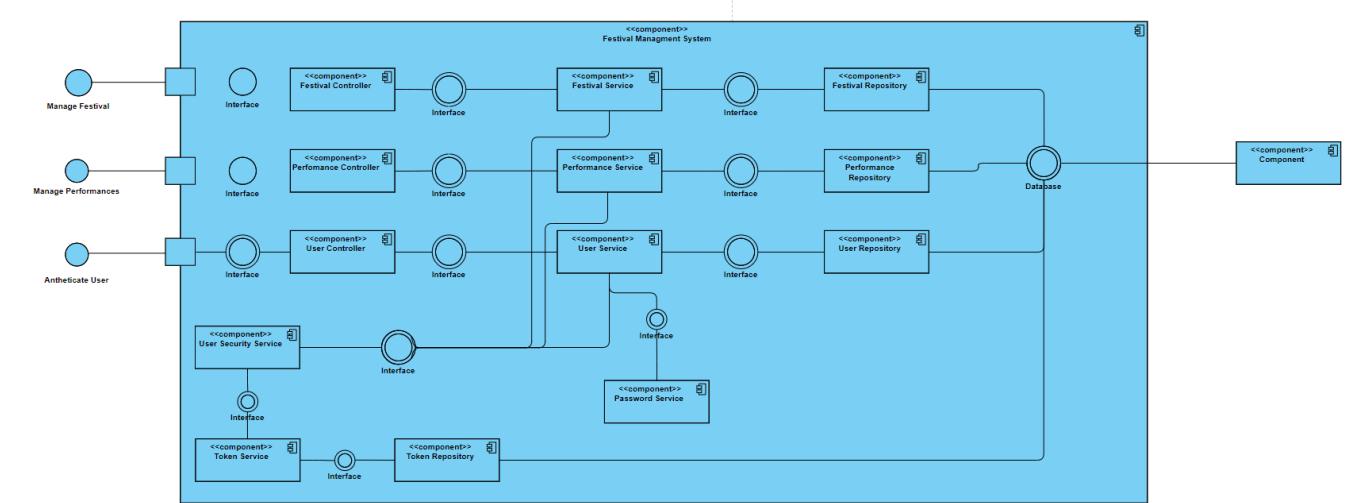
- Το σύστημα διασφαλίζει την ασφαλή διαχείριση passwords (pattern check, strong password rules, hashed).
- Modular design: separation of concerns (Controllers, Services, DTOS, DAO).
- Επαλήθευση των tokens των χρηστών σε κάθε αίτημα.



4 Σχεδιασμός Συστήματος

4.1 Αρχιτεκτονική και Πλαίσιο Συστήματος

Component Diagram



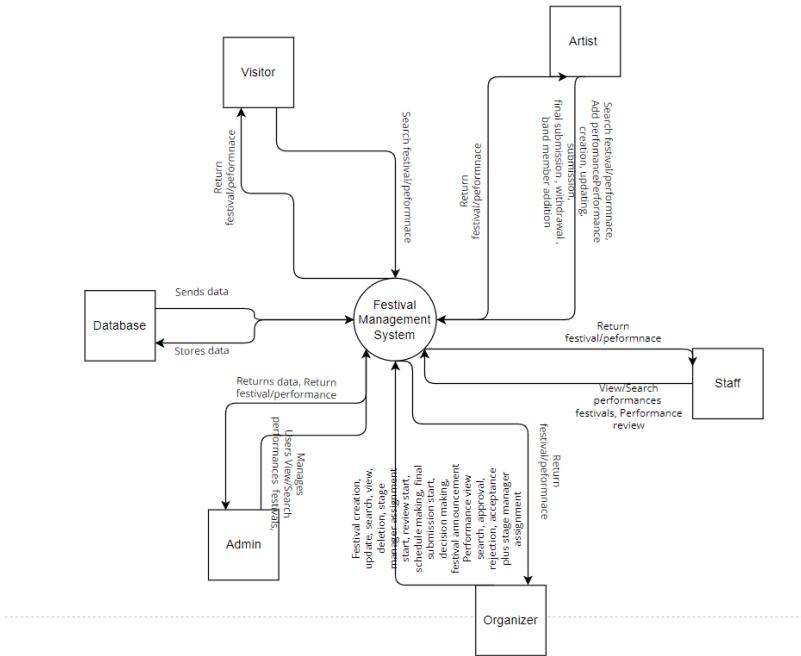
Το διάγραμμα παρουσιάζει την αρχιτεκτονική του Festival Management System. Το σύστημα αποτελείται από τρεις βασικές λειτουργικές ενότητες: διαχείριση φεστιβάλ, διαχείριση παραστάσεων και διαχείριση χρηστών.

Κάθε ενότητα περιλαμβάνει έναν Controller, ένα Service και ένα Repository που επικοινωνούν μεταξύ τους ακολουθώντας την αρχιτεκτονική τριάντων 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 service για τον έλεγχο ασφάλειας των χρηστών (User Security), την ασφάλεια των κωδικών (Password Service) και για την διαχείριση των Token (Token Service). 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:

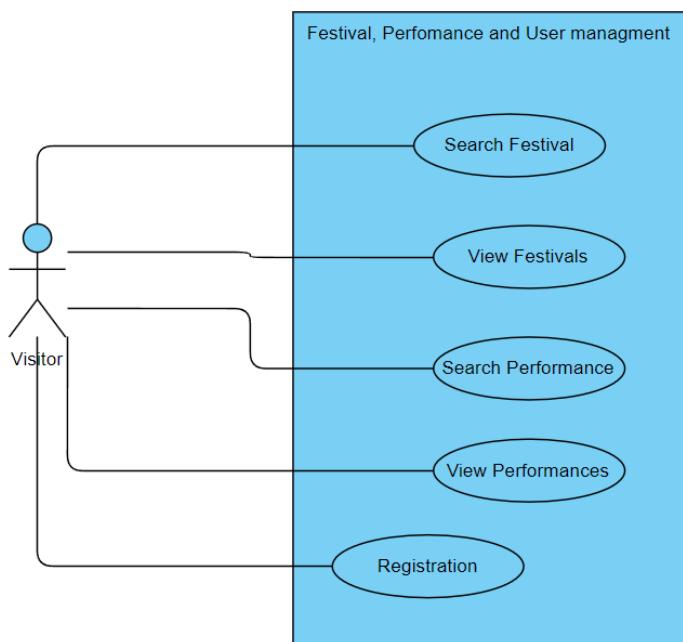
- **Visitor**: Ανώνυμος χρήστης που μπορεί μόνο να αναζητά και να βλέπει βασικές πληροφορίες για φεστιβάλ και προγραμματισμένες παραστάσεις. Η πρόσβαση του είναι περιορισμένη σε στοιχεία όπως όνομα, ημερομηνίες, χώρος, περιγραφή και βασικά χαρακτηριστικά παραστάσεων.
- **Artist**: Αυθεντικοποιημένος χρήστης που δημιουργεί ή συμμετέχει σε παραστάσεις ενός φεστιβάλ. Μπορεί να καταχωρεί, ενημερώνει, υποβάλει ή αποσύρει παραστάσεις και να προσθέτει μέλη συγκροτήματος. Έχει πρόσβαση σε όλες τις λεπτομέρειες των δικών του παραστάσεων, συμπεριλαμβανομένων αξιολογήσεων και σχολίων.
- **Organizer**: Υπεύθυνος για τη δημιουργία και τη διαχείριση ενός φεστιβάλ. Μπορεί να εγκρίνει, απορρίπτει ή αποδεχτεί παραστάσεις και να αναθέτει stage managers. Έχει πλήρη πρόσβαση σε όλα τα στοιχεία του φεστιβάλ που διαχειρίζεται καθώς και στις παραστάσεις του.
- **Staff**: Αυθεντικοποιημένος χρήστης που ορίζεται για συγκεκριμένες παραστάσεις ενός φεστιβάλ. Μπορεί να δει όλες τις λεπτομέρειες μόνο αυτών των παραστάσεων και να συμμετέχει στη διαδικασία αξιολόγησης. Έχει επίσης τα δικαιώματα ενός Visitor.
- **Admin**: Διαχειριστής συστήματος με όλα τα δικαιώματα ενός Visitor, καθώς και πρόσβαση στη διαχείριση χρηστών (δημιουργία, ενημέρωση, διαγραφή, έλεγχος ρόλων). Δεν αποτελεί “super-user” του συστήματος για φεστιβάλ/παραστάσεις, αλλά είναι υπεύθυνος για το user management.



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:

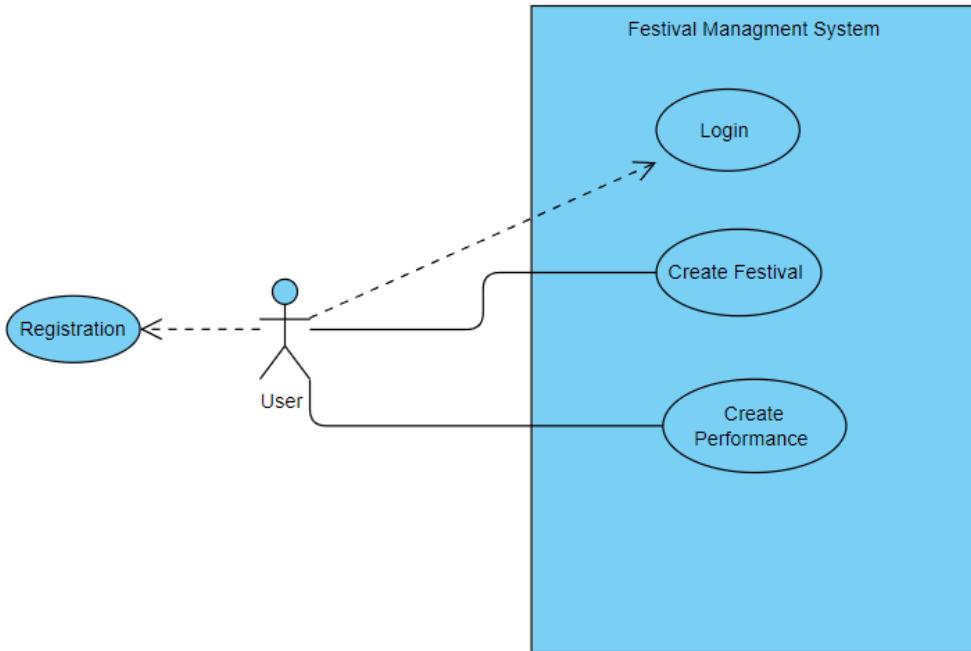
Visitor (Επισκέπτης): Ο βασικός actor που αλληλεπιδρά με το σύστημα, με στόχο την αναζήτηση και προβολή πληροφοριών για φεστιβάλ και παραστάσεις.

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.
5. Registration: Εγγραφή του χρήστη στο σύστημα.



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 δίνονται στον user αφού έχει εκτελέσει τις αντίστοιχες διαδικασίες. Επίσης, μέσω του ρόλου Visitor υποστηρίζεται η εγγραφή του στο σύστημα.

Συμμετέχοντες:

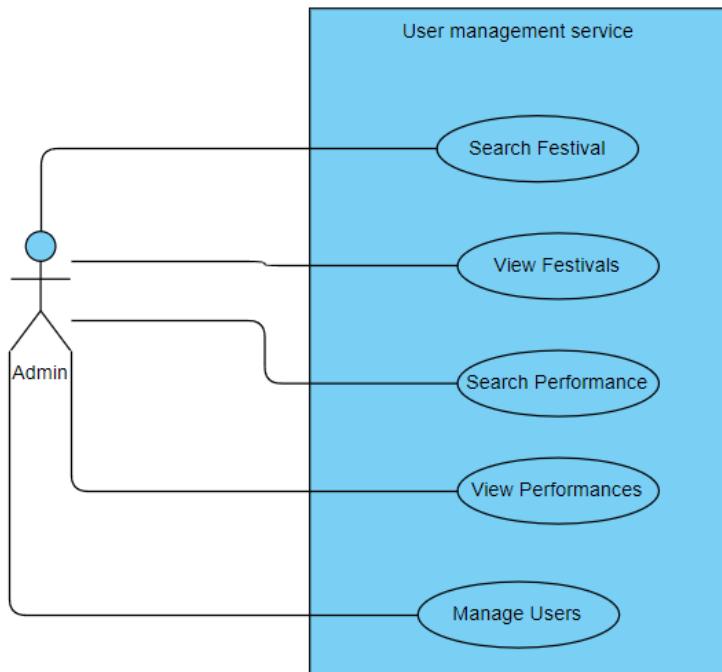
- Visitor (Επισκέπτης): Μπορεί να πραγματοποιήσει εγγραφή (Registration) για να γίνει User.
- User (Χρήστης): Δημιουργεί φεστιβάλ και παραστάσεις.
- Organizer (Διοργανωτής): Συνδέεται με τη δημιουργία φεστιβάλ.
- Artist (Καλλιτέχνης): Συνδέεται με τη δημιουργία παραστάσεων.
- Staff (Προσωπικό): Συνδέεται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



Περιγραφή διαγράμματος:

Το διάγραμμα παρουσιάζει τις βασικές λειτουργίες που μπορεί να εκτελέσει ένας διαχειριστής (Admin). Ο διαχειριστής έχει τη δυνατότητα να δει και να αναζητήσει φεστιβάλ και παραστάσεις και να διαχειριστεί τους υπόλοιπους χρήστες τους συστήματος.

Συμμετέχοντες:

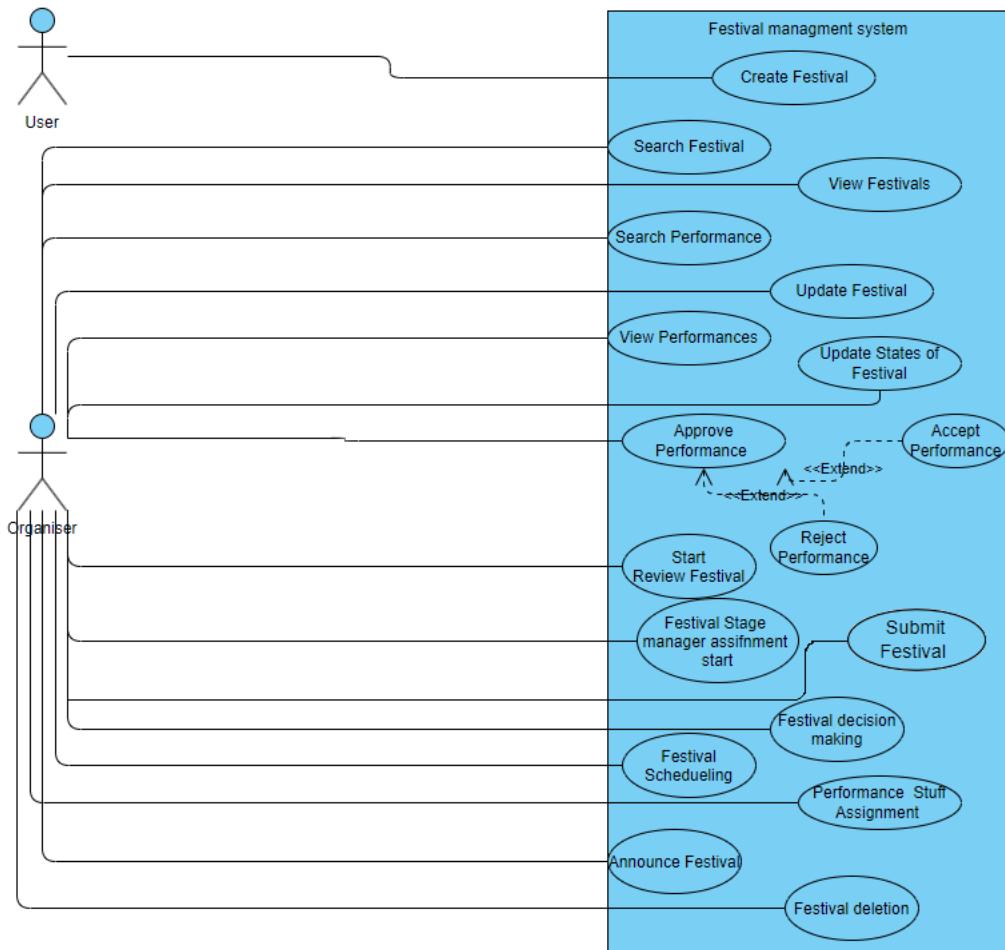
- Admin (διαχειριστής): Μπορεί να δει και να αναζητήσει φεστιβάλ και παραστάσεις και να διαχειριστεί τους υπόλοιπους χρήστες τους συστήματος.

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.
5. Manage Users: Μπορεί να διαχειριστεί τους υπόλοιπους χρήστες του συστήματος.



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.

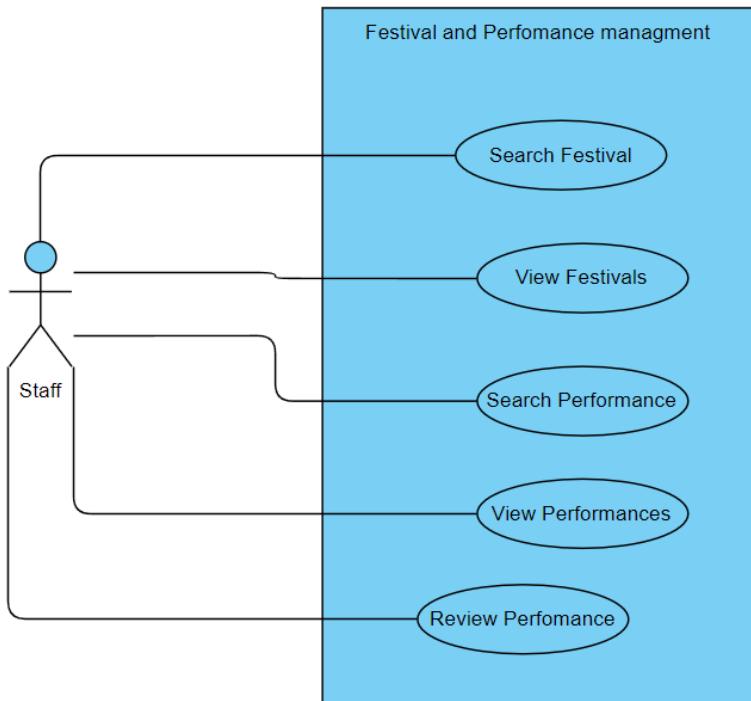
Χρήσεις (Use Cases):

1. Search festival / performance: Αναζήτηση φεστιβάλ / παραστάσεων.
2. View festival / performance: Προβολή λεπτομερειών για ένα φεστιβάλ / μία παράσταση.
3. Update festival: Ενημέρωση των πληροφοριών του φεστιβάλ.
4. Update states of festival: Ενημέρωση της κατάστασης του φεστιβάλ.
5. Approve performance: Έγκριση performance που υποβάλλουν οι καλλιτέχνες.
6. Accept performance: Αποδοχή του performance που έχει υποβληθεί.
7. Reject performance: Απόρριψη performance που δεν πληρούν τα κριτήρια.
8. Submit festival: Υποβολή του φεστιβάλ.



9. Start review festival: Αξιολόγηση του φεστιβάλ από τον διοργανωτή.
10. Festival stage manager assignment start: Ενεργοποίηση της διαδικασίας ανάθεσης stage managers.
11. Festival decision making: Έναρξη και καταγραφή της τελικής απόφασης για το φεστιβάλ.
12. Festival scheduling: Έναρξη δημιουργίας του προγράμματος του φεστιβάλ.
13. Performance staff assignment: Ανάθεση ρόλων/καθηκόντων στο προσωπικό (Staff).
14. Announce festival: Επίσημη ανακοίνωση του φεστιβάλ.
15. Festival deletion: Διαγραφή του φεστιβάλ από την βάση δεδομένων.

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:

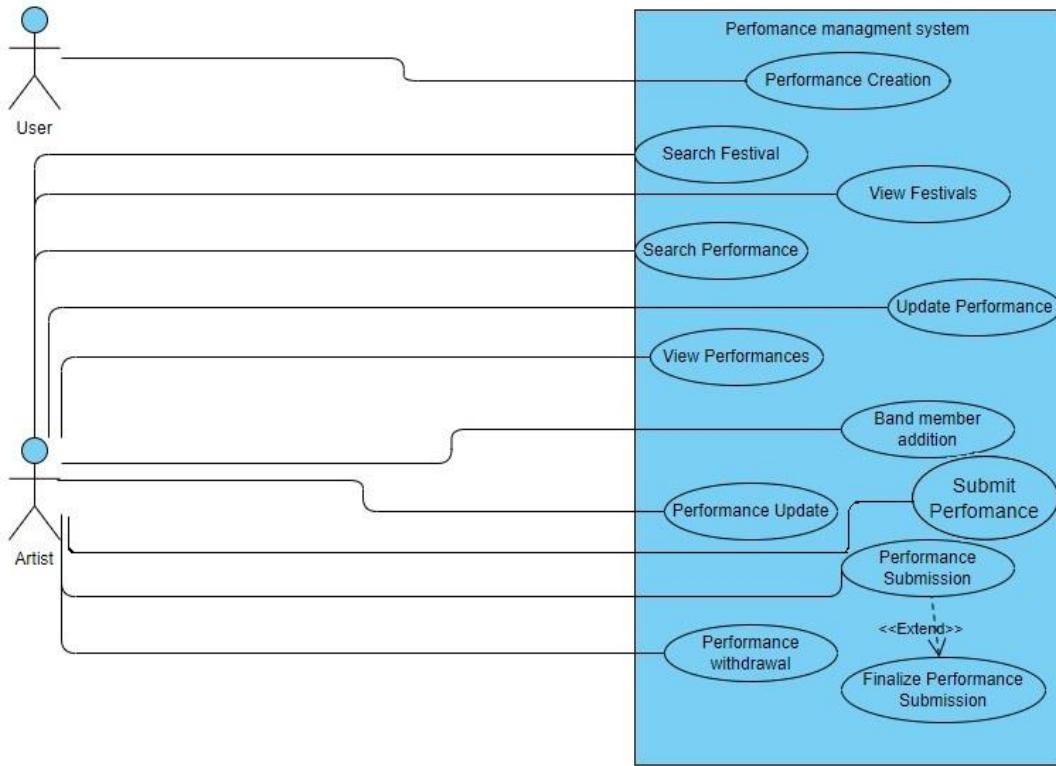
Staff (Προσωπικό): Actor που έχει αρμοδιότητες σχετικά με την οργάνωση και τροποποίηση πληροφοριών των φεστιβάλ.

Χρήσεις (Use Cases):

1. Search festival / Performance: Αναζήτηση φεστιβάλ / παραστάσεων στο σύστημα.
2. View festival / Performance: Προβολή φεστιβάλ / παραστάσεων.
3. Review Performance: Αξιολόγηση παράστασης η οποία του έχει ανατεθεί.



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.
7. Submit Performance: Υποβολή της παράστασης.

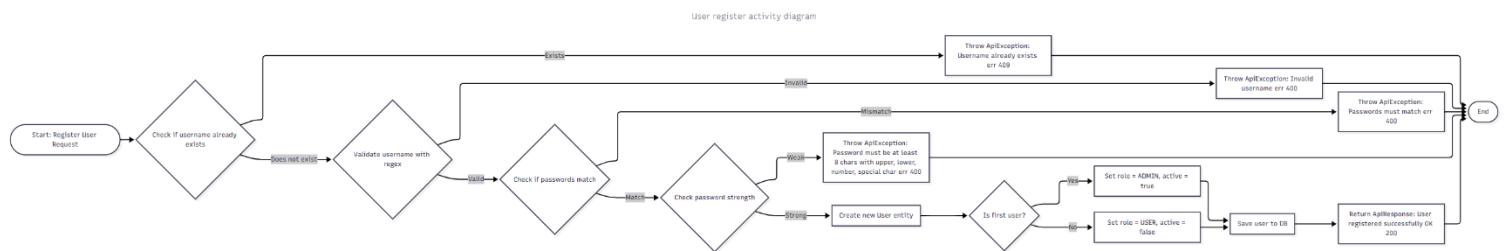


4.3 Συμπεριφορά Συστήματος

4.3.1 Activity Diagrams

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.

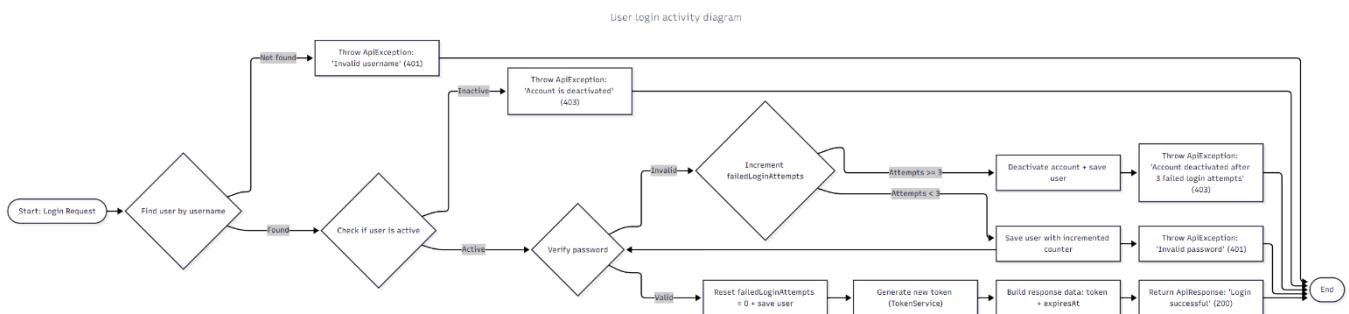
Activity Diagram – registerUser:



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.

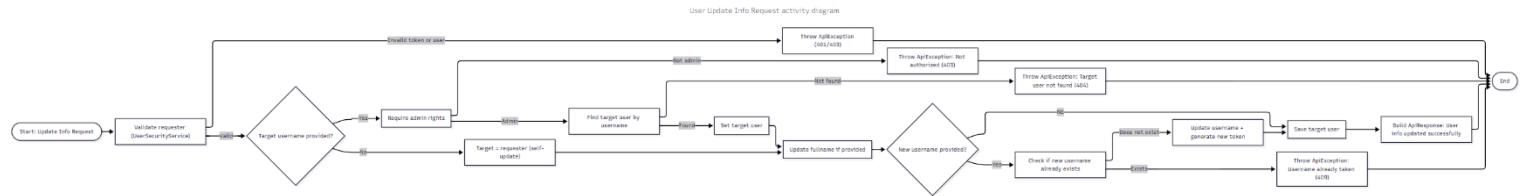
Activity Diagram – userlogin:



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.

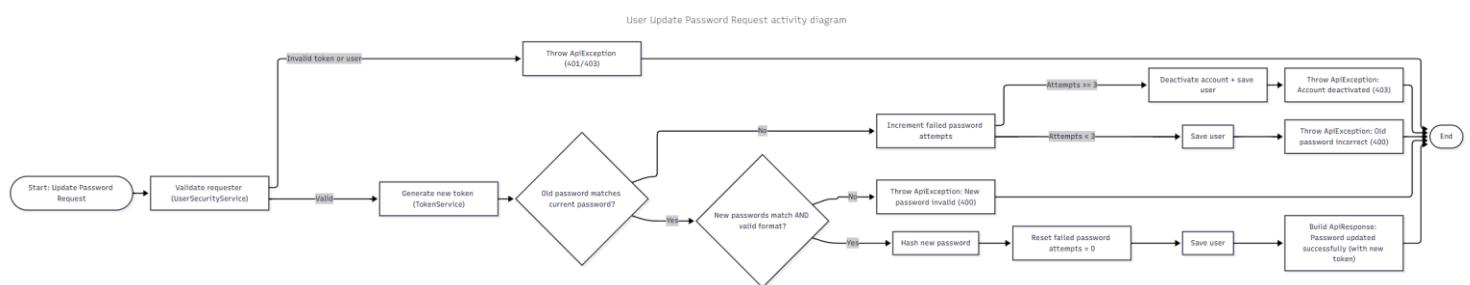


Activity Diagram – updateUserInfo:



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. Άλλιώς καλείται η εξαίρεση. Στο τέλος αποθηκεύονται οι αλλαγές και επιστρέφεται μήνυμα επιτυχίας.

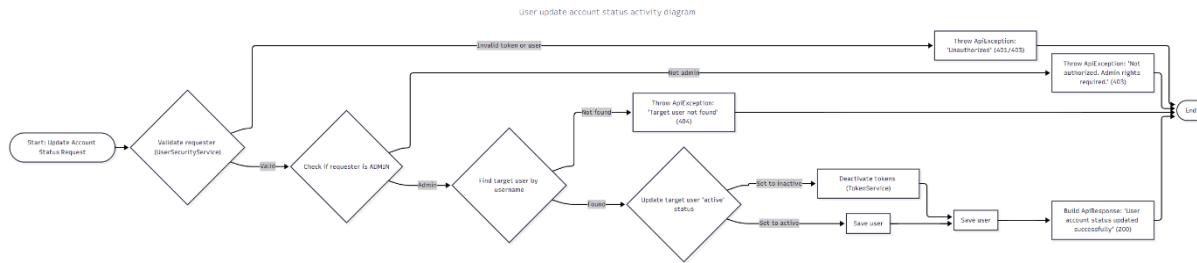
Activity Diagram – updateUserPassword:



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 προσπαθειες, ο λογαριασμός απενεργοποιείται και εκκινείται σχετική εξαίρεση. Αν οι αποτυχημένες προσπαθειες 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.

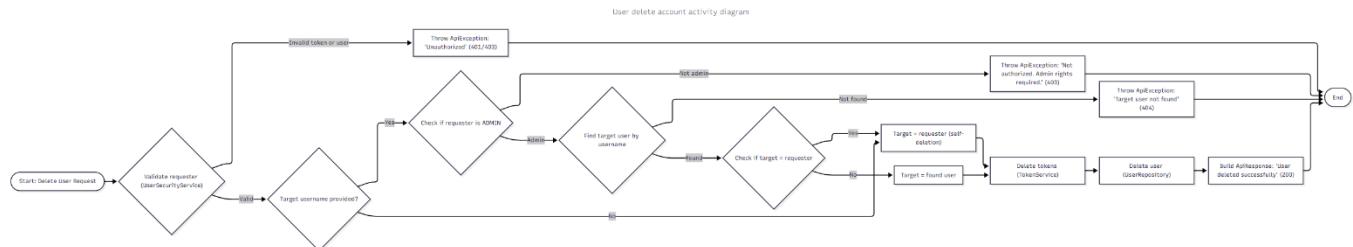


Activity Diagram – updateAccount:



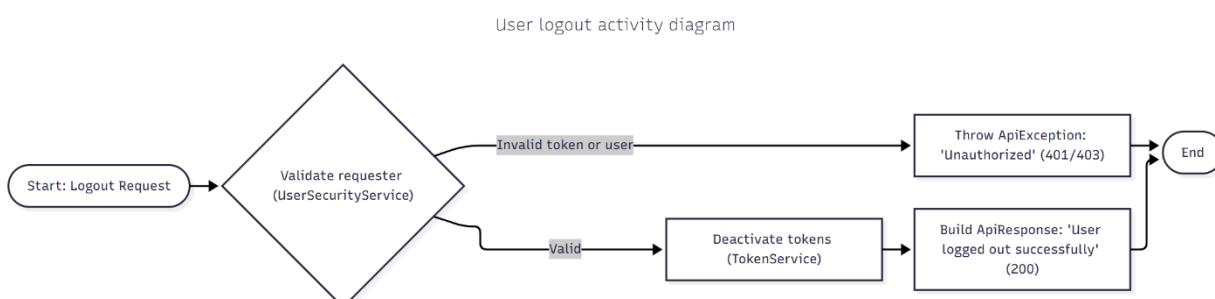
Το διάγραμμα παρουσιάζει τη διαδικασία ενημέρωσης της κατάστασης λογαριασμού χρήστη. Αρχικά ελέγχεται η εγκυρότητα του requester. Αν δεν είναι Admin ή το token είναι άκυρο, επιστρέφεται σφάλμα. Αν ο Admin ζητά αλλαγή, αναζητείται ο target username. Αν βρεθεί, ενημερώνεται η κατάσταση (active/inactive) 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». 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». Αν είναι Admin, αναζητείται ο target user. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Target user not found». Αν βρεθεί, καθορίζεται ο χρήστης που θα διαγραφεί. Ακολουθεί διαγραφή των tokens και του χρήστη, και επιστρέφεται μήνυμα επιτυχίας «User deleted successfully».

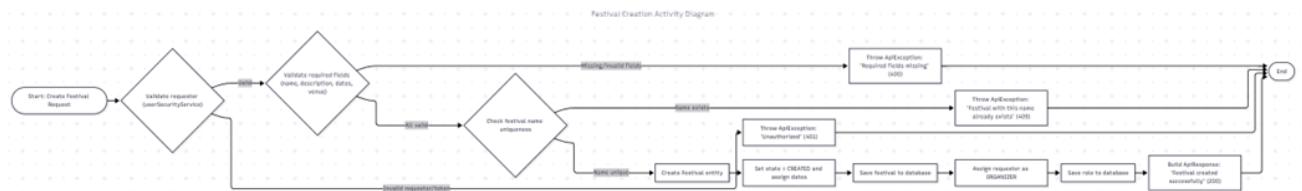
Activity Diagram – User logout:





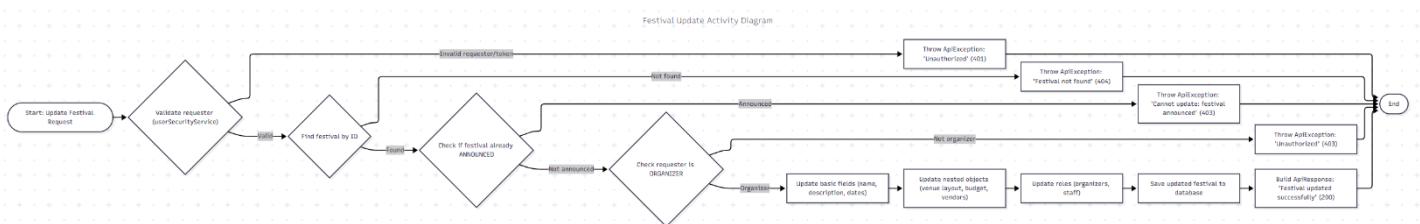
Το διάγραμμα παρουσιάζει τη διαδικασία αποσύνδεσης χρήστη (logout). Αρχικά ελέγχεται η εγκυρότητα του requester. Αν το token είναι άκυρο ή ο χρήστης μη εξουσιοδοτημένος, επιστρέφεται σφάλμα «Unauthorized». Αν ο χρήστης είναι έγκυρος, απενεργοποιούνται τα tokens του και επιστρέφεται μήνυμα επιτυχίας «User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

Activity Diagram – CreateFestival:



Το διάγραμμα παρουσιάζει τη διαδικασία δημιουργίας φεστιβάλ. Αρχικά ελέγχεται η εγκυρότητα του requester μέσω του token. Αν είναι άκυρο, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια ελέγχονται τα υποχρεωτικά πεδία (όνομα, περιγραφή, ημερομηνίες, χώρος). Αν λείπει κάποιο, επιστρέφεται σφάλμα «Required fields missing». Έπειτα γίνεται έλεγχος μοναδικότητας του ονόματος: αν υπάρχει ήδη, επιστρέφεται σφάλμα «Festival with this name already exists». Αν όλα είναι έγκυρα, δημιουργείται το φεστιβάλ με κατάσταση CREATED, αποθηκεύεται στη βάση, ο requester ανατίθεται ως ORGANIZER, και επιστρέφεται μήνυμα επιτυχίας «Festival created successfully» με id, όνομα και organizer.

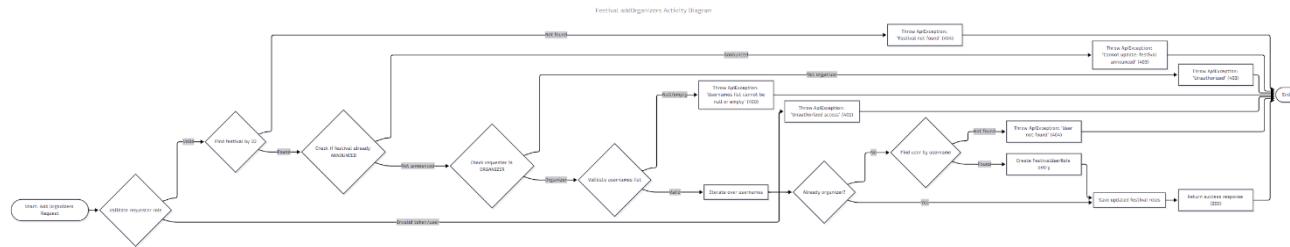
Activity Diagram – update festival:



Το διάγραμμα παρουσιάζει τη διαδικασία ενημέρωσης ενός φεστιβάλ. Αρχικά ελέγχεται η εγκυρότητα του requester μέσω token. Αν είναι άκυρο, επιστρέφεται σφάλμα «Unauthorized». Έπειτα αναζητείται το φεστιβάλ με βάση το ID· αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν το φεστιβάλ έχει ήδη ανακοινωθεί, επιστρέφεται σφάλμα «Cannot update: festival announced». Στη συνέχεια ελέγχεται αν ο requester είναι ORGANIZER του φεστιβάλ· διαφορετικά επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, ενημερώνονται τα βασικά πεδία (όνομα, περιγραφή, ημερομηνίες), τα nested objects (venue layout, budget, vendors), καθώς και οι organizers και staff. Τέλος αποθηκεύεται το φεστιβάλ και επιστρέφεται μήνυμα επιτυχίας «Festival updated successfully» με τα νέα στοιχεία.



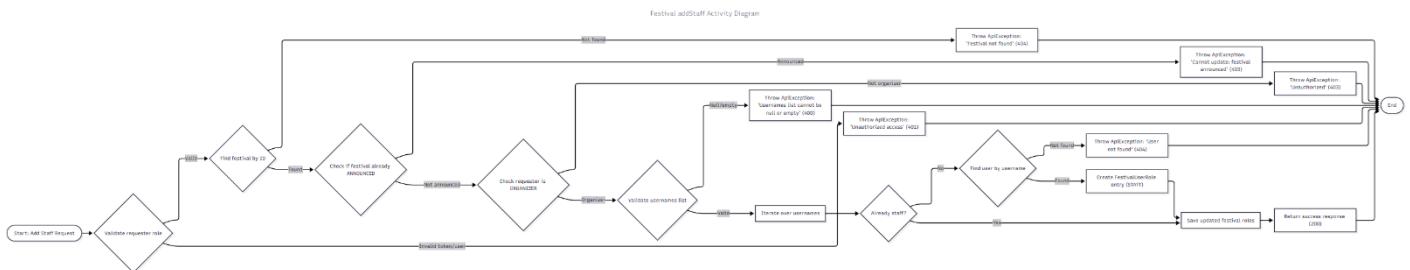
Activity Diagram – add organizers festival:



Το διάγραμμα παρουσιάζει τη διαδικασία προσθήκης νέων organizers σε ένα φεστιβάλ. Αρχικά ελέγχεται η εγκυρότητα του requester μέσω token· αν είναι άκυρο, επιστρέφεται σφάλμα «Unauthorized access». Έπειτα αναζητείται το φεστιβάλ με βάση το ID· αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν το φεστιβάλ έχει ήδη ανακοινωθεί, επιστρέφεται σφάλμα «Cannot update: festival announced». Στη συνέχεια ελέγχεται αν ο requester είναι ORGANIZER· διαφορετικά επιστρέφεται σφάλμα «Unauthorized». Ακολουθεί έλεγχος της λίστας με usernames· αν είναι κενή ή null, επιστρέφεται σφάλμα «Usernames list cannot be null or empty». Επειδή όλες οι λίστες περιέχουν τον χρήστη που αποθηκεύεται στην αρχή, αν ο χρήστης είναι ήδη organizer, παραλείπεται. Αν δεν είναι, γίνεται αναζήτηση στη βάση χρηστών· αν δεν βρεθεί, επιστρέφεται σφάλμα «User not found». Αν βρεθεί, δημιουργείται και αποθηκεύεται νέος ρόλος ORGANIZER. Όταν ολοκληρωθεί η επεξεργασία όλων των usernames, αποθηκεύεται το φεστιβάλ με τις ενημερωμένες σχέσεις ρόλων και επιστρέφεται μήνυμα επιτυχίας «Organizers added successfully» με την ενημερωμένη λίστα των organizers.

Για κάθε username στη λίστα, αν ο χρήστης είναι ήδη organizer, παραλείπεται. Αν δεν είναι, γίνεται αναζήτηση στη βάση χρηστών· αν δεν βρεθεί, επιστρέφεται σφάλμα «User not found». Αν βρεθεί, δημιουργείται και αποθηκεύεται νέος ρόλος ORGANIZER. Όταν ολοκληρωθεί η επεξεργασία όλων των usernames, αποθηκεύεται το φεστιβάλ με τις ενημερωμένες σχέσεις ρόλων και επιστρέφεται μήνυμα επιτυχίας «Organizers added successfully» με την ενημερωμένη λίστα των organizers.

Activity Diagram – add staff festival:

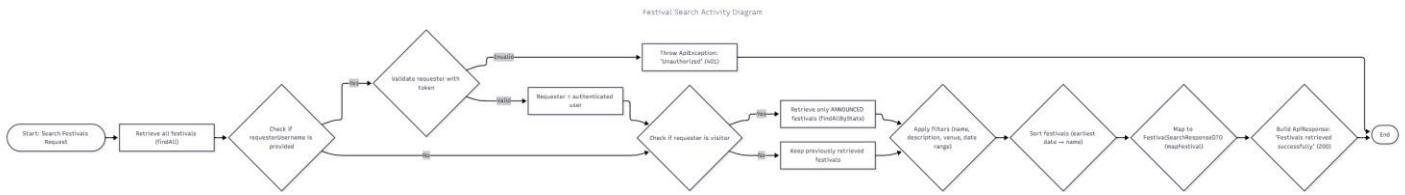


Το διάγραμμα παρουσιάζει τη διαδικασία προσθήκης νέων staff σε ένα φεστιβάλ. Αρχικά ελέγχεται η εγκυρότητα του requester μέσω token· αν είναι άκυρο, επιστρέφεται σφάλμα «Unauthorized access». Έπειτα αναζητείται το φεστιβάλ με βάση το ID· αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν το φεστιβάλ έχει ήδη ανακοινωθεί, επιστρέφεται σφάλμα «Cannot update: festival announced». Στη συνέχεια ελέγχεται αν ο requester είναι ORGANIZER· διαφορετικά επιστρέφεται σφάλμα «Unauthorized». Επειδή όλες οι λίστες περιέχουν τον χρήστη που αποθηκεύεται στην αρχή, αν ο χρήστης είναι ήδη staff, παραλείπεται. Αν δεν είναι, γίνεται αναζήτηση στη βάση χρηστών· αν δεν βρεθεί, επιστρέφεται σφάλμα «User not found». Αν βρεθεί, δημιουργείται και αποθηκεύεται νέος ρόλος STAFF. Όταν ολοκληρωθεί η επεξεργασία όλων των usernames, αποθηκεύεται το φεστιβάλ με τις ενημερωμένες σχέσεις ρόλων και επιστρέφεται μήνυμα επιτυχίας «Staff members added successfully» με την ενημερωμένη λίστα του προσωπικού.

Για κάθε username στη λίστα, αν ο χρήστης είναι ήδη staff, παραλείπεται. Αν δεν είναι, γίνεται αναζήτηση στη βάση χρηστών· αν δεν βρεθεί, επιστρέφεται σφάλμα «User not found». Αν βρεθεί, δημιουργείται και αποθηκεύεται νέος ρόλος STAFF. Όταν ολοκληρωθεί η επεξεργασία όλων των usernames, αποθηκεύεται το φεστιβάλ με τις ενημερωμένες σχέσεις ρόλων και επιστρέφεται μήνυμα επιτυχίας «Staff members added successfully» με την ενημερωμένη λίστα του προσωπικού.

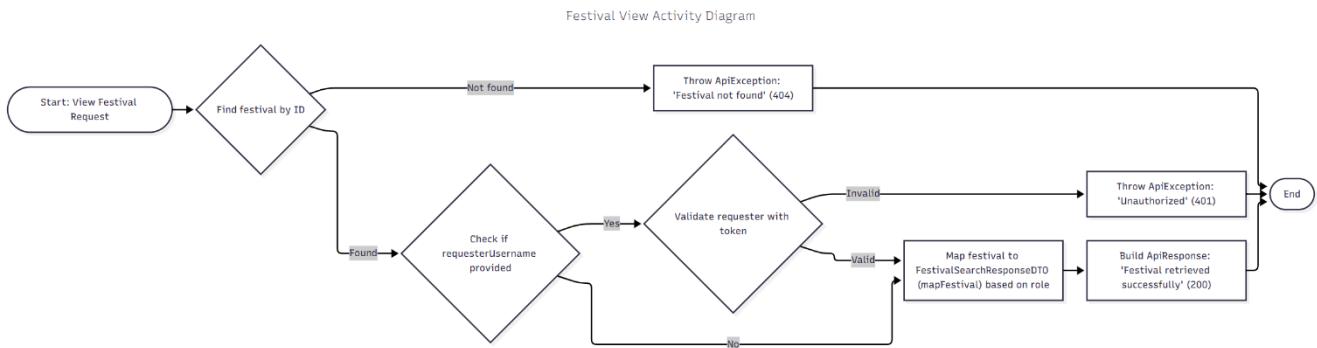


Activity Diagram – Search festival:



Το διάγραμμα παρουσιάζει τη διαδικασία αναζήτησης φεστιβάλ. Αν παρέχεται username και token, ελέγχεται η εγκυρότητα του requester· αν είναι άκυρο, επιστρέφεται σφάλμα «Unauthenticated». Αν όχι, ο χρήστης θεωρείται επισκέπτης. Στη συνέχεια ανακτώνται όλα τα φεστιβάλ από τη βάση. Για επισκέπτες, εμφανίζονται μόνο φεστιβάλ με κατάσταση ANNOUNCED. Στη συνέχεια εφαρμόζονται τα φίλτρα αναζήτησης βάσει ονόματος, περιγραφής, χώρου και ημερομηνιών. Τα αποτελέσματα ταξινομούνται πρώτα με βάση την πιο πρώιμη ημερομηνία και μετά κατά όνομα. Τέλος, τα φεστιβάλ μετατρέπονται σε DTOs και επιστρέφεται μήνυμα επιτυχίας «Festivals retrieved successfully» με τη λίστα των ταιριαστών φεστιβάλ.

Activity Diagram – View festival:

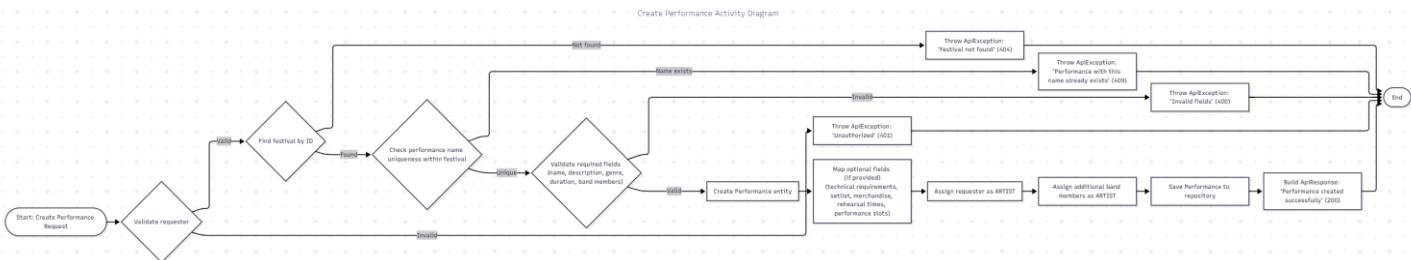


Το διάγραμμα παρουσιάζει τη διαδικασία εμφάνισης ενός συγκεκριμένου φεστιβάλ. Αρχικά, το σύστημα αναζητά το φεστιβάλ στο repository χρησιμοποιώντας το festivalId από το αίτημα. Αν δεν βρεθεί φεστιβάλ, επιστρέφεται σφάλμα «Festival not found» . Αν στο αίτημα παρέχεται requesterUsername, τότε γίνεται έλεγχος εγκυρότητας του χρήστη μέσω validateRequester με το αντίστοιχο token. Σε περίπτωση που ο έλεγχος αποτύχει, επιστρέφεται σφάλμα «Unauthenticated». Αν δεν παρέχεται username, ο χρήστης θεωρείται επισκέπτης (visitor).

Στη συνέχεια, το φεστιβάλ μετατρέπεται σε FestivalSearchResponseDTO μέσω της μεθόδου mapFestival, η οποία προσαρμόζει τις πληροφορίες ανάλογα με το ρόλο (visitor ή authenticated χρήστης). Τέλος, επιστρέφεται επιτυχής απόκριση «Festival retrieved successfully» με το DTO του φεστιβάλ.

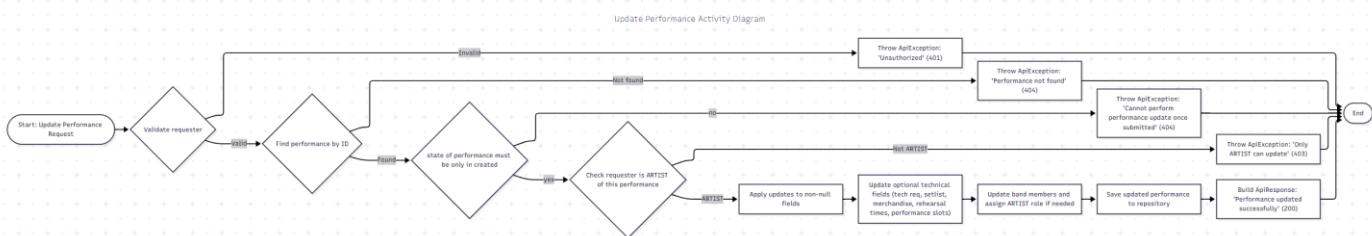


Activity Diagram – Create performance:



Το διάγραμμα παρουσιάζει τη διαδικασία δημιουργίας μιας παράστασης (performance) για ένα φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη με validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια ανακτάται το φεστιβάλ βάσει festivalId; αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Μετά, ελέγχεται η μοναδικότητα του ονόματος παράστασης μέσα στο φεστιβάλ και η εγκυρότητα των υποχρεωτικών πεδίων (όνομα, περιγραφή, είδος, διάρκεια, band members). Αν υπάρχει πρόβλημα, επιστρέφεται το αντίστοιχο σφάλμα. Αν όλα είναι έγκυρα, δημιουργείται η οντότητα Performance, αντιστοιχίζονται προαιρετικά πεδία (setlist, merchandise, technical requirements, προτιμώμενες ώρες πρόβας/slots) και αποθηκεύεται. Ο δημιουργός ανατίθεται ως ARTIST, όπως και τα υπόλοιπα μέλη της μπάντας αν δεν είναι ήδη καλλιτέχνες για το φεστιβάλ. Τέλος, η παράσταση αποθηκεύεται και επιστρέφεται επιτυχής απόκριση με id, name, state και mainArtist.

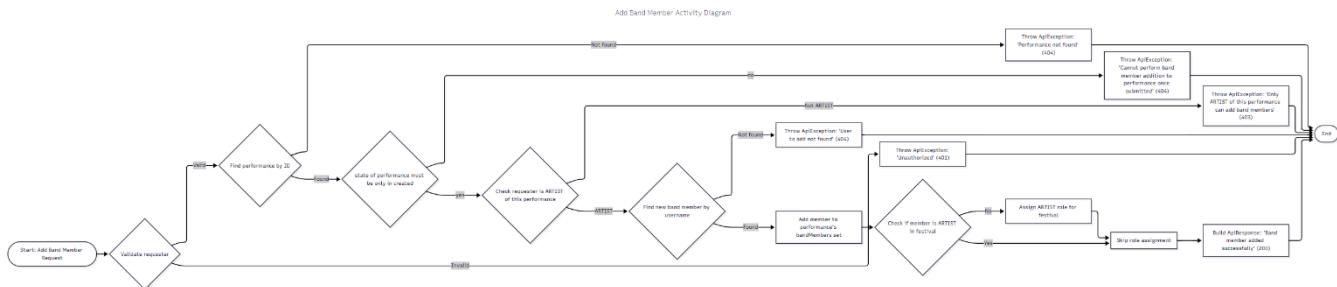
Activity Diagram – Update performance:



Το διάγραμμα παρουσιάζει τη διαδικασία ενημέρωσης μιας παράστασης (performance). Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, ανακτάται η παράσταση βάσει performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Μόνο performance σε κατάσταση created μπορούν να δεχθούν αλλαγές. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Cannot perform performance update once submitted». Επειτα ελέγχεται ότι ο χρήστης είναι ο δημιουργός (ARTIST) της παράστασης· αν όχι, επιστρέφεται σφάλμα «Only ARTIST can update». Αν όλα είναι έγκυρα, εφαρμόζονται οι ενημερώσεις στα πεδία που παρέχονται (όνομα, περιγραφή, είδος, διάρκεια, technical requirements, setlist, merchandise, προτιμώμενες ώρες πρόβας και slots). Ο έλεγχος μοναδικότητας του ονόματος γίνεται μέσα στο φεστιβάλ, εκτός της τρέχουσας παράστασης. Τέλος, ενημερώνονται τα μέλη της μπάντας (και οι αντίστοιχοι ARTIST ρόλοι στο φεστιβάλ αν χρειάζεται), η παράσταση αποθηκεύεται και επιστρέφεται επιτυχής απόκριση με id, name, genre και state.

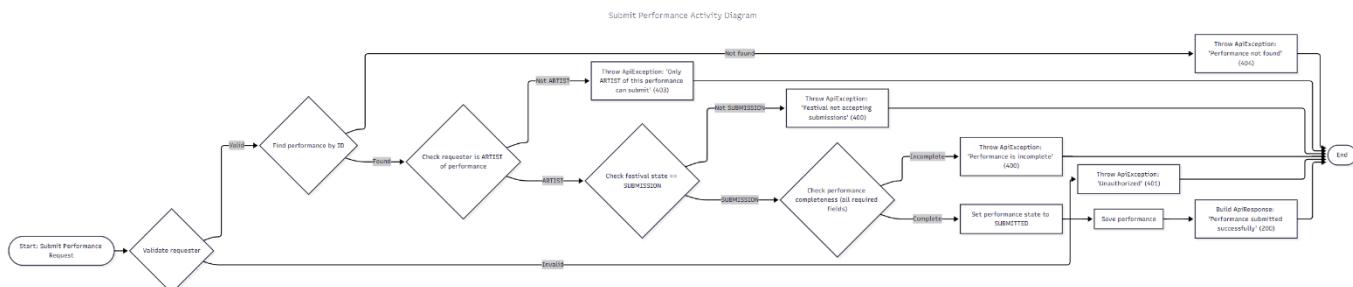


Activity Diagram – Add band member to festival:



Το διάγραμμα παρουσιάζει τη διαδικασία προσθήκης νέου μέλους μπάντας σε μια παράσταση. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthenticated». Στη συνέχεια, ανακτάται η παράσταση βάσει performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Μόνο performance σε κατάσταση created μπορούν να δεχθούν αλλαγές. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Cannot perform band member addition to performance once submitted». Έπειτα ελέγχεται ότι ο χρήστης είναι ο δημιουργός (ARTIST) της παράστασης: αν όχι, επιστρέφεται σφάλμα «Only ARTIST can add band members ». Αν όλα είναι έγκυρα, ανακτάται ο χρήστης που θα προστεθεί ως νέο μέλος μπάντας. Αν δεν βρεθεί, επιστρέφεται σφάλμα «User to add not found». Το νέο μέλος προστίθεται στην παράσταση και, αν δεν έχει ήδη ρόλο ARTIST στο φεστιβάλ, του ανατίθεται. Τέλος επιστρέφεται επιτυχής απόκριση με performanceId και bandMemberAdded.

Activity Diagram – Submit performance:

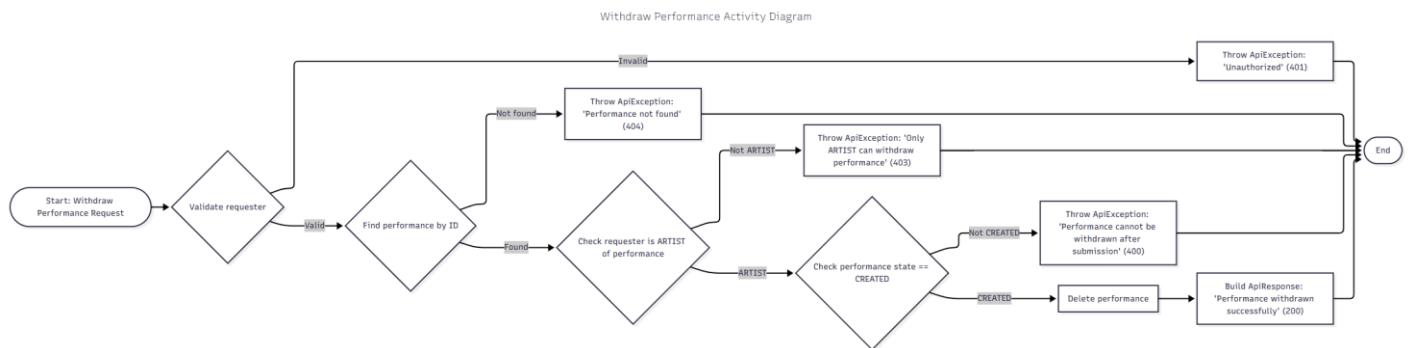


Το διάγραμμα παρουσιάζει τη διαδικασία υποβολής μιας παράστασης (performance) για ένα φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthenticated». Στη συνέχεια, το σύστημα αναζητά την παράσταση βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Αν βρεθεί, ελέγχεται ότι ο requester είναι ο ARTIST της συγκεκριμένης παράστασης. Αν δεν είναι, επιστρέφεται σφάλμα « Only ARTIST of this performance can submit it». Έπειτα, ελέγχεται ότι το φεστιβάλ βρίσκεται σε κατάσταση SUBMISSION.



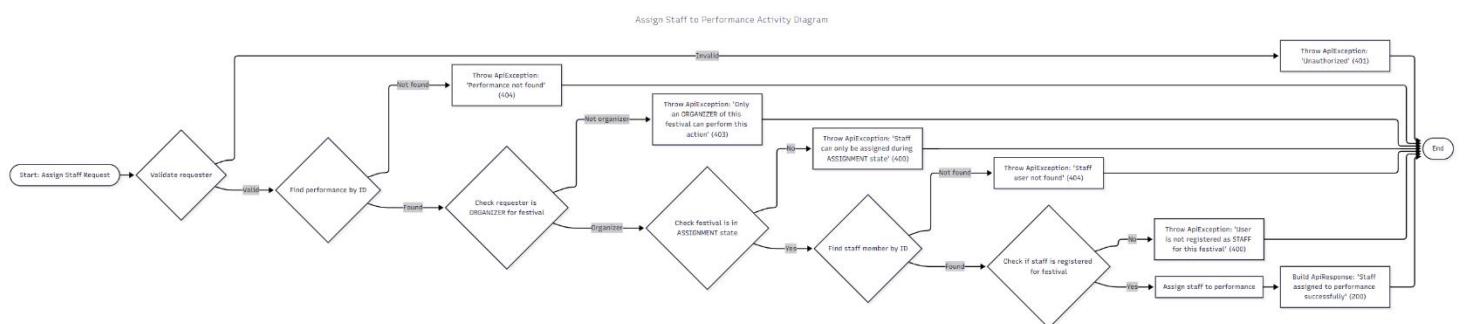
Αν όχι, επιστρέφεται σφάλμα «Festival is not accepting submissions currently». Ακολουθεί έλεγχος της πληρότητας της παράστασης: όλα τα υποχρεωτικά πεδία (όνομα, περιγραφή, είδος, διάρκεια, τεχνικές απαιτήσεις, setlist, merchandise, προτιμήσεις πρόβας και slots) πρέπει να είναι παρόντα. Αν κάποιο πεδίο λείπει, επιστρέφεται σφάλμα «Performance is incomplete and cannot be submitted». Αν όλοι οι έλεγχοι περάσουν, η κατάσταση της παράστασης ενημερώνεται σε SUBMITTED, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Performance submitted successfully» με το id, name και state της παράστασης.

Activity Diagram – Withdraw performance:



Το διάγραμμα παρουσιάζει τη διαδικασία ανάκλησης (withdraw) του performance πριν την υποβολή του. Αρχικά γίνεται έλεγχος εγκυρότητας του αιτούντος χρήστη, με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά την performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Αν βρεθεί, ελέγχεται ότι ο requester είναι ARTIST της performance. Αν δεν είναι, επιστρέφεται σφάλμα «Only ARTIST of this performance can update it». Μετά, ελέγχεται η κατάσταση της performance: μόνο performances σε κατάσταση CREATED μπορούν να ανακληθούν. Αν η performance έχει ήδη υποβληθεί ή βρίσκεται σε διαφορετική κατάσταση, επιστρέφεται σφάλμα «Performance cannot be withdrawn after submission». Αν όλες οι συνθήκες είναι έγκυρες, η performance διαγράφεται από το repository και επιστρέφεται επιτυχής απόκριση «Performance withdrawn successfully» με το id και το name της performance.

Activity Diagram – add staff to performance:



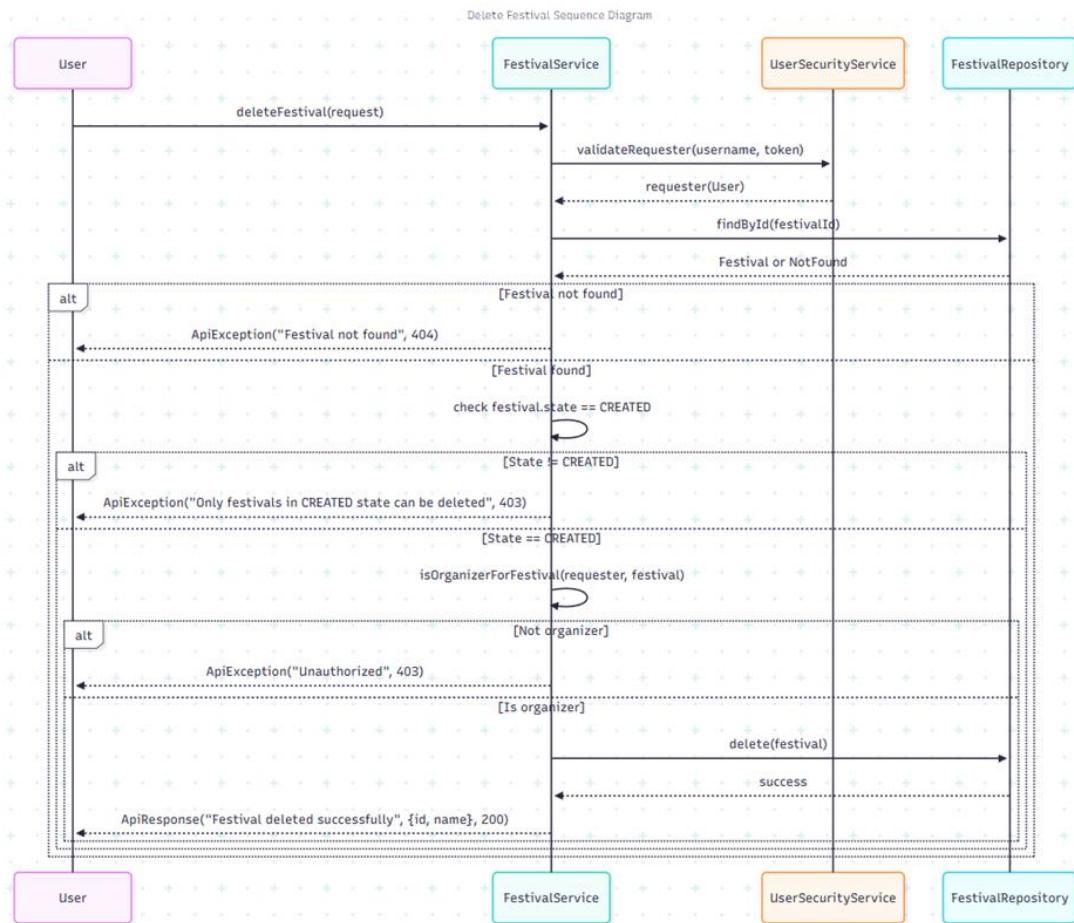


Το διάγραμμα παρουσιάζει τη διαδικασία ανάθεσης ενός μέλους STAFF σε ένα performance. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά την performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Έπειτα, ελέγχεται ότι ο requester είναι ORGANIZER του φεστιβάλ στο οποίο ανήκει η performance. Αν δεν είναι, επιστρέφεται σφάλμα «Only an ORGANIZER of this festival can perform this action». Μετά, ελέγχεται ότι το φεστιβάλ βρίσκεται σε κατάσταση ASSIGNMENT. Αν όχι, επιστρέφεται σφάλμα «Staff can only be assigned during ASSIGNMENT state».

Στη συνέχεια, το σύστημα αναζητά το STAFF μέλος βάσει του staffUserId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Staff user not found». Ακολουθεί έλεγχος ότι το STAFF μέλος είναι καταχωρημένο για το συγκεκριμένο φεστιβάλ. Αν όχι, επιστρέφεται σφάλμα «User is not registered as STAFF for this festival». Αν όλα τα παραπάνω είναι έγκυρα, το STAFF μέλος ανατίθεται ως υπεύθυνος για το συγκεκριμένο performance και η αλλαγή αποθηκεύεται στο repository. Τέλος, επιστρέφεται επιτυχής απόκριση «Staff assigned to performance successfully» με το performanceId, performanceName και assignedStaff.

4.3.2 Sequence Diagrams

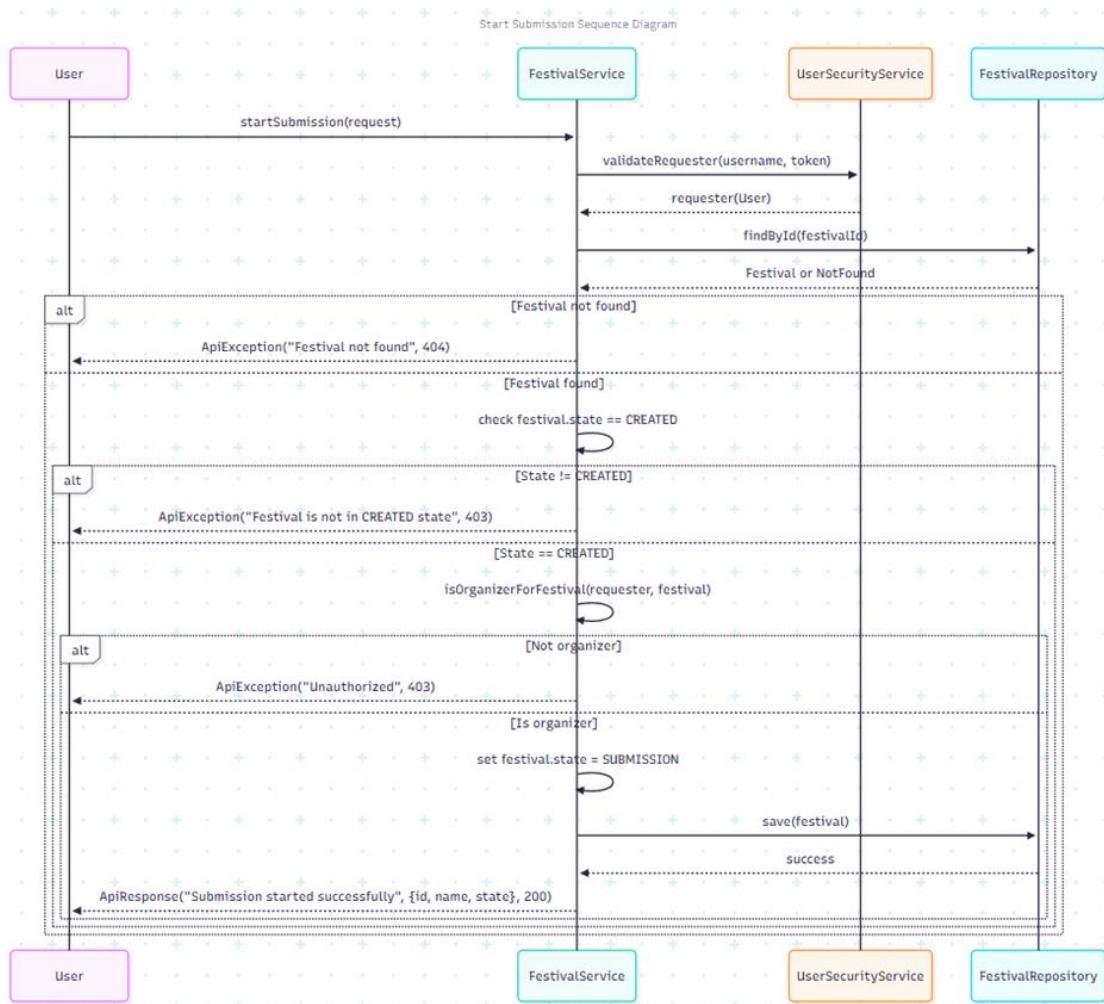
sequence diagram – Delete festival





Το παραπάνω διάγραμμα ακολουθίας παρουσιάζει τη διαδικασία διαγραφής ενός φεστιβάλ. Αρχικά, γίνεται έλεγχος εγκυρότητας του χρήστη μέσω validateRequester με το αντίστοιχο token. Σε περίπτωση που ο έλεγχος αποτύχει, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει festivalId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ. Μόνο φεστιβάλ σε κατάσταση CREATED μπορούν να διαγραφούν· σε διαφορετική περίπτωση επιστρέφεται σφάλμα. Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, το φεστιβάλ διαγράφεται από το repository και επιστρέφεται επιτυχής απόκριση «Festival deleted successfully» με το id και το name του φεστιβάλ.

Sequence diagram – Start submission



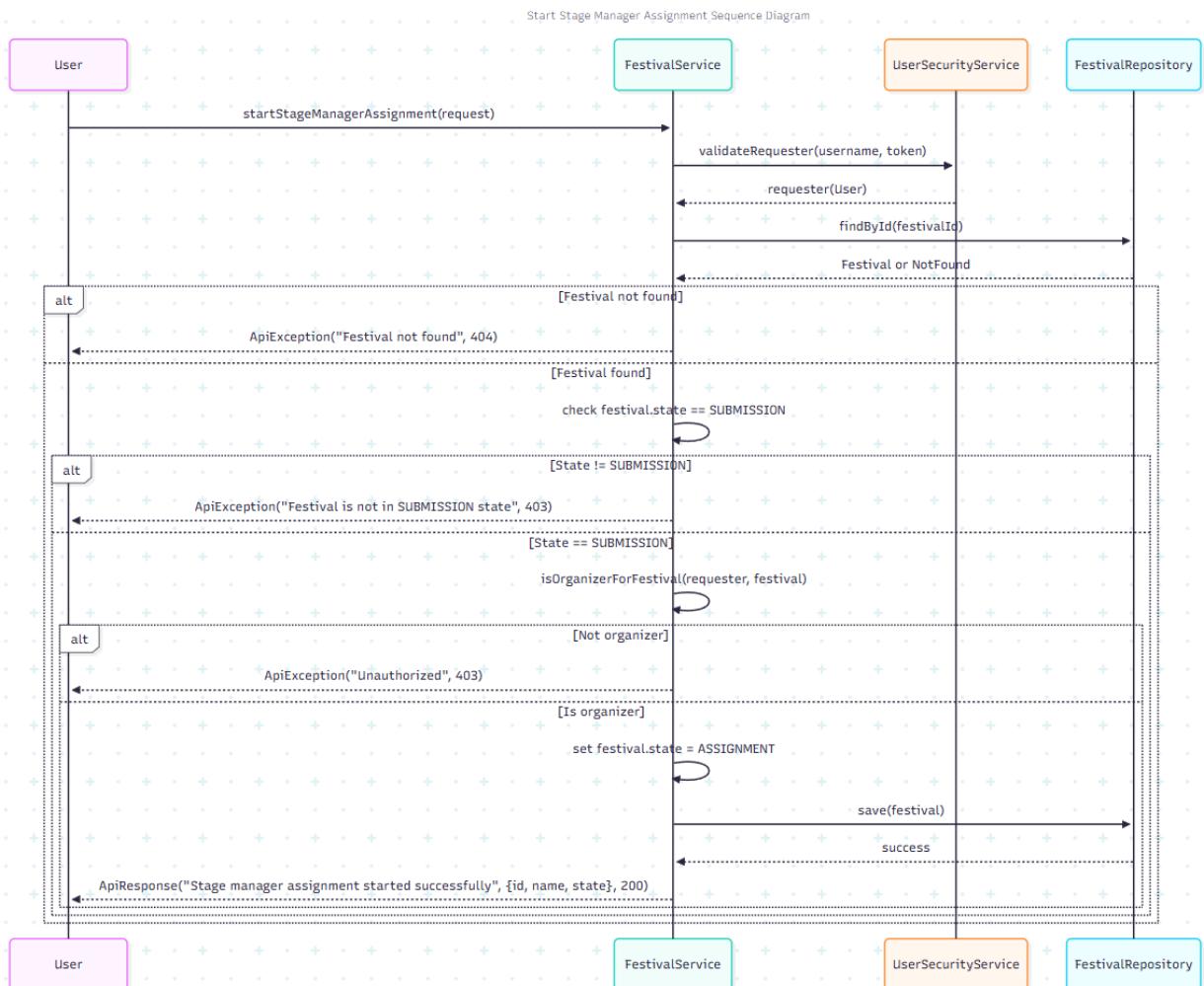
Το διάγραμμα αυτό, παρουσιάζει τη διαδικασία έναρξης της φάσης υποβολής ενός φεστιβάλ.

Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου `validateRequester`. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του `festivalId`. Αν δεν



βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ μόνο φεστιβάλ σε κατάσταση CREATED μπορούν να ξεκινήσουν τη φάση υποβολής. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in CREATED state». Επειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε SUBMISSION, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Submission started successfully» με το id, το name και τη νέα κατάσταση του φεστιβάλ.

Sequence diagram – startStageManagerAssignment

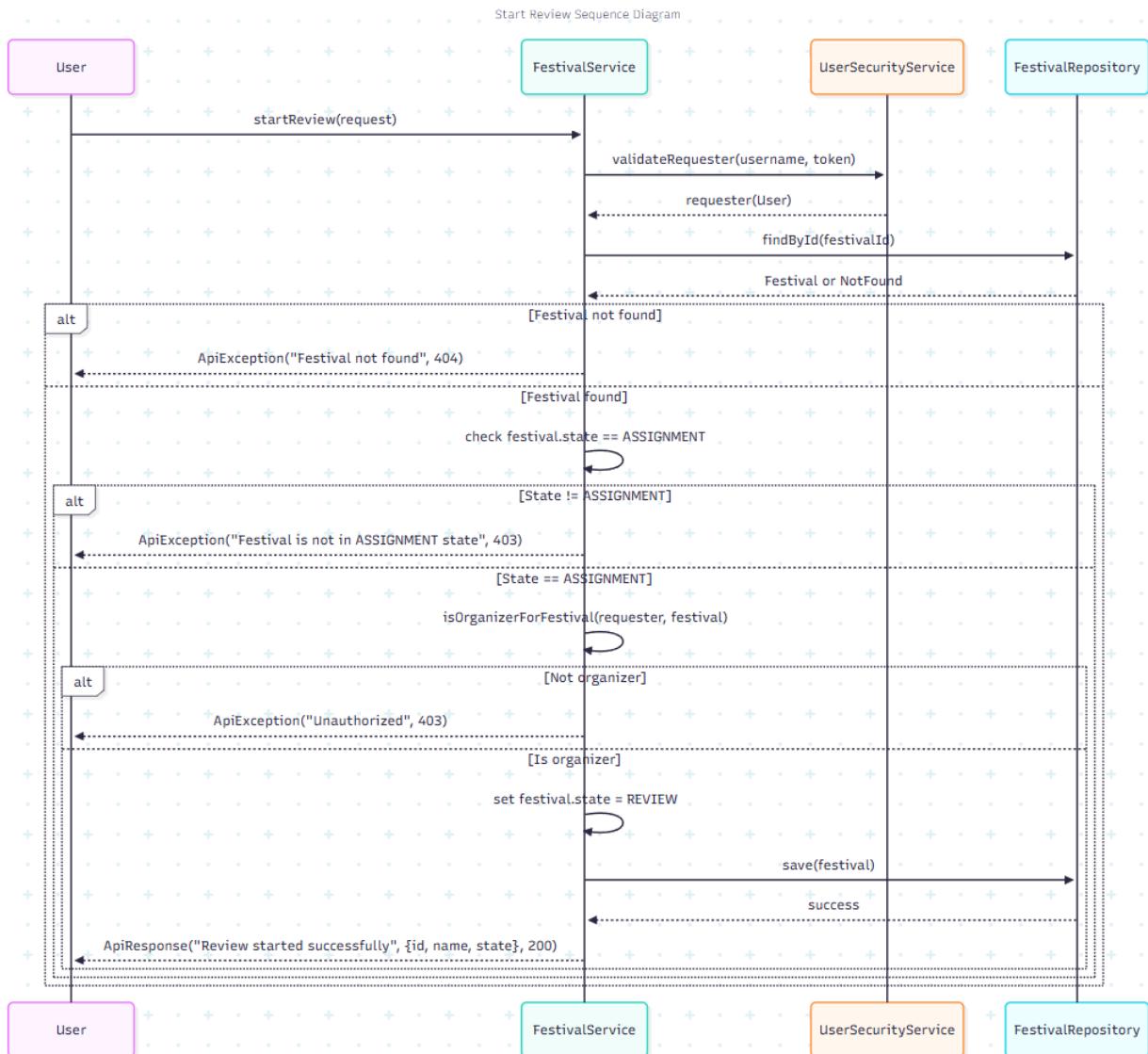


Το διάγραμμα παρουσιάζει τη διαδικασία έναρξης της φάσης ανάθεσης stage managers ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του festivalId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ μόνο φεστιβάλ σε κατάσταση SUBMISSION μπορούν να ξεκινήσουν τη φάση ανάθεσης stage managers. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in SUBMISSION state».



Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthenticated». Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε ASSIGNMENT, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Stage manager assignment started successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – startReview

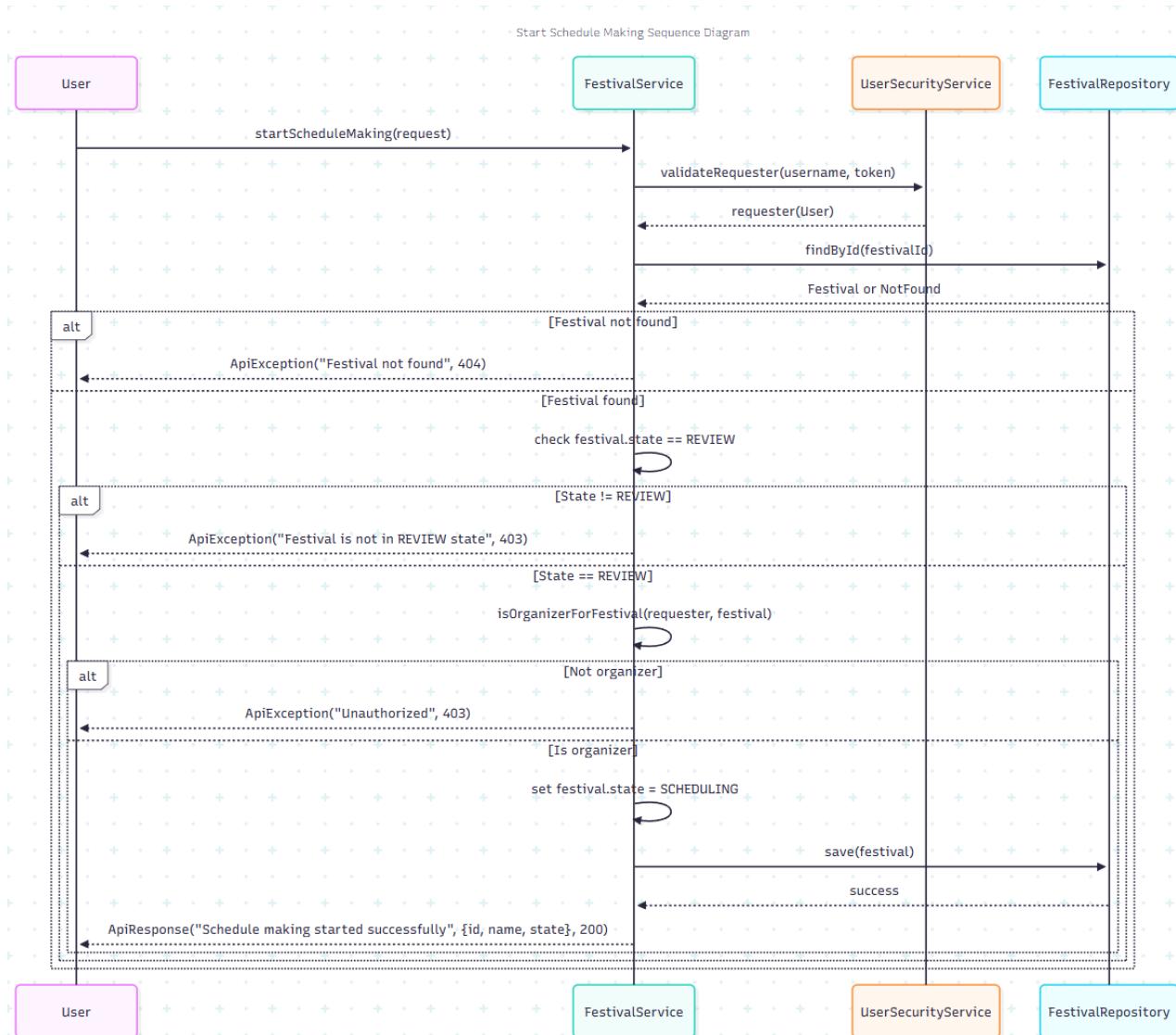


Το διάγραμμα παρουσιάζει τη διαδικασία έναρξης της φάσης αξιολόγησης (review) ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου `validateRequester`. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthenticated». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του `festivalId`. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ· μόνο φεστιβάλ σε κατάσταση `ASSIGNMENT` μπορούν να ξεκινήσουν τη φάση αξιολόγησης. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in `ASSIGNMENT` state».



Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε REVIEW, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Review started successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – schedule making

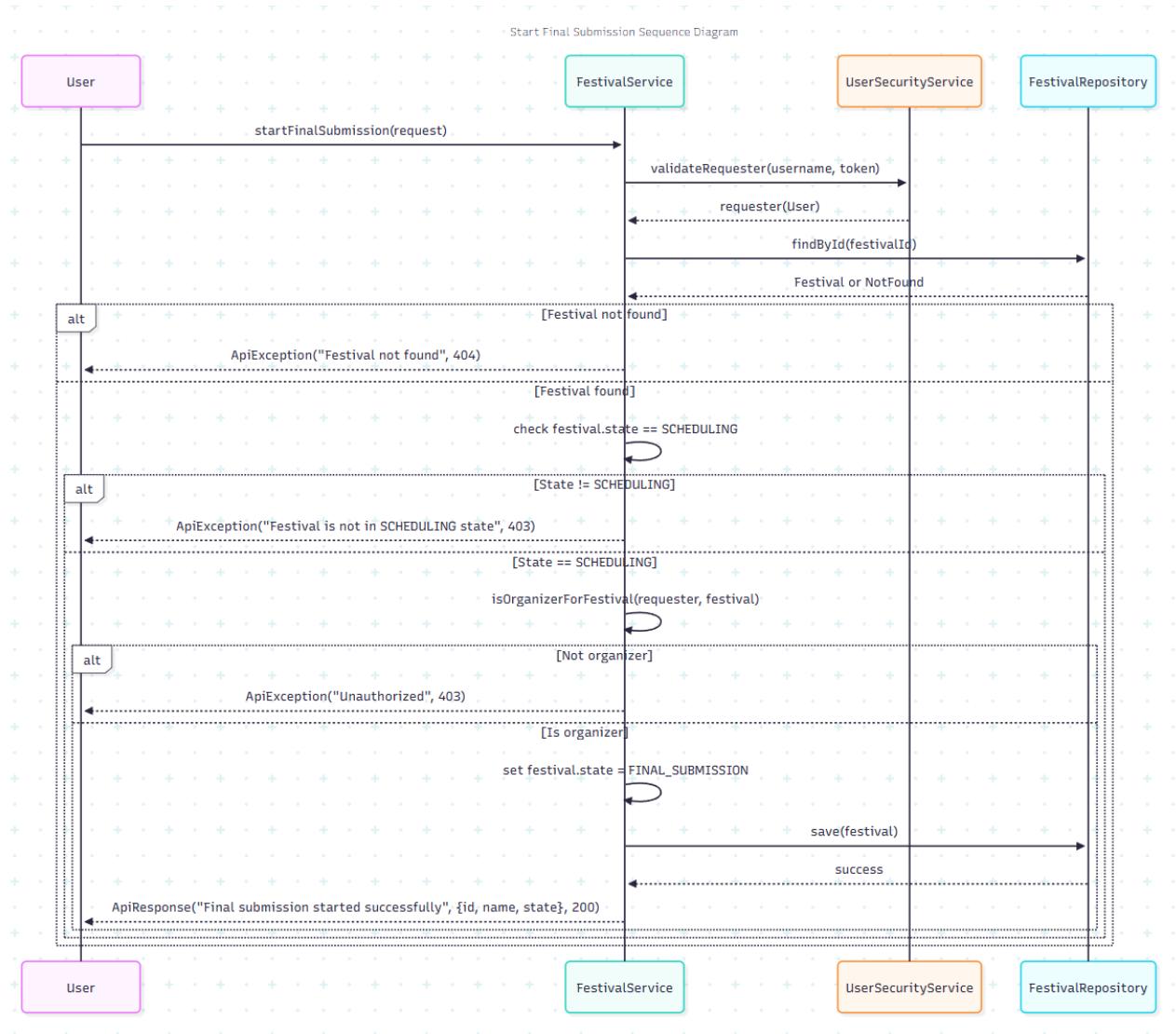


Το διάγραμμα παρουσιάζει τη διαδικασία έναρξης της φάσης δημιουργίας προγράμματος (schedule making) ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του festivalId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ· μόνο φεστιβάλ σε κατάσταση REVIEW μπορούν να ξεκινήσουν τη φάση δημιουργίας προγράμματος. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in REVIEW state».



Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε SCHEDULING, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Schedule making started successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – final submittion

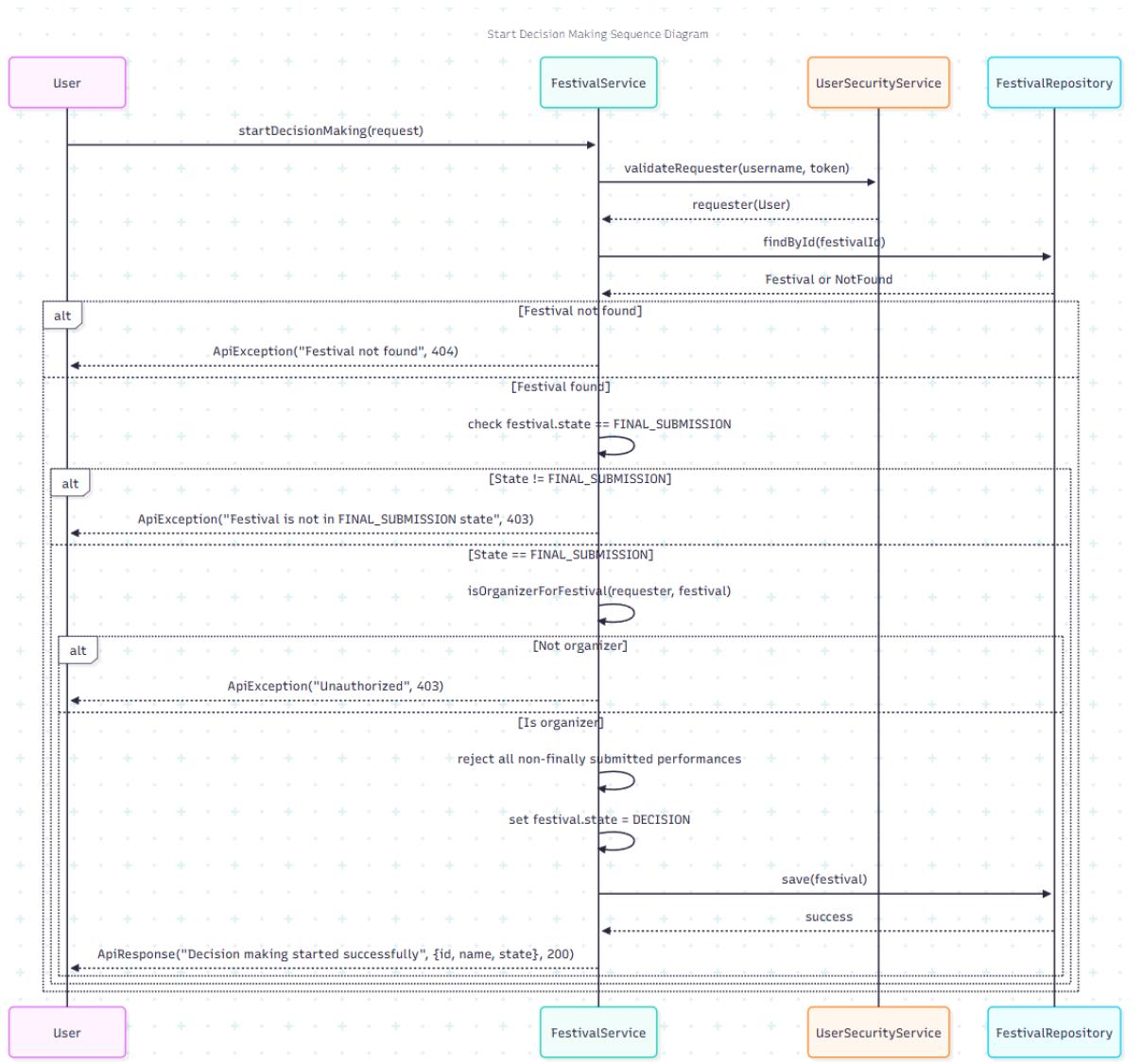


Το διάγραμμα παρουσιάζει τη διαδικασία έναρξης της φάσης τελικών υποβολών (final submission) ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου `validateRequester`. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του `festivalId`. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ· μόνο φεστιβάλ σε κατάσταση `SCHEDULING` μπορούν να ξεκινήσουν τη φάση τελικών υποβολών. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in `SCHEDULING` state».



Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthenticated». Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε FINAL_SUBMISSION, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Final submission started successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – final decision making

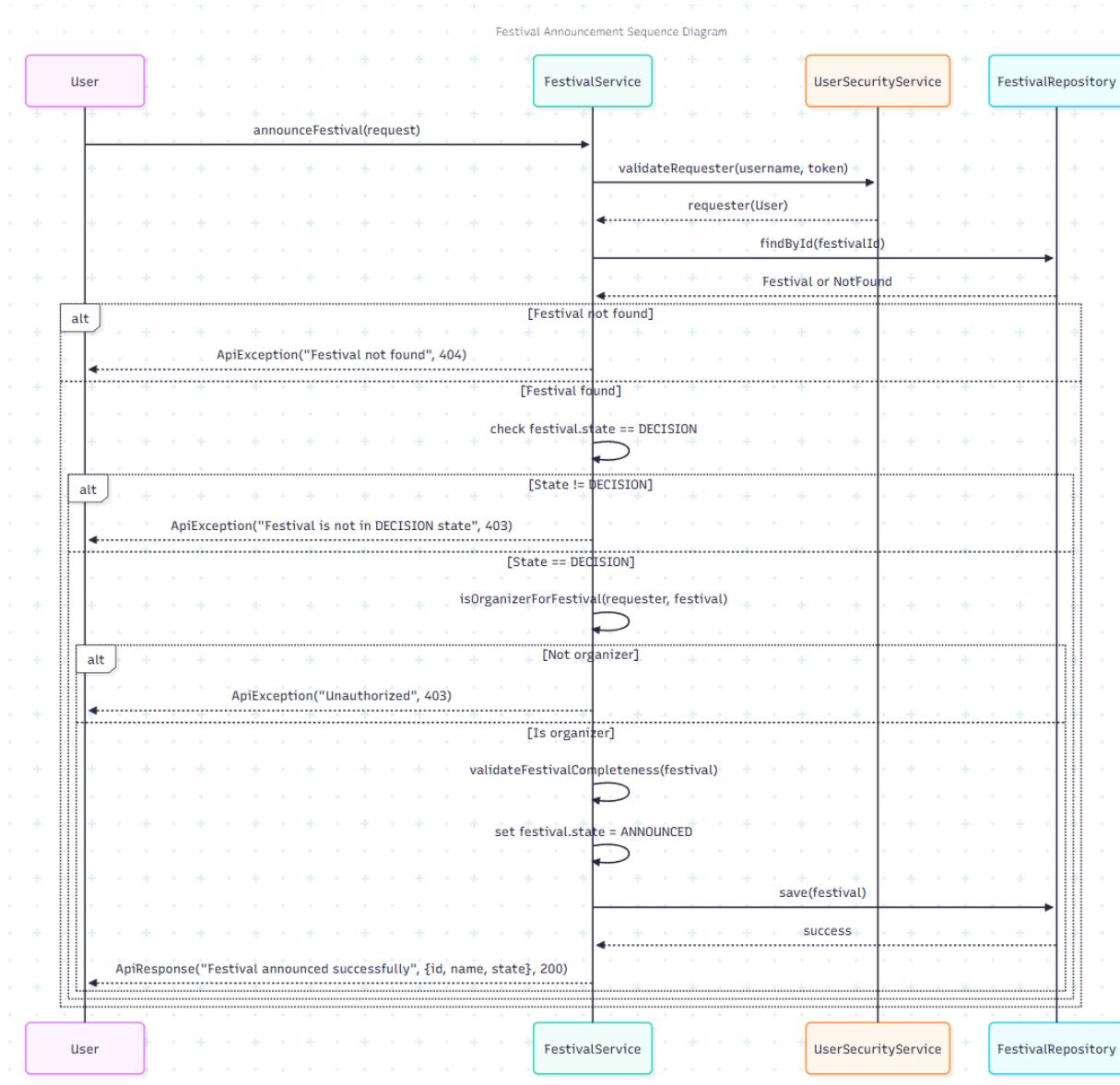


Το διάγραμμα παρουσιάζει τη διαδικασία έναρξης της φάσης λήψης αποφάσεων (decision-making) ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthenticated». Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του festivalId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ· μόνο φεστιβάλ σε κατάσταση FINAL_SUBMISSION μπορούν να ξεκινήσουν τη φάση λήψης αποφάσεων. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in FINAL_SUBMISSION state».



Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Αν όλα είναι έγκυρα, όλα τα performances που δεν έχουν υποβληθεί τελικά επισημαίνονται ως REJECTED με σχόλια «AUTOMATICALLY REJECTED – NOT FINALLY SUBMITTED». Στη συνέχεια, η κατάσταση του φεστιβάλ ενημερώνεται σε DECISION, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Decision making started successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – festival announce



Το διάγραμμα παρουσιάζει τη διαδικασία δημόσιας ανακοίνωσης ενός φεστιβάλ. Αρχικά γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized».

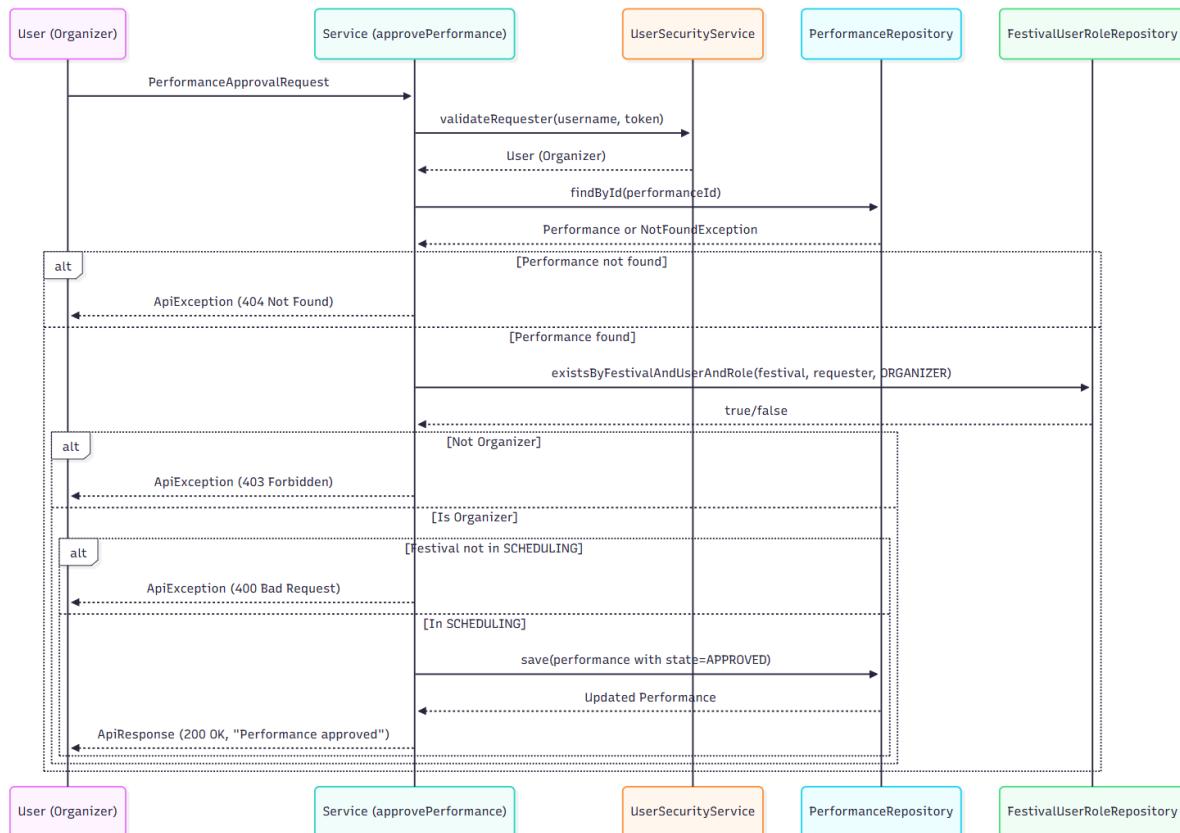


Στη συνέχεια, το σύστημα αναζητά το φεστιβάλ βάσει του festivalId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Festival not found». Αν βρεθεί, ελέγχεται η κατάσταση του φεστιβάλ· μόνο φεστιβάλ σε κατάσταση DECISION μπορούν να ανακοινωθούν. Σε διαφορετική περίπτωση επιστρέφεται σφάλμα «Festival is not in DECISION state».

Έπειτα, ελέγχεται ότι ο requester είναι οργανωτής του φεστιβάλ. Αν δεν είναι, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια γίνεται έλεγχος ότι όλα τα απαραίτητα στοιχεία του φεστιβάλ είναι πλήρη μέσω validateFestivalCompleteness.

Αν όλα είναι έγκυρα, η κατάσταση του φεστιβάλ ενημερώνεται σε ANNOUNCED, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Festival announced successfully» με το id, το name και τη νέα state του φεστιβάλ.

Sequence diagram – performance approval request



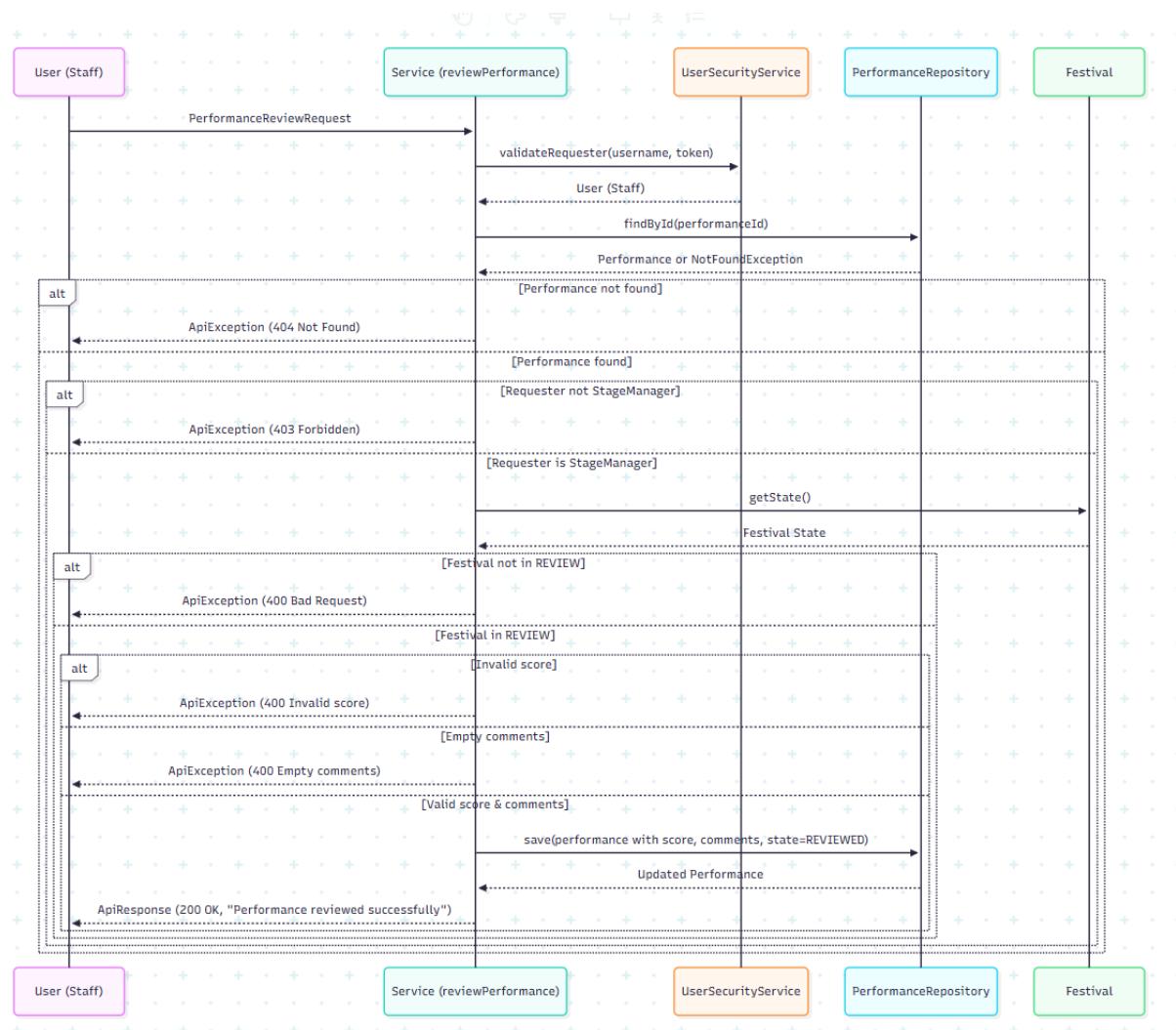
Το διάγραμμα παρουσιάζει τη διαδικασία έγκρισης του performance από έναν οργανωτή. Αρχικά γίνεται έλεγχος εγκυρότητας του αιτούντος χρήστη με χρήση της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found».

Έπειτα, ελέγχεται ότι ο requester είναι ORGANIZER του φεστιβάλ στο οποίο ανήκει το performance. Αν δεν είναι, επιστρέφεται σφάλμα «Only an organizer of the festival can approve this performance». Μετά, ελέγχεται ότι το φεστιβάλ βρίσκεται σε κατάσταση SCHEDULING. Αν όχι, επιστρέφεται σφάλμα «Performances can only be approved when the festival is in



SCEDULING state». Αν όλα τα παραπάνω είναι έγκυρα, η κατάσταση του performance ενημερώνεται σε APPROVED, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Performance approved successfully» με το performanceId, performanceName και το νέο state.

Sequence diagram – performance sequence

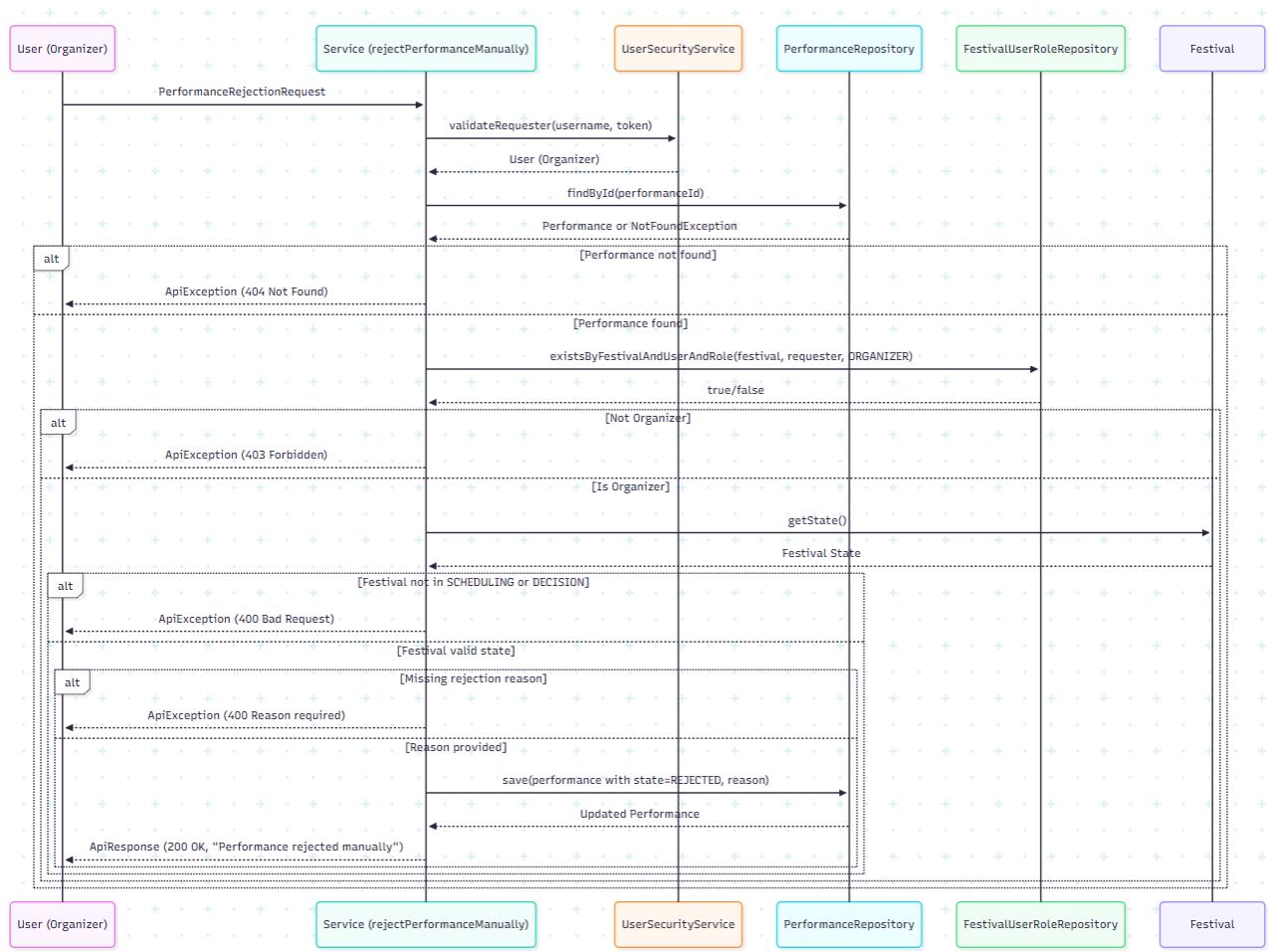


Το διάγραμμα παρουσιάζει τη διαδικασία αξιολόγησης (review) του performance από το μέλος STAFF που έχει οριστεί ως υπεύθυνος. Αρχικά, γίνεται έλεγχος εγκυρότητας του αιτούντος χρήστη με χρήση της μεθόδου validateRequester. Αν τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα «Unauthorized». Στη συνέχεια, το σύστημα αναζητά το performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found». Έπειτα, ελέγχεται ότι ο αιτών είναι το STAFF μέλος που έχει ανατεθεί ως stage manager στη συγκεκριμένη performance. Αν δεν είναι, επιστρέφεται σφάλμα «Only the assigned STAFF member can review this performance». Ακολουθεί έλεγχος ότι το φεστιβάλ βρίσκεται σε κατάσταση REVIEW. Αν όχι, επιστρέφεται σφάλμα «Performance can only be reviewed when the festival is in REVIEW state».



Μετά, ελέγχεται η εγκυρότητα του σκορ (score). Αν είναι εκτός ορίων 0–10, επιστρέφεται σφάλμα «Score must be between 0 and 10». Επίσης, ελέγχονται τα σχόλια του αξιολογητή· αν είναι κενά, επιστρέφεται σφάλμα «Reviewer comments cannot be empty». Αν όλα τα παραπάνω είναι έγκυρα, το performance ενημερώνεται με το score, τα σχόλια, και η κατάστασή της αλλάζει σε REVIEWED. Τέλος, αποθηκεύεται στο repository και επιστρέφεται επιτυχής απόκριση «Performance reviewed successfully» με πληροφορίες για το id, το όνομα, το σκορ, τα σχόλια και την κατάσταση.

Sequence diagram – reject performance

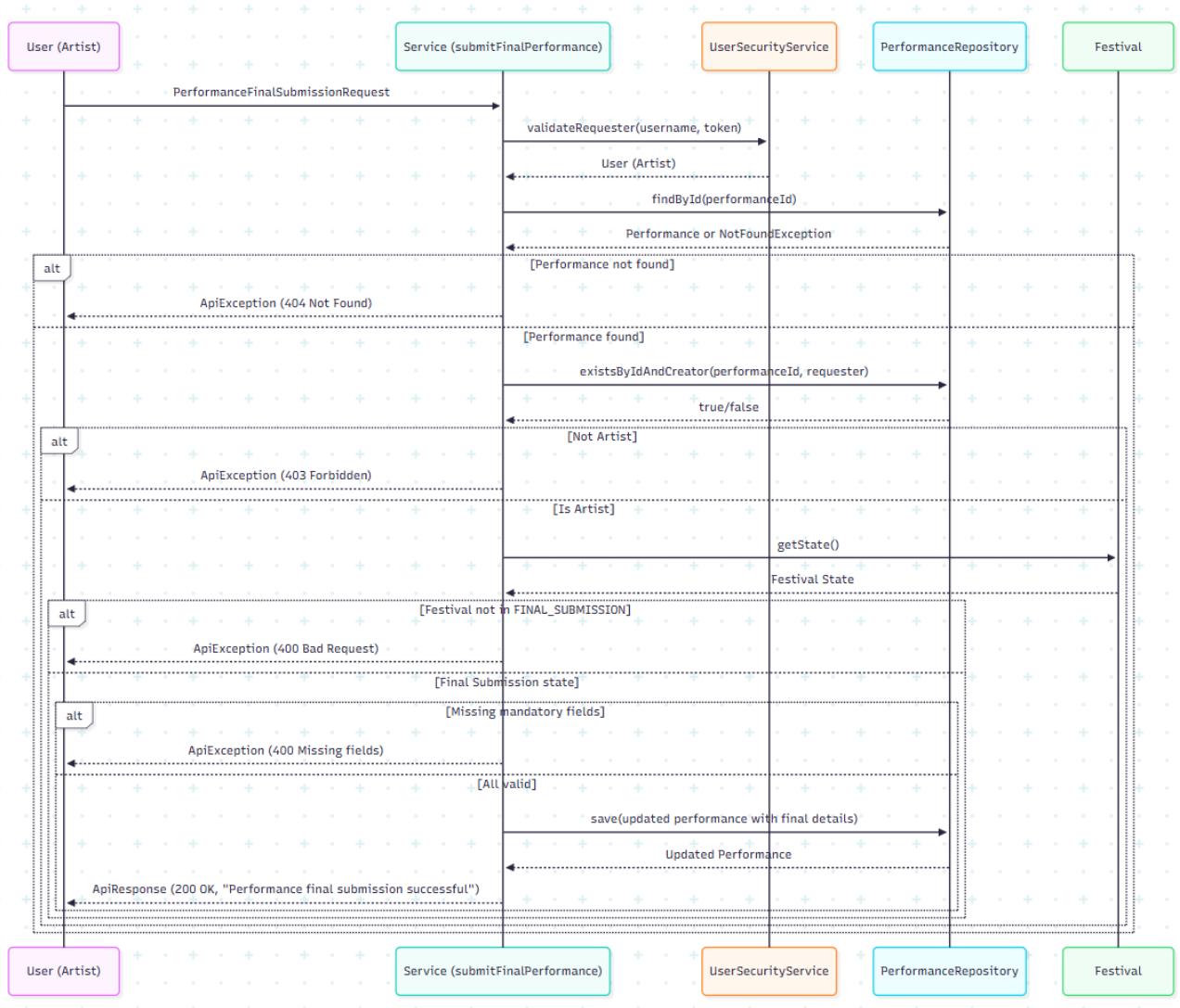


Το διάγραμμα παρουσιάζει τη διαδικασία χειροκίνητης απόρριψης ενός performance. Αρχικά γίνεται έλεγχος εγκυρότητας του αιτούντος μέσω της validateRequester. Αν τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα. Έπειτα αναζητείται το performance βάσει performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα «Performance not found».

Ακολουθεί έλεγχος ότι ο αιτών είναι organizer του φεστιβάλ. Αν όχι, επιστρέφεται σφάλμα. Στη συνέχεια ελέγχεται αν το φεστιβάλ βρίσκεται σε κατάσταση scheduling ή decision. Αν όχι, επιστρέφεται σφάλμα. Έπειτα ελέγχεται ότι υπάρχει αιτιολόγηση απόρριψης. Αν λείπει, επιστρέφεται σφάλμα. Αν όλα είναι έγκυρα, η performance αλλάζει σε κατάσταση rejected, αποθηκεύεται ο λόγος απόρριψης και επιστρέφεται επιτυχής απόκριση με μήνυμα «Performance rejected manually».



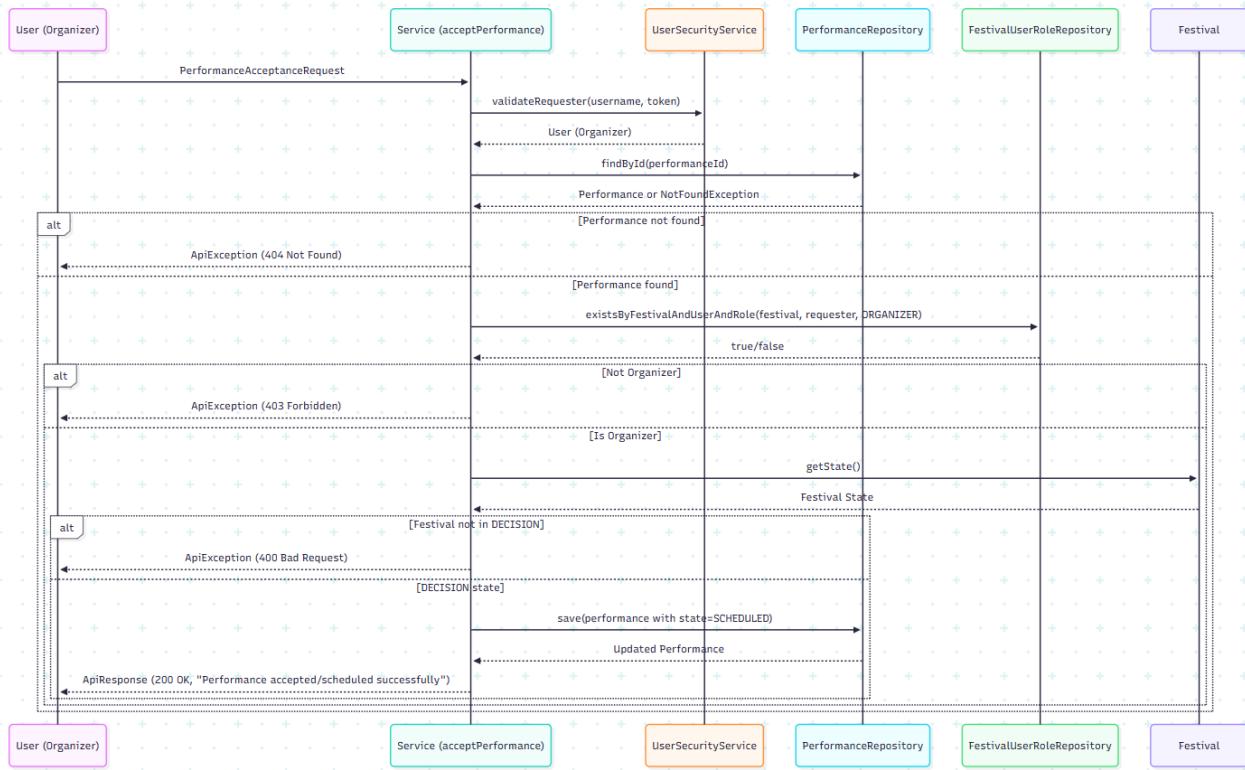
Sequence diagram – submit final performance



Το διάγραμμα δείχνει τη διαδικασία τελικής υποβολής στοιχείων του performance. Ο αιτόν χρήστης ελέγχεται με την validateRequester. Αν τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα. Στη συνέχεια το σύστημα αναζητά το performance με βάση το performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα. Έπειτα επαληθεύεται ότι ο αιτόν είναι ARTIST του συγκεκριμένου performance. Αν όχι, επιστρέφεται σφάλμα. Ακολουθεί έλεγχος ότι το φεστιβάλ βρίσκεται σε κατάσταση FINAL_SUBMISSION. Αν όχι, επιστρέφεται σφάλμα. Στη συνέχεια ελέγχεται ότι έχουν δοθεί υποχρεωτικά στοιχεία: setlist, rehearsal times, performance time slots. Αν λείπει κάποιο, επιστρέφεται σφάλμα. Αν όλα είναι έγκυρα, τα στοιχεία ενημερώνονται, το performance σημειώνεται ως τελικά υποβληθείσα, και αποθηκεύεται. Τέλος, επιστρέφεται επιτυχής απόκριση με μήνυμα «Performance final submission successful» και τα σχετικά δεδομένα.



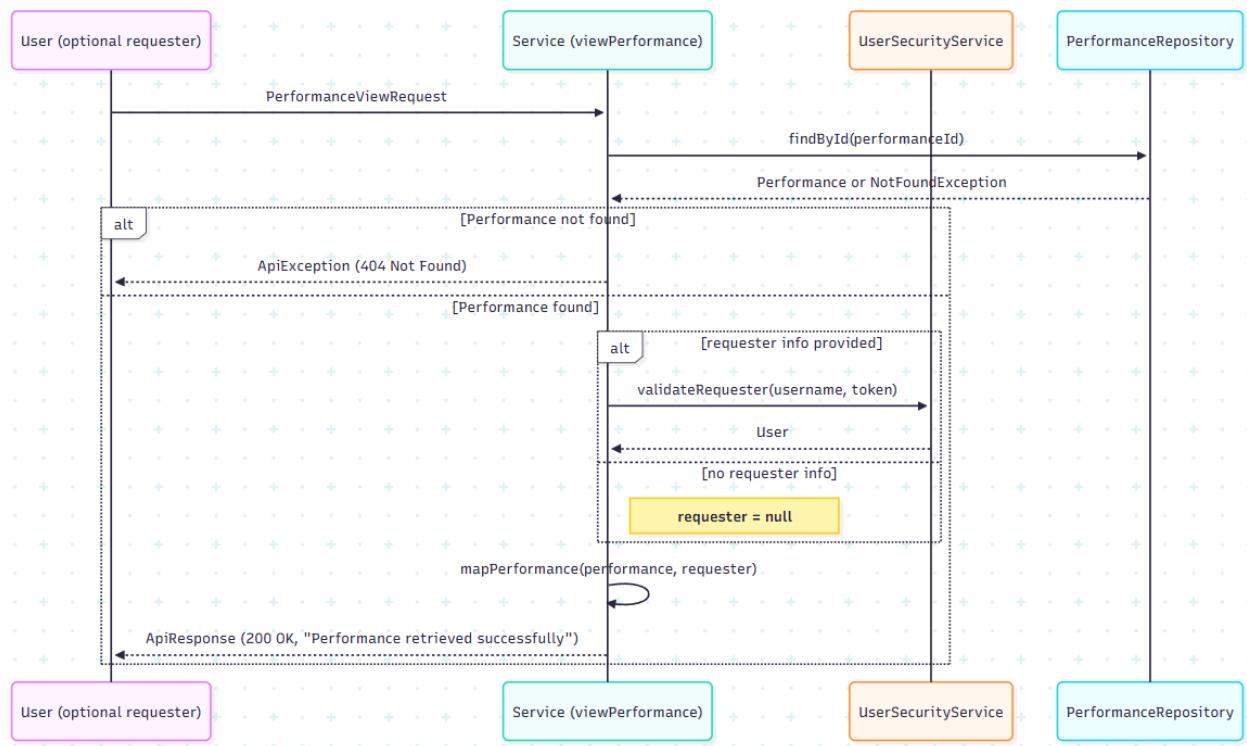
Sequence diagram – performance accept



Το διάγραμμα παρουσιάζει τη διαδικασία αποδοχής ενός performance μετά τη φάση DECISION. Αρχικά, γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος με τη μέθοδο validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα. Στη συνέχεια, το σύστημα αναζητά το performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα. Έπειτα, ελέγχεται ότι ο requester είναι ORGANIZER του φεστιβάλ στο οποίο ανήκει το performance. Αν όχι, επιστρέφεται σφάλμα. Στη συνέχεια, επαληθεύεται ότι το φεστιβάλ βρίσκεται σε κατάσταση DECISION. Αν όχι, επιστρέφεται σφάλμα. Αν όλα τα παραπάνω είναι έγκυρα, η κατάσταση του performance ενημερώνεται σε SCHEDULED και αποθηκεύεται στο repository. Τέλος, επιστρέφεται επιτυχής απόκριση με τα στοιχεία του performance και μήνυμα «Performance accepted/scheduled successfully».



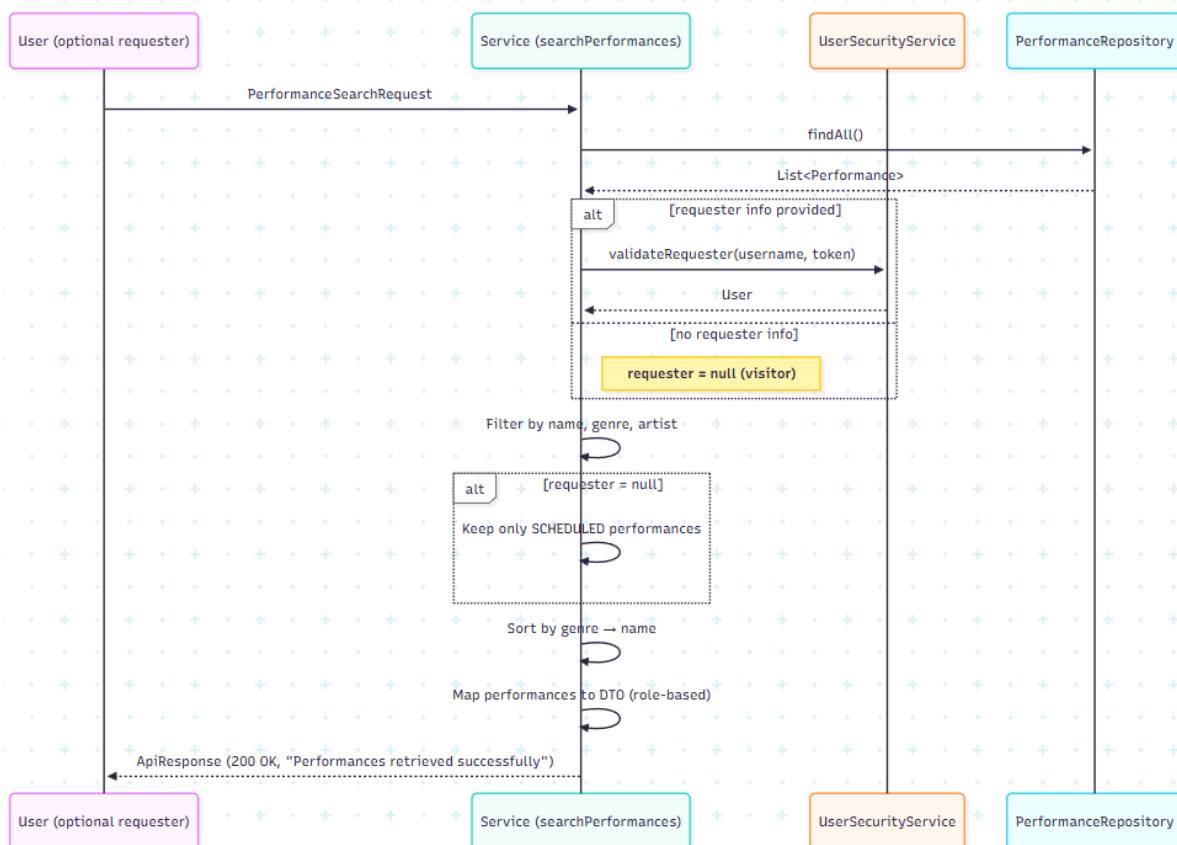
Sequence diagram – view performance



Το διάγραμμα παρουσιάζει τη διαδικασία ανάκτησης λεπτομερειών ενός performance. Αρχικά, το σύστημα αναζητά το performance βάσει του performanceId. Αν δεν βρεθεί, επιστρέφεται σφάλμα. Αν παρέχεται requesterUsername και token, γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος μέσω της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα. Στη συνέχεια, το performance χαρτογραφείται σε μορφή DTO μέσω της μεθόδου mapPerformance, η οποία επιστρέφει διαφορετικές λεπτομέρειες ανάλογα με τον ρόλο του χρήστη (creator, band member, staff, organizer, visitor). Τέλος, επιστρέφεται επιτυχής απόκριση με τα στοιχεία του performance και μήνυμα «Performance retrieved successfully».



Sequence diagram – performance search



Το διάγραμμα παρουσιάζει τη διαδικασία αναζήτησης performances. Αρχικά, το σύστημα ανακτά όλα τα performances από το repository. Αν παρέχεται requesterUsername και token, γίνεται έλεγχος εγκυρότητας του χρήστη αιτούντος μέσω της μεθόδου validateRequester. Αν το token ή τα στοιχεία είναι άκυρα, επιστρέφεται σφάλμα.

Στη συνέχεια, τα performances φιλτράρονται με βάση τα κριτήρια που παρέχονται στο PerformanceSearchRequest:

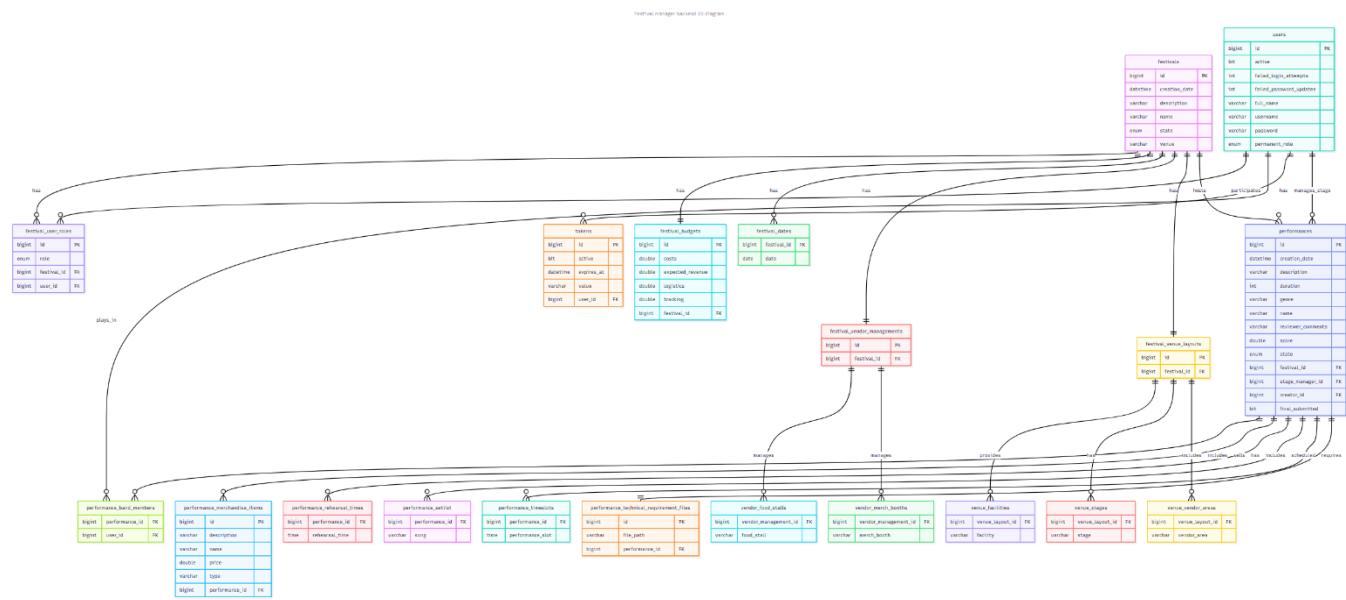
1. Όνομα: Ελέγχεται αν όλα τα λέξεις που περιλαμβάνονται στο όνομα αναζήτησης περιέχονται στο όνομα του performance.
2. Είδος (Genre): Ελέγχεται αν όλες οι λέξεις περιέχονται στο genre του performance.
3. Καλλιτέχνης (Creator ή Band Members): Ελέγχεται αν όλες οι λέξεις ταιριάζουν είτε με τον creator είτε με κάποιο μέλος της μπάντας.

Αν δεν υπάρχει έγκυρος requester (δηλαδή visitor), εμφανίζονται μόνο performances σε κατάσταση SCHEDULED. Τα αποτελέσματα ταξινομούνται πρώτα κατά genre και μετά κατά όνομα. Τέλος, τα performances χαρτογραφούνται σε DTO ανάλογα με τον ρόλο του requester και επιστρέφεται επιτυχής απόκριση με τη λίστα των performances.



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, κατάσταση ενεργοποίησης, καθώς και ο μόνιμος ρόλος του στο σύστημα (Admin - User). Οι χρήστες σχετίζονται με festivals και performances μέσω ρόλων.
 - Festival: Περιλαμβάνει τις πληροφορίες ενός φεστιβάλ, όπως τίτλο, περιγραφή, ημερομηνίες, τοποθεσία και κατάσταση. Κάθε festival μπορεί να περιέχει:
 - Προϋπολογισμό (budget).
 - Σχέδιο τοποθεσίας (venue layout) με σκηνές, εγκαταστάσεις και χώρους πωλητών.
 - Vendors (φαγητό, εμπορεύματα).
 - Πολλαπλές παραστάσεις (performances).

Οι ρόλοι των χρηστών σε ένα festival ορίζονται μέσω της ενδιάμεσης οντότητας festival user roles.

- Performance: Αντιπροσωπεύει μια καλλιτεχνική συμμετοχή σε ένα festival. Περιλαμβάνει στοιχεία όπως τίτλο, περιγραφή, διάρκεια, είδος μουσικής/τέχνης, κατάσταση έγκρισης, σχόλια αξιολογητών, σκορ και αν έχει υποβληθεί τελικά. Κάθε performance συνδέεται με:
 - Τον καλλιτέχνη (artist).
 - Τον stage manager (staff).



- Ένα μόνο festival.
Επιπλέον, μπορεί να περιλαμβάνει setlist, ώρες πρόβας, timeslots, merchandise items και αρχεία τεχνικών απαιτήσεων.
- Festival Budget: Ορίζει οικονομικά στοιχεία ενός φεστιβάλ (κόστη, έσοδα, logistics, tracking). Υπάρχει σχέση 1:1 με το festival.
- Festival Dates: Αποθηκεύει τις ημερομηνίες διεξαγωγής ενός φεστιβάλ (1:N με festival).
- Festival User Roles: Ενδιάμεση οντότητα που συσχετίζει χρήστες με festivals και αποθηκεύει τον ρόλο τους (π.χ. Organizer, Staff, Artist).
- Vendor Management: Αφορά τη διαχείριση πωλητών μέσα στο festival (φαγητό, merch booths).
- Venue Layouts: Αναπαριστούν την τοποθεσία/κάτοψη του festival και περιέχουν επιμέρους στοιχεία: σκηνές, εγκαταστάσεις, vendor areas.
- Performance Band Members: Συσχέτιση πολλών-προς-πολλά μεταξύ users (artists) και performances.
- Performance Merchandise Items: Προϊόντα που πωλούνται σε ένα performance (π.χ. CD, μπλουζάκια).
- Performance Rehearsal Times / Timeslots: Καταγράφουν ώρες πρόβας και ώρες πραγματοποίησης της παράστασης.
- Performance Setlist: Περιλαμβάνει τα τραγούδια/κομμάτια μιας παράστασης.
- Performance Technical Requirement Files: Αρχεία που περιγράφουν τεχνικές απαιτήσεις για την εκτέλεση μιας performance.
- Tokens: Σχετίζονται με τους users και εξυπηρετούν τον μηχανισμό authentication/authorization.

Συσχετίσεις

- Festival – Performance (1:N): Κάθε festival φιλοξενεί πολλά performances.
- Performance – User (M:N μέσω band_members): Ένα performance μπορεί να έχει πολλούς καλλιτέχνες και κάθε καλλιτέχνης να συμμετέχει σε πολλά performances.
- Festival – User (M:N μέσω festival_user_roles): Οι χρήστες αναλαμβάνουν διαφορετικούς ρόλους σε διαφορετικά festivals.
- Festival – Budget (1:1): Κάθε festival έχει έναν προϋπολογισμό.
- Festival – Dates (1:N): Ένα festival μπορεί να εκτείνεται σε πολλές ημερομηνίες.
- Festival – Vendor Management (1:1) → Food Stalls / Merch Booths (1:N): Περιγράφεται η διαχείριση πωλητών.
- Festival – Venue Layout (1:1) → Facilities / Stages / Vendor Areas (1:N): Ο χώρος του festival με αναλυτικά στοιχεία υποδομής.
- Performance – Rehearsals, Setlists, Timeslots (1:N): Κάθε performance έχει πολλαπλά βοηθητικά στοιχεία.



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμπικλής

- Performance – Technical Requirement Files (1:1): Κάθε performance συνδέεται με αρχεία τεχνικών απαιτήσεων.
- User – Tokens (1:N): Οι χρήστες διαθέτουν tokens για login/session.

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.



5 Υλοποίηση Συστήματος

5.1 GitHub

Η ανάπτυξη του έργου πραγματοποιήθηκε μέσω GitHub, σε ένα ιδιωτικό αποθετήριο με τίτλο [Festival-Management-Backend](#).

Το αποθετήριο είναι διαθέσιμο στη διεύθυνση:

<https://github.com/stelios1361/Festival-Management-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.

Η οργάνωση του κώδικα είναι ομαδοποιημένη 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 UUID 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("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("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$10$ey6vbL6idFb0abibQFpr3ONU6j8zKjZSj8CImVrVDkbogsgLDwbWY
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("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("testMatchFails_forWrongPassword completed successfully\n");
}
```



```
-----  
T E S T S  
-----  
Running com.festivalmanager.security.PasswordServiceTest  
== PasswordServiceTest setup completed ==  
Running testMatchFails_forWrongPassword:  
Raw password: password  
Hashed password: $2a$10$htm98cY9AaQWI9wdiSYqHu/WGI3zlaznZ8ad3zuq7KBLXpNfx7lMi  
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(Optional.of(value: user));  
    when(methodCall: tokenService.validateToken(value: "token123", requestingUser: user)).thenReturn(true);  
  
    System.out.println("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("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(Optional.empty());  
  
    System.out.println("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("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(Optional.of(value: user));

    System.out.println("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("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(Optional.of(value: user));
    doThrow(new ApiException(message: "Invalid token", status: HttpStatus.UNAUTHORIZED))
        .when(e: tokenService).validateToken(value: "badToken", requestingUser: user);

    System.out.println("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("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:

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

    System.out.println(: "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(: "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(: Collections.singletonList(: token));
    when(methodCall: tokenRepository.saveAll(entities: anyList())).thenReturn(: Collections.singletonList(: token));

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

    tokenService.deactivateTokens(user);

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

    System.out.println(: "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).

```
@Test  
void testDeleteTokens() {  
    User user = new User();  
  
    System.out.println("Running testDeleteTokens");  
  
    tokenService.deleteTokens(user);  
  
    verify(mockTokenRepository).deleteByUser(user);  
  
    System.out.println("testDeleteTokens completed successfully\\n");  
}
```

```
==== TokenServiceTest setup completed ====  
  
Running testDeleteTokens  
testDeleteTokens completed successfully  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.995 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 επιστρέψει 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("Running testValidateToken_success");

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

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

    System.out.println("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("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("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.

```
@Test  
void testRegisterUser_success() {  
    RegisterRequest req = new RegisterRequest();  
    req.setUsername(username: "alicel");  
    req.setPassword1(password1: "Strong@123");  
    req.setPassword2(password2: "Strong@123");  
    req.setFullscreen(fullname: "Alice Example");  
  
    when(methodCall: userRepository.existsByUsername(username: "alicel")).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("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("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(true);

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

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

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

    System.out.println("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(Optional.of(value: user));
    when(methodCall: passwordService.matches(rawPassword: "pw123", hashedPassword: "hashedPw")).thenReturn(true);
    when(methodCall: tokenService.generateToken(user)).thenReturn(new com.festivalmanager.model.Token());

    System.out.println("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("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(: Optional.of(value: user));
    when(methodCall: passwordService.matches(rawPassword: "wrong", hashedPassword: "hashedPw")).thenReturn(: false);

    System.out.println(: "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(: "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(user);
    when(methodCall: passwordService.matches(rawPassword: "old", hashedPassword: "hashedOld")).thenReturn(true);
    when(methodCall: passwordService.hash(rawPassword: "New@1234")).thenReturn("hashedNew");
    when(methodCall: tokenService.generateToken(user)).thenReturn(new com.festivalmanager.model.Token());

    System.out.println("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("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(user);

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

    userService.deleteUser(request: req);

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

    System.out.println("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@l23");
    req.setPassword2(password2: "Strong@l23");
    req.setFullscreen(fullname: "Invalid Name");

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

    System.out.println("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("Invalid username"));

    System.out.println("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.setFullscreen(fullname: "Weak Password User");

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

    System.out.println("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("Password must be at least 8 characters"));

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



```
== UserSTest 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.

The screenshot shows the Postman interface with a POST request to `http://localhost:8080/api/users/register`. The request body is set to `raw` and contains the following JSON:

```
1 {
2   "username": "user1",
3   "fullname": "User One",
4   "password1": "Password@123",
5   "password2": "Password@123"
6 }
```

The response status is `200 OK`. The response body is:

```
1 {
2   "timestamp": "2025-09-17T21:16:32.486998",
3   "status": 200,
4   "message": "User registered successfully",
5   "data": []
6 }
```



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

The screenshot shows a POST request to `http://localhost:8080/api/users/login`. The Body tab is selected, showing raw JSON input:

```
1 {
2     "username": "user1",
3     "password": "Password@123"
4 }
```

The response tab shows a 200 OK status with the following JSON body:

```
1 {
2     "timestamp": "2025-09-17T21:22:41.338Z",
3     "status": 200,
4     "message": "Login successful",
5     "data": [
6         {
7             "expiresAt": "2025-09-17T23:22:41.324Z",
8             "token": "236f31ba-2952-4ca0-a0af-5a703ee2fb89"
9     ]}
```

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

The screenshot shows a POST request to `http://localhost:8080/api/users/login`. The Body tab is selected, showing raw JSON input:

```
1 {
2     "username": "user2",
3     "password": "Password@123"
4 }
```

The response tab shows a 403 Forbidden status with the following JSON body:

```
1 {
2     "timestamp": "2025-09-17T21:24:45.110Z",
3     "status": 403,
4     "message": "Account is deactivated. Please contact admin.",
5     "data": null
6 }
```



- Εκτελούμε updateAccountStatus για τον user2 και βλέπουμε ότι ο user2 έχει ενεργοποιηθεί.

POST http://localhost:8080/api/users/updateaccountstatus

Params Authorization Headers (9) Body Scripts Settings

(none) form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "requesterUsername": "user1",  
3   "token": "236f31ba-2952-4ca0-a0af-5a703ee2fb09",  
4   "targetUsername": "user2",  
5   "newActive": true  
6 }
```

Body Cookies Headers (5) Test Results | ⏱

{ } JSON Preview Visualize

```
1 {  
2   "timestamp": "2025-09-17T21:39:18.3187439",  
3   "status": 200,  
4   "message": "User account status updated successfully",  
5   "data": {}  
6 }
```

200 OK 233 ms 2

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

POST http://localhost:8080/api/users/login

Params Authorization Headers (9) Body Scripts Settings

(none) form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "username": "user2",  
3   "password": "Password@123"  
4 }
```

Body Cookies Headers (5) Test Results | ⏱

{ } JSON Preview Visualize

```
1 {  
2   "timestamp": "2025-09-17T21:41:11.5742627",  
3   "status": 200,  
4   "message": "Login successful",  
5   "data": [  
6     {"expiresAt": "2025-09-17T23:41:11.1439257",  
7     "token": "430dfda4-b223-4986-be8c-3eef3369ea7a"}  
8   ]  
9 }
```

200 OK 499 ms 3



- Κάνουμε αλλαγή του ονόματος του δεύτερου χρήστη.

The screenshot shows a Postman request to `http://localhost:8080/api/users/updateuserinfo`. The request method is POST. The Body tab contains the following JSON payload:

```
1 {  
2   "requesterUsername": "user2",  
3   "token": "430dfda4-b223-4986-be8c-3eef3369ea7a",  
4   "newFullName": "User two Updated"  
5 }
```

The response status is 200 OK, with a timestamp of 2025-09-17T21:43:11.3669408, status 200, message "User information updated successfully", and an empty data object.

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

The screenshot shows a Postman request to `http://localhost:8080/api/users/updateuserpassword`. The request method is POST. The Body tab contains the following JSON payload:

```
1 {  
2   "requesterUsername": "user2",  
3   "token": "430dfda4-b223-4986-be8c-3eef3369ea7a",  
4   "oldPassword": "Password@123",  
5   "newPassword1": "NewPass@123",  
6   "newPassword2": "NewPass@123"  
7 }
```

The response status is 200 OK, with a timestamp of 2025-09-17T21:44:45.9729334, status 200, message "Password updated successfully", and a data object containing an expiresAt timestamp (2025-09-17T23:44:46.5951683) and a new token value (2d7592d4-e152-4dad-98e1-faebe67234c87).



- Βλέπουμε ότι ο χρήστης 2 κάνει επιτυχώς logout.

```
POST http://localhost:8080/api/users/logout

Body (raw JSON)
{
  "requesterUsername": "user2",
  "token": "2d7592d4-e152-4dad-98c1-faeb67234c87"
}

200 OK | 14 ms | 2
{} JSON | Preview | Visualize | ↴
{
  "timestamp": "2025-09-17T21:46:13.6683285",
  "status": 200,
  "message": "User logged out successfully",
  "data": {}
}
```

- Ταυτόχρονα με το logout το token απενεργοποιείται.

```
Body (empty) | Cookies | Headers (5) | Test Results | ↴
{} JSON | Preview | Visualize | ↴
401 Unauthorized | 13 ms | ↴
{
  "timestamp": "2025-09-17T21:48:18.688315",
  "status": 401,
  "message": "Token is inactive",
  "data": null
}
```



- Τέλος βλέπουμε ότι ο user2 μπορεί να διαγράψει τον εαυτό του.

POST http://localhost:8080/api/users/delete

Params Authorization Headers (9) Body Scripts Settings

Body (raw JSON)

```
1 {
2   "requesterUsername": "user2",
3   "token": "3a7527b5-d735-4147-89f6-15de1ccd46cb"
4 }
```

Body Cookies Headers (5) Test Results 200 OK 468 ms 26

{ JSON Preview Visualize }

```
1 {
2   "timestamp": "2025-09-17T21:51:19.6703339",
3   "status": 200,
4   "message": "User deleted successfully",
5   "data": {}
6 }
```

Festival creation and organizer assignment

- Αρχικά δημιουργούμε έναν χρήστη ακόμα με νέα στοιχεία.

POST http://localhost:8080/api/users/register

Params Authorization Headers (9) Body Scripts Settings

Body (raw JSON)

```
1 {
2   "username": "organizer1",
3   "fullname": "Organizer One",
4   "password1": "Organizer@123",
5   "password2": "Organizer@123"
6 }
```

Body Cookies Headers (5) Test Results 200 OK 74 ms 26

{ JSON Preview Visualize }

```
1 {
2   "timestamp": "2025-09-17T22:07:35.1851626",
3   "status": 200,
4   "message": "User registered successfully",
5   "data": {}
6 }
```



- Κάνουμε activate τον νέο χρήστη.

POST http://localhost:8080/api/users/updateaccountstatus

Params Authorization Headers (9) Body Scripts Settings

(none form-data x-www-form-urlencoded raw binary GraphQL JSON)

```
1
2
3 {
4     "requesterUsername": "user1",
5     "token": "236f31ba-2952-4ca0-a0af-5a783ee2fb09",
6     "targetUsername": "organizer1",
7     "newActive": true
}
```

Body Cookies Headers (5) Test Results (1)

{ } JSON Preview Visualize

200 OK 13 ms 2

```
1 {
2     "timestamp": "2025-09-17T22:10:17.7303622",
3     "status": 200,
4     "message": "User account status updated successfully",
5     "data": {}
}
```

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

POST http://localhost:8080/api/users/login

Params Authorization Headers (9) Body Scripts Settings

(none form-data x-www-form-urlencoded raw binary GraphQL JSON)

```
1
2
3 {
4     "username": "organizer1",
5     "password": "Organizer@123"
}
```

Body Cookies Headers (5) Test Results (1)

{ } JSON Preview Visualize

200 OK 64 ms 3

```
1 {
2     "timestamp": "2025-09-17T22:11:30.1728003",
3     "status": 200,
4     "message": "Login successful",
5     "data": {
6         "expiresAt": "2025-09-18T00:11:30.1698111",
7         "token": "c4a1c871-4878-4307-b7dd-3daff62bd61b9"
8     }
9 }
```



- Δημιουργία φεστιβάλ δίνοντας τα σωστά στοιχεία του φεστιβάλ.

```
POST http://localhost:8080/api/festivals/createstival
```

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "requesterUsername": "organizer1",
3   "token": "c4a1c871-4878-4307-b7dd-3daf62bd61b9",
4   "name": "Rock Summer 2025",
5   "description": "Three days of rock music under the stars.",
6   "dates": [
7     "2025-07-01",
8     "2025-07-02",
9     "2025-07-03"
10  ],
11  "venue": "Athens Olympic Stadium"
12 }
```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
200 OK 186 ms 32
```

```
1 {
2   "timestamp": "2025-09-17T22:38:11.0726786",
3   "status": 200,
4   "message": "Festival created successfully",
5   "data": {
6     "organizer": "organizer1",
7     "name": "Rock Summer 2025",
8     "id": 1
9   }
10 }
```

- Βλέπουμε τα στοιχεία του φεστιβάλ κάνοντας χρήση του viewfestival.

```
GET http://localhost:8080/api/festivals/viewfestival
```

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "requesterUsername": "organizer1",
3   "token": "c4a1c871-4878-4307-b7dd-3daf62bd61b9",
4   "festivalId": 1
5 }
```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
200 OK 29 ms 55
```

```
1 {
2   "timestamp": "2025-09-17T22:33:27.4185283",
3   "status": 200,
4   "message": "Festival retrieved successfully",
5   "data": {
6     "festival": {
7       "id": 1,
8       "name": "Rock Summer 2025",
9       "description": "Three days of rock music under the stars.",
10      "venue": "Athens Olympic Stadium",
11      "dates": [
12        "2025-07-03",
13        "2025-07-02",
14        "2025-07-01"
15      ],
16      "organizers": [
17        "organizer1"
18      ],
19      "venueLayout": null,
20      "budget": null,
21      "vendorManagement": null,
22      "staff": []
23    }
24  }
25 }
```



- Πραγματοποιείται αλλαγή στα στοιχεία του φεστιβάλ.

The screenshot shows a POSTMAN interface with the following details:

- Method:** PUT
- URL:** http://localhost:8080/api/festivals/updatefestival
- Headers:** (9 items)
- Body:** raw JSON (selected)
- JSON Content:**

```
7  "dates": [
8    "2025-07-01",
9    "2025-07-02",
10   "2025-07-03",
11   "2025-07-04"
12 ],
13 "venueLayout": {
14   "stages": ["Main Stage", "Electronic Stage"],
15   "vendorAreas": ["Food Court", "Merch Zone"],
16   "facilities": ["Restrooms", "Info Booth"]
17 },
18 "vendorManagement": {
19   "foodStalls": ["BurgerKing", "VeganBites"],
20   "merchandiseBooths": ["BandMerch", "VinylShop"]
21 },
22 "budget": {
23   "tracking": 10000.0,
24   "costs": 75000.0,
25   "logistics": 20000.0,
26   "expectedRevenue": 150000.0
27 },
28 "organizers": [ "user2", "user3" ],
29 "staff": [ "user4", "user5" ]
30 }
```
- Response:** 200 OK (658 ms)

The response body is also displayed in JSON format:

```
1  {
2   "timestamp": "2025-09-17T22:38:58.5156193",
3   "status": 200,
4   "message": "Festival updated successfully",
5   "data": {
6     "identifier": "0338e2e9-6faa-4996-9490-0ba9e4d063bb",
7     "name": "Rock Summer 2025 Reloaded",
8     "organizers": [
9       "user2",
10      "organizer1",
11      "user3"
12     ],
13   }
14 }
```



321-4002 – Τεχνολογία Λογισμικού
Τίτλος Μελέτης: Festival Management System
Στυλιανός Νικολόπουλος, Γεώργιος Αναγνωστόπουλος, Γεώργιος Καμποτσικλής

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

GET http://localhost:8080/api/festivals/viewfestival

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

Body Cookies Headers (5) Test Results | ⏱

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

200 OK ⏱ 2.75 s ⏵

```
        "-----"
20    ],
21    "users3": [
22      "venueLayout": {
23        "stages": [
24          "Electronic Stage",
25          "Main Stage"
26        ],
27        "vendorAreas": [
28          "Merch Zone",
29          "Food Court"
30        ],
31        "facilities": [
32          "Info Booth",
33          "Restrooms"
34        ]
35      },
36      "budget": {
37        "tracking": 100000.0,
38        "costs": 75000.0,
39        "logistics": 20000.0,
40        "expectedRevenue": 150000.0
41      },
42      "vendorManagement": {
43        "foodStalls": [
44          "BurgerKing",
45          "VeganBites"
46        ],
47        "merchandiseBooths": [
48          "VinylShop",
49          "BandMerch"
50        ]
51      },
52      "staff": [
53        "user5",
54        "user4"
55      ]
56    }
57  }
58 }
```



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 ριών δεδομένων ανάμεσα στους χρήστες και το σύστημα. Επιπλέον, προκλήσεις παρουσιάστηκαν στην ενσωμάτωση των κατάλληλων μηχανισμών ασφάλειας για την ανθεντικοποίηση χρηστών.

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 η γνωστή αρχιτεκτονική τριών επιπέδων (Controller–Service–Repository) για τον διαχωρισμό ευθυνών, γεγονός που ενίσχυσε το modularity και τη δυνατότητα επαναχρησιμοποίησης του κώδικα. Επίσης, η τεκμηρίωση μέσω 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.