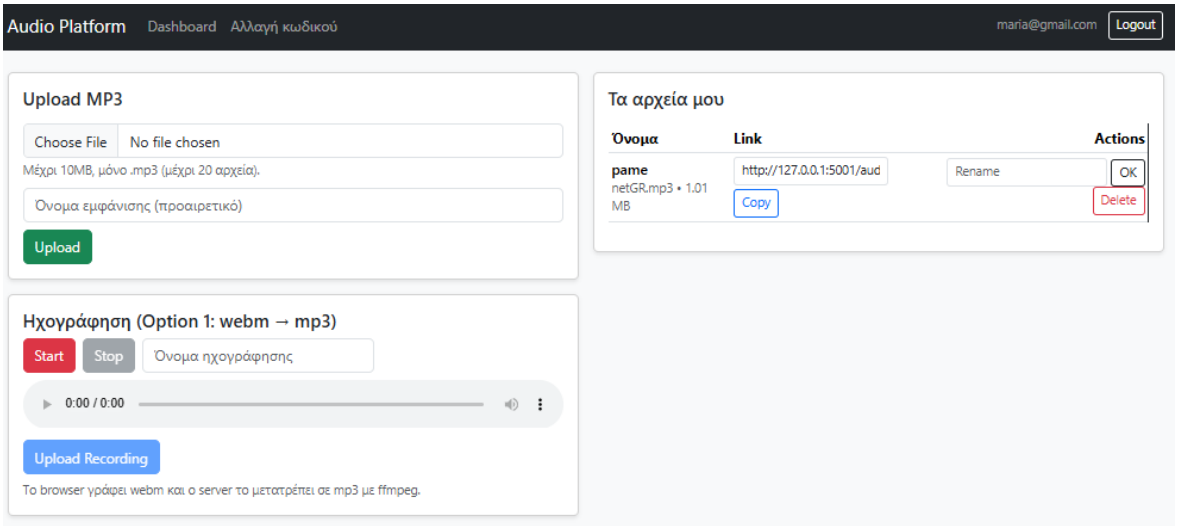
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Πλατφόρμα οργάνωσης αρχείων ήχου για ενσωμάτωση σε διαδραστικό εκπαιδευτικό υλικό»



MELINA ISMAILATI

2021195

Επιβλέπων

Τσιακμάκης Κυριάκος

Ιανουάριος

2026

Πλατφόρμα οργάνωσης αρχείων ήχου για ενσωμάτωση σε διαδραστικό εκπαιδευτικό υλικό

Κωδικός: **25338**

Φοιτητής: Melina Ismailati

Εισηγητής: Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 29-10-2025

Ημερομηνία περάτωσης Π.Ε. 20-01-2026

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

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει τη σχεδίαση και υλοποίηση μιας διαδικτυακής πλατφόρμας οργάνωσης και φιλοξενίας αρχείων ήχου (mp3) με σκοπό την ενσωμάτωσή τους σε διαδραστικό εκπαιδευτικό υλικό στο Genially. Το Genially δεν επιτρέπει απευθείας μεταφόρτωση mp3, αλλά δέχεται URL που καταλήγει σε αρχείο ".mp3", γεγονός που δημιουργεί πρακτικές δυσκολίες σε εκπαιδευτικά σενάρια. Η πλατφόρμα αναπτύχθηκε με Python (Flask), MySQL και Bootstrap, παρέχοντας σύστημα εισόδου χρηστών, μεταφόρτωση mp3 με περιορισμούς (έως 20 αρχεία ανά χρήστη και μέγιστο μέγεθος 10MB ανά αρχείο), άμεση παραγωγή δημόσιου συνδέσμου με σταθερό όνομα (uuid), καθώς και δυνατότητες μετονομασίας και διαγραφής χωρίς να σπάει το URL. Επιπλέον, υλοποιήθηκε ολοκληρωμένο περιβάλλον διαχείρισης για τον admin με λίστες χρηστών/αρχείων και στατιστικά χρήσης.

« Audio file organization platform for integration into interactive educational material»

Abstract

This thesis presents the design and implementation of a web platform for organizing and hosting audio files (MP3) intended for integration into interactive educational content created with Genially. Genially does not allow direct MP3 uploads inside a project; instead, it supports audio insertion only through a URL that ends with a ".mp3" file, which creates practical difficulties for educational use cases. The platform was developed using Python (Flask), MySQL, and Bootstrap, providing user authentication, MP3 uploads with constraints (up to 20 files per user and a maximum size of 10 MB per file), and immediate generation of a public, stable audio link based on a unique identifier (UUID). Users can rename files for better organization and delete them when no longer needed, without breaking previously used URLs. In addition, a complete administration area was implemented, including user and file management pages and usage statistics.

Ευχαριστίες

Να ευχαριστήσω όσους με βοήθησαν για την ολοκλήρωση αυτής της εργασίας και τον επιβλέπων για την συνεχή καθοδήγηση σε κάθε βήμα.

Περιεχόμενα

Περίληψη iv
Abstract v
Ευχαριστίες vi
Περιεχόμενα vii
Κατάλογος Σχημάτων x
Κεφάλαιο 1ο: Εισαγωγή 11
1.1 Γενικό πλαίσιο και κίνητρο 11
1.2 Περιγραφή προβλήματος 12
1.3 Στόχος της εργασίας 12
1.4 Τεχνολογίες και εργαλεία που χρησιμοποιήθηκαν 13
1.5 Βασική ιδέα λειτουργίας της πλατφόρμας 14
1.6 Δομή της διπλωματικής εργασίας 15
Κεφάλαιο 2ο: Θεωρητική προσέγγιση 16
2.1 Διαδικτυακές εφαρμογές και μοντέλο client–server 16
2.2 Η έννοια του URL και γιατί μας ενδιαφέρει να τελειώνει σε .mp3 16
2.3 Flask: Τι είναι και γιατί επιλέχθηκε 17
2.3.1 Routes και endpoints 17
2.4 Templates και Bootstrap: γιατί μας βοηθάνε 17
2.5 Βάση δεδομένων MySQL: ρόλος και χρησιμότητα 18
2.5.1 Τι αποθηκεύουμε στη βάση 18
2.6 Authentication και ρόλοι (User/Admin) 19
2.6.1 “Προκαθορισμένοι” λογαριασμοί με κωδικό (claim flow) 19
2.7 Αποθήκευση αρχείων (File Uploads) και περιορισμοί 20
2.8 UUID και “σταθερό” public link 20
2.9 Serving mp3 και έννοια του streaming 21
2.10 Βασικά θέματα ασφάλειας 21
2.10.1 Hashing κωδικών 21
2.10.2 CSRF protection 21
2.10.3 Έλεγχοι upload 22

2.10.4 Πρόσβαση admin endpoints 22
2.11 Ηχογράφηση από μικρόφωνο και μετατροπή σε mp3 22
Κεφάλαιο 3ο: Προδιαγραφές και στόχοι λειτουργίας του συστήματος 23
3.1 Εισαγωγή 23
3.2 Τι προσφέρει η πλατφόρμα στον χρήστη 23
3.2.1 Είσοδος στο σύστημα και dashboard 23
3.2.2 Ενεργοποίηση λογαριασμού με κωδικό (λογική “claim”) 23
3.2.3 Μεταφόρτωση mp3 και άμεσο link 24
3.2.4 Μετονομασία “εμφάνισης” (rename χωρίς αλλαγή URL) 24
3.2.5 Διαγραφή αρχείου 24
3.2.6 Αλλαγή κωδικού σε ξεχωριστή σελίδα 24
3.3 Περιορισμοί χρήσης (ώστε να μην ξεφύγει το σύστημα) 25
3.4 Λειτουργίες διαχειριστή (admin) και έλεγχος συστήματος 25
3.5 Συμβατότητα με Genially 27
3.6 Σενάρια χρήσης 27
3.6.1 Σενάριο χρήστη: upload και χρήση στο Genially 27
3.6.2 Σενάριο χρήστη: rename χωρίς να σπάσει το link 27
3.6.3 Σενάριο διαχειριστή: δημιουργία πρόσβασης σε φοιτητή 27
Κεφάλαιο 4ο: Σχεδίαση και αρχιτεκτονική συστήματος 28
4.1 Εισαγωγή 28
4.2 Αρχιτεκτονική υψηλού επιπέδου 28
4.3 Εσωτερική οργάνωση 29
4.4 Σχεδίαση ροής “Claim Account” (email + κωδικός) 29
4.5 Σχεδίαση ροής upload και παραγωγής public mp3 link 30
4.6 Σχεδίαση δημόσιου endpoint /audio/<uuid>.mp3 (για Genially) 32
4.7 Σχεδίαση admin υποσυστήματος (σε ξεχωριστές σελίδες) 32
4.8 Σχεδίαση σελίδας αλλαγής κωδικού (ξεχωριστή σελίδα) 33
Κεφάλαιο 5ο: Σχεδίαση βάσης δεδομένων και μοντέλο δεδομένων 35
5.1 Εισαγωγή 35
5.2 Οντότητες και πίνακες 35
5.3 ER διάγραμμα (Entity–Relationship) 35
5.4 Πίνακας users (περιγραφή πεδίων) 37
5.5 Πίνακας audio_files (περιγραφή πεδίων) 37
5.6 Κανόνες και περιορισμοί σε επίπεδο δεδομένων 38
Κεφάλαιο 6ο: Υλοποίηση της πλατφόρμας 39

6.1 Εισαγωγή 39
6.2 Δομή του project (φάκελοι και αρχεία) 39
6.3 Ρυθμίσεις και βασικές παράμετροι λειτουργίας 39
6.4 Υλοποίηση εισόδου (login) και συνεδρίας χρήστη 40
6.5 Υλοποίηση “Claim” λογαριασμού (email + κωδικός) 40
6.6 Upload mp3 με όρια (10MB, 20 αρχεία) 40
6.7 Δημιουργία public link και σταθερό URL για Genially 41
6.8 Serving mp3 με byte-range υποστήριξη 41
6.9 Rename και delete αρχείων 41
6.10 Αλλαγή κωδικού σε ξεχωριστή σελίδα 42
6.11 Admin περιβάλλον (ξεχωριστές σελίδες) 42
Κεφάλαιο 7ο: Δοκιμή 44
7.1 Εισαγωγή 44
7.2 Προσέγγιση δοκιμών 44
7.3 Σενάρια δοκιμών χρήστη (User) 44
Κεφάλαιο 8ο: Παρουσίαση εφαρμογής και ενδεικτική χρήση 50
8.1 Σελίδα σύνδεσης χρήστη 50
8.2 Dashboard χρήστη (λίστα αρχείων και upload) 51
8.3 Παράδειγμα δημιουργίας και χρήσης public link 52
8.4 Rename αρχείου χωρίς να σπάει το Genially link 53
8.5 Διαγραφή αρχείου και απελευθέρωση χώρου/ορίων 54
8.6 Σελίδα αλλαγής κωδικού (ξεχωριστή) 54
8.7 Admin περιβάλλον (εποπτεία και διαχείριση) 55
8.7.1 Admin Στατιστικά (/admin) 55
8.7.2 Διαχείριση χρηστών (/admin/users) 55
8.7.3 Διαχείριση αρχείων (/admin/files) 56
Κεφάλαιο 9ο: Συμπεράσματα και μελλοντικές επεκτάσεις 58
9.1 Πλεονεκτήματα της προσέγγισης 58
9.2 Δυσκολίες και σημεία που χρειάζονται προσοχή 59
9.3 Προτάσεις για μελλοντικές επεκτάσεις 60
ΒΙΒΛΙΟΓΡΑΦΙΑ 62
ΠΑΡΑΡΤΗΜΑ Α 63

Κατάλογος Σχημάτων

Εικόνα 4.1: Αρχιτεκτονική του συστήματος (Client–Server–DB–Storage–Genially)-Αρχιτεκτονική υψηλού επιπέδου της πλατφόρμας..... 28
Εικόνα 4.2: Ροή claim λογαριασμού με email και κωδικό..... 30
Εικόνα 4.3: Ροή upload mp3 και παραγωγής public link..... 31
Εικόνα 4.4: Ροή upload mp3 και παραγωγής public link - Ροή εξυπηρέτησης mp3 προς Genially 32
Εικόνα 4.5: Πλοήγηση και ενότητες του admin περιβάλλοντος..... 33
Εικόνα 4.6: Ροή αλλαγής κωδικού χρήστη..... 34
Εικόνα 5.1: ER διάγραμμα της βάσης δεδομένων (Users–AudioFiles)..... 36
Εικόνα 8.1: Σελίδα σύνδεσης (Login) 50
Εικόνα 8.2: Dashboard χρήστη με φόρμα upload και λίστα αρχείων για (α) χρήστη (β) admin 51
Εικόνα 8.3: Εμφάνιση public link μετά το upload..... 52
Εικόνα 8.4: Εισαγωγή ήχου στο Genially μέσω URL..... 53
Εικόνα 8.5: Μετονομασία αρχείου (display name) χωρίς αλλαγή URL..... 53
Εικόνα 8.6: Σελίδα αλλαγής κωδικού χρήστη 54
Εικόνα 8.7: Admin Dashboard με στατιστικά και γραφήματα 55
Εικόνα 8.8: Λίστα χρηστών και δημιουργία pending χρήστη με κωδικό..... 56
Εικόνα 8.9: Λίστα αρχείων (admin) με φίλτρα και δυνατότητα διαγραφής 57

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενικό πλαίσιο και κίνητρο

Τα τελευταία χρόνια, όλο και περισσότερο εκπαιδευτικό υλικό μεταφέρεται σε ψηφιακή μορφή και ειδικά σε μορφές που είναι διαδραστικές, ώστε ο μαθητής/φοιτητής να μην είναι απλά "θεατής", αλλά να συμμετέχει ενεργά. Πλατφόρμες όπως το Genially δίνουν τη δυνατότητα να φτιάξουμε παρουσιάσεις, μαθήματα, κουίζ, παιχνίδια γνώσεων και γενικά περιεχόμενο που μπορεί να εμπλουτιστεί με εικόνες, βίντεο, κουμπιά, κινήσεις και άλλα στοιχεία που κάνουν την εμπειρία πιο ζωντανή [1].

Ένα από τα πιο χρήσιμα στοιχεία σε τέτοιο υλικό είναι ο ήχος. Ο ήχος μπορεί να χρησιμοποιηθεί σαν αφήγηση, σαν επεξήγηση μιας διαφάνειας, σαν οδηγία σε ένα παιχνίδι, σαν παράδειγμα (π.χ. σε γλωσσικά μαθήματα), ή ακόμα και σαν ηχητικό ερέθισμα/feedback όταν ο χρήστης απαντά σωστά ή λάθος. Πρακτικά, σε πολλές εκπαιδευτικές δραστηριότητες, ο ήχος δεν είναι "διακοσμητικός" αλλά βασικό κομμάτι της διδακτικής εμπειρίας.

Το πρόβλημα όμως στην πράξη είναι ότι, ενώ θεωρητικά η ενσωμάτωση ήχου σε πλατφόρμες διαδραστικού υλικού φαίνεται εύκολη, συχνά υπάρχουν περιορισμοί που δυσκολεύουν την άμεση χρήση. Στην παρούσα εργασία, ο βασικός περιορισμός που συναντήθηκε είναι ότι το Genially δεν επιτρέπει την απευθείας μεταφόρτωση (upload) αρχείων ήχου mp3 μέσα στο έργο. Αντίθετα, επιτρέπει την εισαγωγή ήχου μόνο μέσω URL, δηλαδή μέσω συνδέσμου που "δείχνει" σε ένα αρχείο ήχου, και μάλιστα με την προϋπόθεση ότι το URL καταλήγει ξεκάθαρα σε .../name_of_file.mp3. Με απλά λόγια: ο ήχος πρέπει να φιλοξενείται κάπου αλλού και το Genially απλά τον "τραβάει" από εκεί [2].

Αυτό οδηγεί σε ένα πρακτικό και πραγματικό κενό: αν ένας εκπαιδευτικός ή ένας φοιτητής θέλει να βάλει δικούς του ήχους (π.χ. ηχογραφήσεις, μικρά αποσπάσματα, οδηγίες), πρέπει να βρει έναν τρόπο να:

1. ανεβάσει τα mp3 σε κάποιο server/χώρο που να τα σερβίρει δημόσια,
2. πάρει ένα σωστό link που να τελειώνει σε .mp3,
3. κρατήσει μια βασική οργάνωση (γιατί εύκολα χάνεται η μπάλα με πολλά αρχεία),
4. και να υπάρχει ένας έλεγχος/περιορισμός ώστε το σύστημα να μην γεμίσει ανεξέλεγκτα.

Από αυτή την ανάγκη προέκυψε η ιδέα για την πλατφόρμα της διπλωματικής: μια πλατφόρμα οργάνωσης και φιλοξενίας αρχείων ήχου ειδικά σχεδιασμένη για ενσωμάτωση σε Genially και παρόμοια διαδραστικά εργαλεία.

1.2 Περιγραφή προβλήματος

Το βασικό πρόβλημα συνοψίζεται ως εξής:

- Το Genially **δεν επιτρέπει upload mp3** στο ίδιο το έργο στην free έκδοση.
- Επιτρέπει όμως να προστεθεί ήχος μέσω **URL**.
- Το URL πρέπει να "μοιάζει" με άμεσο link σε αρχείο mp3 και να **τελειώνει σε .mp3**.
- Οι χρήστες (π.χ. φοιτητές) πρέπει να μπορούν να ανεβάζουν τα δικά τους αρχεία, αλλά με όρια (π.χ. αποθήκευση, μέγεθος, αριθμός).

Χωρίς μια τέτοια πλατφόρμα, η λύση συνήθως είναι χειροκίνητη και άναρχη: ανεβάσματα σε τυχαία sites, links που λήγουν, υπηρεσίες που δεν δίνουν direct mp3 URL, ή ακόμα και χρήση υπηρεσιών που βάζουν περιορισμούς/διαφημίσεις. Αυτό δεν είναι πρακτικό για ένα οργανωμένο μάθημα, ειδικά όταν υπάρχουν πολλοί χρήστες και πολλά projects.

Άρα, αυτό που χρειάζεται είναι μια εφαρμογή που:

- να λειτουργεί σαν "κεντρική αποθήκη" mp3,
- να δίνει άμεσα το link μετά το upload,
- να επιτρέπει βασική διαχείριση (rename, delete),
- να έχει ρόλους (user/admin),
- και να εφαρμόζει όρια (π.χ. 20 αρχεία ανά χρήστη, 10MB το καθένα).

1.3 Στόχος της εργασίας

Στόχος της παρούσας διπλωματικής είναι η σχεδίαση και υλοποίηση μιας διαδικτυακής πλατφόρμας που επιτρέπει την ανέβασμα (upload), οργάνωση και φιλοξενία αρχείων ήχου mp3, και τη δημιουργία ενός άμεσου public συνδέσμου (URL) που μπορεί να ενσωματωθεί σε εκπαιδευτικό διαδραστικό υλικό (Genially).

Πιο συγκεκριμένα, στόχοι της εφαρμογής είναι:

1. **Διαχείριση χρηστών με login**
    - Ο χρήστης να μπορεί να εισέρχεται στο σύστημα.
    - Ο admin να έχει πλήρη έλεγχο και εποπτεία.

2. **Upload αρχείων mp3**
    - Με περιορισμό μεγέθους 10MB ανά αρχείο.
    - Με περιορισμό πλήθους έως 20 αρχεία ανά χρήστη.
3. **Αυτόματη παραγωγή link**
    - Μετά το upload, ο χρήστης να παίρνει άμεσα το link για χρήση στο Genially.
    - Το link να καταλήγει σε .mp3.
4. **Βασικές λειτουργίες διαχείρισης αρχείων**
    - Rename (μετονομασία "εμφάνισης").
    - Delete (διαγραφή).
5. **Ενδεχόμενη υποστήριξη ηχογράφησης**
    - Να εξεταστεί η δυνατότητα ο χρήστης να ηχογραφεί από μικρόφωνο και να αποθηκεύεται το αρχείο, ώστε να μη χρειάζεται να έχει έτοιμο mp3 από πριν.
    - Στην υλοποίηση χρησιμοποιείται η λογική "browser recording" και μετατροπή σε mp3 στον server.
6. **Admin περιβάλλον διαχείρισης**
    - Σελίδες για προβολή χρηστών και αρχείων.
    - Στατιστικά (π.χ. αριθμός χρηστών, αριθμός αρχείων, συνολικός χώρος, uploads ανά ημέρα).
    - Ενέργειες (ενεργοποίηση/απενεργοποίηση χρήστη, διαγραφή αρχείων).

1.4 Τεχνολογίες και εργαλεία που χρησιμοποιήθηκαν

Η εφαρμογή βασίστηκε σε τεχνολογίες που είναι διαδεδομένες και αρκετά "ελαφριές" για web projects, ειδικά σε εκπαιδευτικό πλαίσιο:

- **Python**: κύρια γλώσσα ανάπτυξης [3].
- **Flask**: web framework για routing, sessions, endpoints και γενική δομή της εφαρμογής [4].
- **MySQL**: βάση δεδομένων για αποθήκευση χρηστών και μεταδεδομένων αρχείων [5].
- **Bootstrap**: γρήγορος και καθαρός σχεδιασμός (responsive) για το UI [6].
- **Chart.js**: για βασικά γραφήματα στα admin statistics [7].

- **ffmpeg**: για μετατροπή ηχογραφήσεων (webm) σε mp3 όταν γίνεται recording από browser [8].

Ο συνδυασμός αυτός επιλέχθηκε γιατί:

- επιτρέπει γρήγορη ανάπτυξη,
- είναι αρκετά ευέλικτος,
- καλύπτει τις βασικές ανάγκες (ασφάλεια, βάση, UI),
- και μπορεί να στηθεί και σε απλό server χωρίς "βαριά" υποδομή.

1.5 Βασική ιδέα λειτουργίας της πλατφόρμας

Η λειτουργία της πλατφόρμας μπορεί να περιγραφεί απλά με το παρακάτω σενάριο:

1. Ο χρήστης κάνει login στο σύστημα.
2. Από το dashboard ανεβάζει ένα mp3 αρχείο.
3. Το σύστημα αποθηκεύει το αρχείο στον server και το "βαφτίζει" με ένα μοναδικό UUID 5 χαρακτήρων (π.χ. A7xK2.mp3) [9].
4. Ο χρήστης βλέπει άμεσα το link: http://server/audio/A7xK2.mp3
5. Το link αυτό μπορεί να μπει στο Genially και να παίξει κανονικά σαν ενσωματωμένος ήχος.
6. Ο χρήστης μπορεί να αλλάξει το "όνομα εμφάνισης" του αρχείου (για καλύτερη οργάνωση) χωρίς να αλλάζει το URL.
7. Ο admin μπορεί να βλέπει όλους τους χρήστες και όλα τα αρχεία, και να ελέγχει τη χρήση του συστήματος.

Η επιλογή του "μικρού" UUID (5 χαρακτήρες) έγινε με σκοπό:

- το link να είναι μικρό και εύχρηστο,
- το filename να είναι σταθερό και να μην έχει περίεργους χαρακτήρες,
- και να αποφεύγονται συγκρούσεις/διπλά ονόματα.

1.6 Δομή της διπλωματικής εργασίας

Η εργασία οργανώνεται σε κεφάλαια ώστε να παρουσιάζεται με λογική σειρά:

- **Κεφάλαιο 1**: Εισαγωγή (το πρόβλημα, ο στόχος και η γενική ιδέα).
- **Κεφάλαιο 2**: Θεωρητικό υπόβαθρο (web εφαρμογές, βάσεις, ασφάλεια, media hosting).
- **Κεφάλαιο 3**: Ανάλυση απαιτήσεων (λειτουργικές/μη λειτουργικές απαιτήσεις, περιορισμοί).
- **Κεφάλαιο 4**: Σχεδίαση συστήματος (αρχιτεκτονική, ροές, endpoints, ρόλοι).
- **Κεφάλαιο 5**: Σχεδίαση βάσης δεδομένων (πίνακες και σχέσεις).
- **Κεφάλαιο 6**: Υλοποίηση (κύρια σημεία κώδικα και τεχνολογικές επιλογές).
- **Κεφάλαιο 7**: Δοκιμές και αξιολόγηση (test cases, edge cases, αποτελέσματα).
- **Κεφάλαιο 8**: Συμπεράσματα και μελλοντικές επεκτάσεις.

Κεφάλαιο 2ο: Θεωρητική προσέγγιση

2.1 Διαδικτυακές εφαρμογές και μοντέλο client–server

Μια διαδικτυακή εφαρμογή (web application) είναι ουσιαστικά ένα σύστημα που "τρέχει" σε δύο πλευρές:

- **Client (πελάτης):** συνήθως ο browser του χρήστη (Chrome, Firefox κτλ.). Εκεί εμφανίζεται το περιβάλλον, τα κουμπιά, οι φόρμες, οι πίνακες κ.λπ.
- **Server (εξυπηρετητής):** εκεί βρίσκεται η λογική της εφαρμογής. Ο server δέχεται αιτήματα (requests), τα επεξεργάζεται, μιλάει με τη βάση δεδομένων και επιστρέφει απαντήσεις (responses).

Η επικοινωνία γίνεται μέσω HTTP/HTTPS. Για παράδειγμα:

- Ο χρήστης ανοίγει το /dashboard → ο server επιστρέφει HTML [10].
- Ο χρήστης ανεβάζει ένα mp3 στο /upload → ο server αποθηκεύει το αρχείο, γράφει εγγραφή στη βάση και επιστρέφει αποτέλεσμα.
- Το Genially "τραβάει" το mp3 από /audio/ABCDE.mp3 → ο server πρέπει να του δώσει το αρχείο σαν "stream".

Στη δική μας διπλωματική, το client είναι κυρίως το UI που βλέπει ο χρήστης (Bootstrap templates) και ο server είναι η εφαρμογή Flask που χειρίζεται login, uploads, links, admin σελίδες κτλ.

2.2 Η έννοια του URL και γιατί μας ενδιαφέρει να τελειώνει σε .mp3

Το URL (Uniform Resource Locator) είναι η "διεύθυνση" ενός πόρου (resource) στο web. Όταν λέμε ότι το Genially δέχεται mp3 μόνο μέσω URL που τελειώνει σε .mp3, πρακτικά σημαίνει ότι:

- θέλει να βλέπει ένα link που μοιάζει με άμεση διαδρομή αρχείου ήχου,
- ώστε να το ζητάει απευθείας και να το αναπαράγει.

Πολλές υπηρεσίες φιλοξενίας δεν δίνουν "direct link" (π.χ. σου δίνουν link σε σελίδα με player ή κάνουν redirect). Εδώ όμως χρειαζόμαστε άμεσο endpoint που να σερβίρει το αρχείο ως mp3. Γι' αυτό φτιάχνουμε δικό μας endpoint τύπου:

/audio/<uuid>.mp3

που επιστρέφει το ίδιο το αρχείο, με σωστό Content-Type: audio/mpeg.

2.3 Flask: Τι είναι και γιατί επιλέχθηκε

Το Flask είναι ένα "ελαφρύ" web framework σε Python. Το βασικό του πλεονέκτημα είναι ότι:

- σου δίνει routing (π.χ. @app.get("/dashboard")),
- διαχείριση αιτημάτων (request),
- δυνατότητα να επιστρέφεις HTML, JSON, αρχεία,
- και γενικά μια καθαρή δομή για να χτίσεις web εφαρμογή.

Σε αντίθεση με "βαριά" frameworks, το Flask αφήνει τον προγραμματιστή να επιλέξει τα επιμέρους κομμάτια (π.χ. login system, ORM, forms).

Θέλουμε να έχουμε έλεγχο στη ροή upload/serving,

και να υλοποιήσουμε ακριβώς αυτά που χρειαζόμαστε χωρίς περιττή πολυπλοκότητα.

2.3.1 Routes και endpoints

Κάθε route είναι μια διεύθυνση στην εφαρμογή που κάνει κάτι. Για παράδειγμα:

- /login → εμφανίζει φόρμα login και κάνει authentication.
- /upload → δέχεται αρχείο και το αποθηκεύει.
- /audio/<uuid>.mp3 → σερβίρει δημόσια το mp3.
- /admin/... → σελίδες διαχείρισης.

Αυτό το κομμάτι είναι βασικό γιατί η εφαρμογή μας ουσιαστικά "στήνεται" γύρω από endpoints που εξυπηρετούν τον χρήστη και το Genially.

2.4 Templates και Bootstrap: γιατί μας βοηθάνε

Οι περισσότερες σελίδες της εφαρμογής είναι HTML που δημιουργείται από templates (Jinja2). Έτσι μπορούμε:

- να έχουμε ένα κοινό base.html (με navbar, μηνύματα, scripts),
- και κάθε σελίδα να "κληρονομεί" αυτό το layout.

Το Bootstrap χρησιμοποιείται γιατί:

- δίνει έτοιμα responsive components (cards, tables, forms),
- κάνει το UI αξιοπρεπές χωρίς να χρειάζεται να σχεδιάσουμε από το μηδέν,
- και βοηθάει πολύ σε admin σελίδες όπου υπάρχουν πίνακες, φίλτρα, pagination.

Με λίγα λόγια: μας γλιτώνει χρόνο και κρατάει ένα "καθαρό" στυλ.

2.5 Βάση δεδομένων MySQL: ρόλος και χρησιμότητα

Η εφαρμογή δεν είναι απλά "ανεβάζω ένα αρχείο και τέλος". Θέλουμε:

- χρήστες,
- ρόλους,
- όρια ανά χρήστη,
- ιστορικό uploads,
- στατιστικά για admin.

Άρα χρειαζόμαστε βάση δεδομένων. Επιλέχθηκε η MySQL γιατί είναι:

- πολύ διαδεδομένη,
- αξιόπιστη,
- εύκολη στη φιλοξενία (shared/VM),
- και ταιριάζει σε εφαρμογές με δομημένα δεδομένα.

2.5.1 Τι αποθηκεύουμε στη βάση

Στην εφαρμογή υπάρχουν (λογικά) δύο βασικές οντότητες:

1. **Users (χρήστες)**
    - email (ή NULL για pending accounts)
    - password hash
    - ρόλος (user/admin)
    - κατάσταση ενεργού/ανενεργού
    - στοιχεία όπως created_at, last_login
    - (προαιρετικά/όπως ζητήθηκε) plain_code για να φαίνεται ο κωδικός

2. **AudioFiles (αρχεία ήχου)**
    - uuid (το "δημόσιο" όνομα του mp3)
    - display name (όνομα εμφάνισης)
    - original name (όνομα που ανέβασε ο χρήστης)
    - μέγεθος, τύπος
    - storage path (πού βρίσκεται στο δίσκο)
    - ημερομηνία upload
    - user_id για σύνδεση με τον χρήστη

Με αυτό τον τρόπο, ακόμα κι αν στο δίσκο το αρχείο λέγεται AB12c.mp3, εμείς στη βάση κρατάμε και την πληροφορία για να το διαχειρίζεται ο χρήστης.

2.6 Authentication και ρόλοι (User/Admin)

Σε μια τέτοια πλατφόρμα δεν γίνεται να είναι όλα δημόσια. Χρειαζόμαστε:

- **Login**: να μπαίνει ο χρήστης με email/κωδικό.
- **Role-based access**: ο admin να βλέπει τα πάντα, ο χρήστης μόνο τα δικά του.

Χρησιμοποιείται Flask-Login για να υπάρχει:

- session (να "θυμάται" ότι ο χρήστης είναι logged in),
- εύκολη πρόσβαση στο current_user.

2.6.1 "Προκαθορισμένοι" λογαριασμοί με κωδικό (claim flow)

Στο συγκεκριμένο project υπάρχει μια ιδιαίτερη λογική εγγραφής:

- Ο admin δημιουργεί χρήστες **χωρίς email** (pending).
- Δίνει στον χρήστη έναν κωδικό.
- Ο χρήστης βάζει **το email του + τον κωδικό** και έτσι "δεσμεύει" τον λογαριασμό.

Αυτό μοιάζει με σύστημα "invite codes". Είναι πρακτικό σε εκπαιδευτικό περιβάλλον γιατί:

- δεν υπάρχουν ανοικτές εγγραφές,

- ο admin ελέγχει ποιος θα έχει πρόσβαση,
- και ο χρήστης απλά συμπληρώνει email χωρίς να περιμένει approval.

2.7 Αποθήκευση αρχείων (File Uploads) και περιορισμοί

Η μεταφόρτωση αρχείων είναι σημείο που θέλει προσοχή, γιατί μπορεί να δημιουργήσει προβλήματα (χώρος, ασφάλεια, κακόβουλα αρχεία).

Στην εφαρμογή ισχύουν συγκεκριμένοι περιορισμοί:

- **μόνο mp3**
- **μέγιστο 10MB ανά αρχείο**
- **μέχρι 20 αρχεία ανά χρήστη**

Αυτά υλοποιούνται με:

- έλεγχο επέκτασης/τύπου,
- έλεγχο μεγέθους request,
- και έλεγχο πλήθους εγγραφών στη βάση.

Επιπλέον, αποθηκεύουμε τα αρχεία σε δομή φακέλων ανά χρήστη (π.χ. uploads/u12/AB12c.mp3) ώστε:

- να μην μπερδεύονται τα αρχεία,
- να είναι πιο εύκολη η διαχείριση στο filesystem.

2.8 UUID και “σταθερό” public link

Η βασική ιδέα είναι ότι ο χρήστης μπορεί να κάνει rename το αρχείο, αλλά το URL δεν πρέπει να αλλάζει, γιατί μπορεί να το έχει ήδη βάλει στο Genially.

Γι’ αυτό:

- το πραγματικό filename που σερβίρεται δημόσια είναι uuid.mp3
- ενώ το “rename” αφορά μόνο το display_name στη βάση.

Έτσι:

- ο χρήστης οργανώνει τα αρχεία του με ονόματα που καταλαβαίνει,
- αλλά το Genially link παραμένει το ίδιο και δεν “σπάει”.

2.9 Serving mp3 και έννοια του streaming

Όταν ένα αρχείο ήχου παίζει online, ο browser (ή μια πλατφόρμα όπως το Genially) συχνά δεν το κατεβάζει όλο από την αρχή. Αντίθετα ζητάει τμήματα του αρχείου, ειδικά όταν:

- γίνεται seek (π.χ. πας στη μέση),
- το αρχείο είναι μεγάλο,
- ή ο player θέλει buffering.

Εδώ μπαίνει το HTTP Range header (Range: bytes=...). Αν ο server το υποστηρίζει σωστά:

- επιστρέφει status 206 Partial Content
- και στέλνει μόνο το κομμάτι που ζητήθηκε.

Αυτό αυξάνει τη συμβατότητα και κάνει το playback πιο “σωστό” σε διάφορους players. Για αυτό στην εφαρμογή υλοποιείται byte-range υποστήριξη στο endpoint /audio/<uuid>.mp3.

2.10 Βασικά θέματα ασφάλειας

2.10.1 Hashing κωδικών

Αποθηκεύεται hash.
Στο project, υπάρχει password_hash που χρησιμοποιείται για login.

Στη συγκεκριμένη υλοποίηση κρατάμε και plain_code επειδή το ζητά η λειτουργία admin/εκπαιδευτικό σενάριο, αλλά το σωστό από άποψη ασφάλειας είναι να μην υπάρχει.

2.10.2 CSRF protection

Το CSRF (Cross-Site Request Forgery) είναι επίθεση όπου ένας κακόβουλος μπορεί να προκαλέσει POST request από τον browser σου χωρίς να το καταλάβεις.

Γι’ αυτό χρησιμοποιείται CSRF token σε κάθε POST form. Αν λείπει, ο server απορρίπτει το αίτημα. Αυτό το συναντήσαμε και πρακτικά κατά την ανάπτυξη, όταν έλειπε το token από τις φόρμες.

2.10.3 Έλεγχοι upload

Αν αφήσεις upload ανεξέλεγκτο, μπορεί κάποιος να ανεβάσει:

- τεράστια αρχεία,
- μη επιτρεπτούς τύπους,
- ή να γεμίσει το σύστημα.

Γι' αυτό υπάρχουν όρια μεγέθους και πλήθους, και επιτρέπεται μόνο .mp3.

2.10.4 Πρόσβαση admin endpoints

Ό,τι είναι /admin/... πρέπει να προστατεύεται όχι μόνο με login αλλά και με role check. Διαφορετικά ένας απλός χρήστης μπορεί να δει στατιστικά ή να διαγράψει αρχεία άλλων.

2.11 Ηχογράφηση από μικρόφωνο και μετατροπή σε mp3

Ένα χρήσιμο "bonus" στο σύστημα είναι η δυνατότητα να γράφει ο χρήστης ήχο απευθείας από τον browser.

Στον browser αυτό γίνεται με MediaRecorder API. Συνήθως το αποτέλεσμα είναι μορφή webm (ή άλλη που υποστηρίζει ο browser). Επειδή όμως εμείς θέλουμε mp3 για Genially:

- ο client στέλνει το ηχογραφημένο blob στον server,
- ο server το μετατρέπει σε mp3 με ffmpeg,
- επιστρέφει το public link.

Αυτό κάνει τη διαδικασία πιο εύκολη για τον χρήστη και δεν χρειάζεται να έχει έτοιμο mp3 από πριν, μπορεί να δημιουργήσει το υλικό μέσα από την πλατφόρμα.

Κεφάλαιο 3ο: Προδιαγραφές και στόχοι λειτουργίας του συστήματος

3.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται πιο "συγκεκριμένα" τι πρέπει να προσφέρει η πλατφόρμα και με ποιον τρόπο αναμένεται να λειτουργεί. Με απλά λόγια, εδώ καταγράφονται οι βασικές λειτουργίες που χρειάζεται ο χρήστης, οι περιορισμοί που πρέπει και τα βασικά σενάρια χρήσης για φοιτητές και διαχειριστή.

Ο στόχος είναι να υπάρχει μια καθαρή εικόνα:

- τι κάνει ο χρήστης μέσα στην πλατφόρμα,
- τι μπορεί να κάνει ο admin,
- τι πρέπει να "βγαίνει" προς τα έξω για το Genially (δηλαδή links που τελειώνουν σε .mp3),
- ποια όρια κρατούν τη χρήση σε λογικά πλαίσια.

3.2 Τι προσφέρει η πλατφόρμα στον χρήστη

3.2.1 Είσοδος στο σύστημα και dashboard

Ο χρήστης πρέπει να μπορεί να μπαίνει στο σύστημα μέσω σελίδας σύνδεσης, εισάγοντας:

- το email του
- και έναν κωδικό πρόσβασης

Η σύνδεση είναι απαραίτητη, γιατί τα αρχεία είναι οργανωμένα ανά χρήστη και θέλουμε κάθε χρήστης να βλέπει μόνο τα δικά του αρχεία.

3.2.2 Ενεργοποίηση λογαριασμού με κωδικό (λογική "claim")

Επειδή το σύστημα προορίζεται για εκπαιδευτικό περιβάλλον, δεν υπάρχει "ελεύθερη εγγραφή". Αντί γι' αυτό:

- Ο admin δημιουργεί λογαριασμούς-θέσεις (pending) χωρίς email.
- Κάθε pending λογαριασμός έχει έναν κωδικό.

- Ο χρήστης, την πρώτη φορά που θα μπει, δίνει το email του + τον κωδικό που του έχει δοθεί.
- Το email "γράφεται" πάνω στον pending λογαριασμό και ο χρήστης αποκτά μόνιμη πρόσβαση.

Έτσι ο admin έχει πλήρη έλεγχο στο ποιος θα μπει στο σύστημα, αλλά η διαδικασία είναι εύκολη για τον χρήστη (δεν περιμένει approval, απλά "ενεργοποιείται").

3.2.3 Μεταφόρτωση mp3 και άμεσο link

Ο χρήστης πρέπει να μπορεί να ανεβάζει ένα mp3 αρχείο και αμέσως μετά:

- να εμφανίζεται το αρχείο στη λίστα του
- και να μπορεί να αντιγράψει το link του

Το link πρέπει να είναι τύπου:

/audio/<uuid>.mp3
ώστε να γίνεται αποδεκτό από το Genially.

3.2.4 Μετονομασία "εμφάνισης" (rename χωρίς αλλαγή URL)

Ο χρήστης πρέπει να μπορεί να οργανώνει τα αρχεία του, αλλά χωρίς να χαλάει links που έχει ήδη βάλει στο Genially. Για αυτό:

- το rename αλλάζει μόνο το "όνομα εμφάνισης" (display name)
- το public URL μένει σταθερό και βασίζεται στο uuid

3.2.5 Διαγραφή αρχείου

Ο χρήστης πρέπει να μπορεί να διαγράψει ένα αρχείο όταν δεν το χρειάζεται άλλο. Η διαγραφή πρέπει να:

- αφαιρεί την εγγραφή από τη βάση
- και να αφαιρεί και το mp3 από τον δίσκο (storage)

3.2.6 Αλλαγή κωδικού σε ξεχωριστή σελίδα

Ο χρήστης πρέπει να μπορεί να αλλάξει τον κωδικό του (π.χ. μετά την πρώτη είσοδο). Η αλλαγή γίνεται σε ξεχωριστή σελίδα, π.χ.:

/account/password

Για την αλλαγή απαιτείται:

- ο τρέχων κωδικός (για επιβεβαίωση)
- ο νέος κωδικός (2 φορές)

3.3 Περιορισμοί χρήσης (ώστε να μην ξεφύγει το σύστημα)

Για να μην γεμίσει ο server και για να υπάρχει ισορροπία, ορίστηκαν συγκεκριμένα όρια:

Όριο μεγέθους αρχείου

- Μέγιστο μέγεθος ανά αρχείο: **10 MB**

Αν ένα αρχείο είναι μεγαλύτερο, η πλατφόρμα πρέπει να το απορρίπτει.

Όριο αριθμού αρχείων ανά χρήστη

- Μέγιστος αριθμός αρχείων ανά χρήστη: **20**

Αν ο χρήστης φτάσει τα 20 αρχεία, δεν πρέπει να μπορεί να ανεβάσει άλλο μέχρι να διαγράψει κάποιο.

Τύπος αρχείου

- Επιτρέπονται μόνο mp3 αρχεία (.mp3).

Αυτό βοηθάει:

- στη συμβατότητα με Genially,
- και στο να αποφεύγονται “άσχετα” uploads.

3.4 Λειτουργίες διαχειριστή (admin) και έλεγχος συστήματος

Ο admin είναι ο ρόλος που επιβλέπει τη λειτουργία και έχει πρόσβαση σε στατιστικά και εργαλεία διαχείρισης.

Σελίδες admin (διακριτές)

Ο admin έχει ξεχωριστές σελίδες, όπως:

- /admin (συνολικά στατιστικά + γραφήματα)
- /admin/users (λίστα χρηστών, φίλτρα, ενέργειες)

- /admin/files (λίστα αρχείων όλων των χρηστών, φίλτρα, διαγραφές)

Δημιουργία χρηστών "με κωδικό μόνο"

Ο admin πρέπει να μπορεί από τη σελίδα χρηστών να δημιουργεί pending χρήστες:

- με άδειο email
- και με συγκεκριμένο ή αυτόματα παραγόμενο κωδικό

Ο κωδικός πρέπει να φαίνεται:

- στον admin πίνακα
- και στη βάση (στο πεδίο plain_code), όπως ζητήθηκε.

Ενεργοποίηση/απενεργοποίηση χρήστη

Ο admin πρέπει να μπορεί να αλλάζει την κατάσταση ενός χρήστη (active/inactive). Αν ένας χρήστης είναι inactive, δεν πρέπει να μπορεί να κάνει login.

Διαχείριση αρχείων όλων των χρηστών

Ο admin πρέπει να μπορεί να βλέπει και να διαγράφει αρχεία, σε περίπτωση:

- κατάχρησης,
- λάθους upload,
- ή ανάγκης καθαρισμού χώρου.

Στατιστικά λειτουργίας

Ενδεικτικά, ο admin πρέπει να βλέπει:

- πλήθος χρηστών
- πλήθος αρχείων
- συνολικό χώρο αποθήκευσης
- uploads ανά ημέρα (π.χ. τελευταίες 30 ημέρες)
- top χρήστες ανά χρήση χώρου
- πρόσφατα uploads

Αυτό βοηθάει στην παρακολούθηση και στη “υγεία” του συστήματος.

3.5 Συμβατότητα με Genially

Η πλατφόρμα πρέπει να παράγει link που:

- είναι δημόσια προσβάσιμο
- τελειώνει σε .mp3
- σερβίρει το αρχείο με σωστό Content-Type

Επιπλέον, επειδή αρκετοί players κάνουν seek/buffering, το endpoint πρέπει να υποστηρίζει byte-range (Partial Content). Αυτό αυξάνει την πιθανότητα να δουλέψει σωστά σε διαφορετικούς browsers και environments.

3.6 Σενάρια χρήσης

3.6.1 Σενάριο χρήστη: upload και χρήση στο Genially

1. Ο χρήστης κάνει login.
2. Ανεβάζει mp3.
3. Παίρνει το link /audio/<uuid>.mp3.
4. Το βάζει στο Genially.
5. Ο ήχος παίζει μέσα στο διαδραστικό υλικό.

3.6.2 Σενάριο χρήστη: rename χωρίς να σπάσει το link

1. Ο χρήστης βλέπει ότι το όνομα δεν τον βολεύει.
2. Κάνει rename.
3. Το link μένει ίδιο → δεν χρειάζεται αλλαγή στο Genially.

3.6.3 Σενάριο διαχειριστή: δημιουργία πρόσβασης σε φοιτητή

1. Ο admin δημιουργεί pending λογαριασμό (χωρίς email) με έναν κωδικό.
2. Δίνει τον κωδικό στον φοιτητή.
3. Ο φοιτητής βάζει email + κωδικό → ενεργοποιείται ο λογαριασμός.

Κεφάλαιο 4ο: Σχεδίαση και αρχιτεκτονική συστήματος

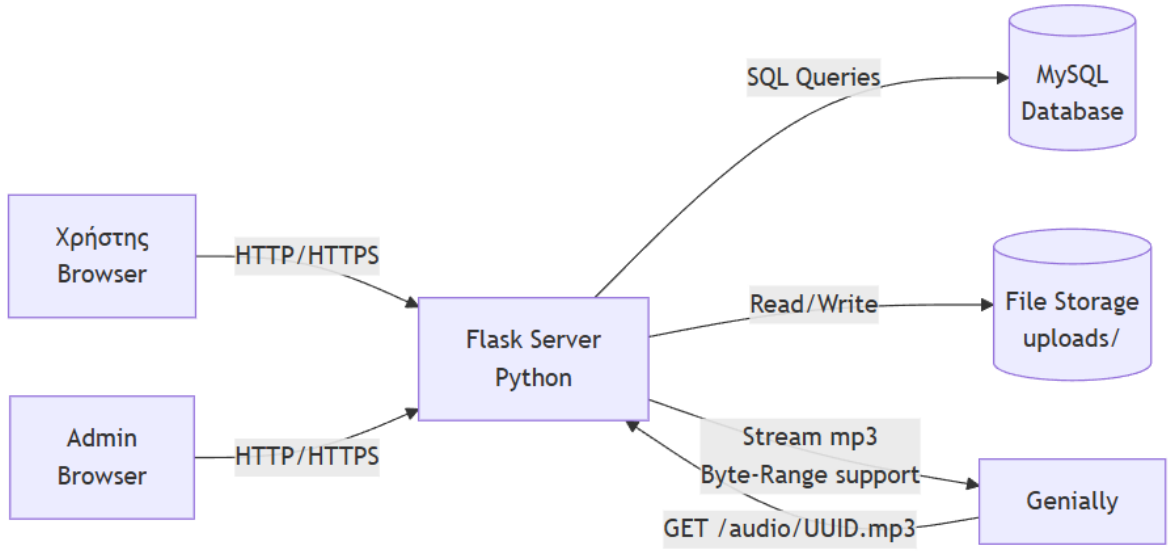
4.1 Εισαγωγή

Εδώ παρουσιάζεται ο τρόπος με τον οποίο στήθηκε η πλατφόρμα σε επίπεδο σχεδίασης. Δηλαδή ποια είναι τα βασικά υποσυστήματα, πώς επικοινωνούν μεταξύ τους, ποιες είναι οι κύριες ροές (login/claim, upload, δημόσιο mp3 link, admin διαχείριση) και πώς οργανώνονται τα endpoints. Ο στόχος είναι να φαίνεται καθαρά ότι το σύστημα δεν είναι απλά "μια σελίδα upload", αλλά μια ολοκληρωμένη εφαρμογή με ρόλους, περιορισμούς και συγκεκριμένο τρόπο εξυπηρέτησης αρχείων ήχου προς το Genially.

4.2 Αρχιτεκτονική υψηλού επιπέδου

Η εφαρμογή ακολουθεί το κλασικό μοντέλο web εφαρμογής:

- ο χρήστης αλληλεπιδρά με το UI από τον browser,
- ο Flask server χειρίζεται τη λογική,
- η MySQL βάση κρατάει χρήστες και μεταδεδομένα,
- τα mp3 αποθηκεύονται στον δίσκο του server (file storage),
- και το Genially λειτουργεί ως "εξωτερικός πελάτης" που ζητάει το mp3 μέσω public URL.



Εικόνα 4.1: Αρχιτεκτονική του συστήματος (Client–Server–DB–Storage–Genially)-Αρχιτεκτονική υψηλού επιπέδου της πλατφόρμας

Το διάγραμμα δείχνει τα βασικά μέρη του συστήματος και την ανταλλαγή δεδομένων: ο χρήστης και ο admin επικοινωνούν με τον Flask server, ο server χρησιμοποιεί MySQL για δεδομένα χρηστών/αρχείων και αποθηκεύει τα mp3 στο storage. Το Genially ζητά τα mp3 απευθείας από το public endpoint /audio/<uuid>.mp3.

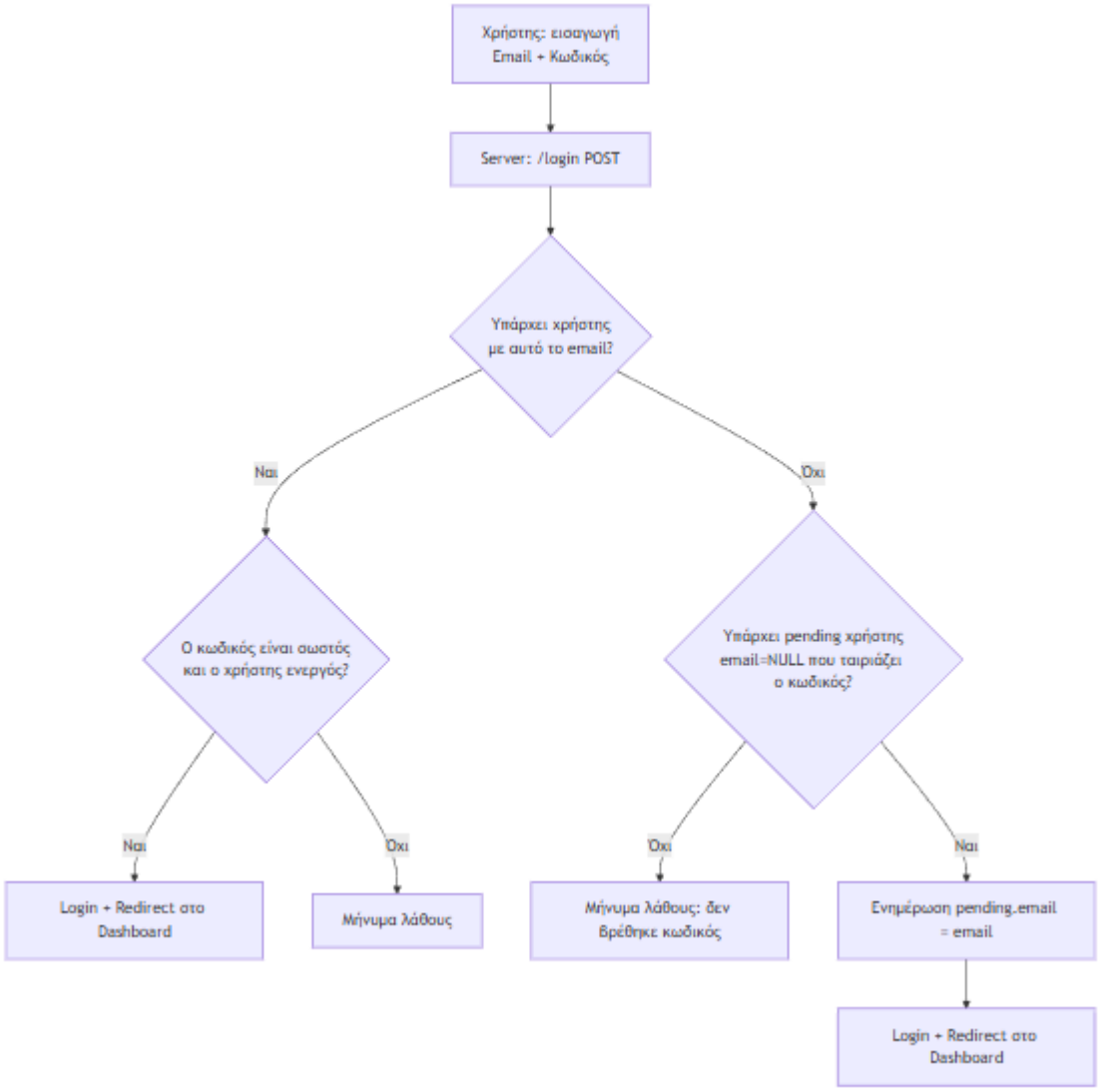
4.3 Εσωτερική οργάνωση

Σε επίπεδο λογικής, η εφαρμογή χωρίζεται σε μερικά "κομμάτια" που συνεργάζονται:

1. **Authentication / Session Layer**
    - Login, logout, session διαχείριση
    - Role check (admin vs user)
    - Claim flow για pending λογαριασμούς
2. **Audio File Management Layer**
    - Upload mp3 με έλεγχο ορίων
    - Rename (μόνο display name)
    - Delete (DB + filesystem)
    - Δημόσιο serving μέσω /audio/<uuid>.mp3
3. **Admin Management Layer**
    - Λίστα χρηστών (search/filter/pagination)
    - Δημιουργία pending χρηστών με κωδικό
    - Διαχείριση αρχείων όλων των χρηστών
    - Στατιστικά + γραφήματα
4. **Storage / Streaming Layer**
    - Αποθήκευση αρχείων ανά χρήστη
    - Byte-range responses για smooth playback
    - (Option 1) Μετατροπή recording webm → mp3 με ffmpeg

4.4 Σχεδίαση ροής "Claim Account" (email + κωδικός)

Αυτή είναι ίσως η πιο ιδιαίτερη λειτουργία του project, γιατί δεν έχουμε "register" όπως σε κλασικά sites. Ο admin δημιουργεί χρήστες χωρίς email και ο χρήστης συμπληρώνει το email του στην πρώτη είσοδο.

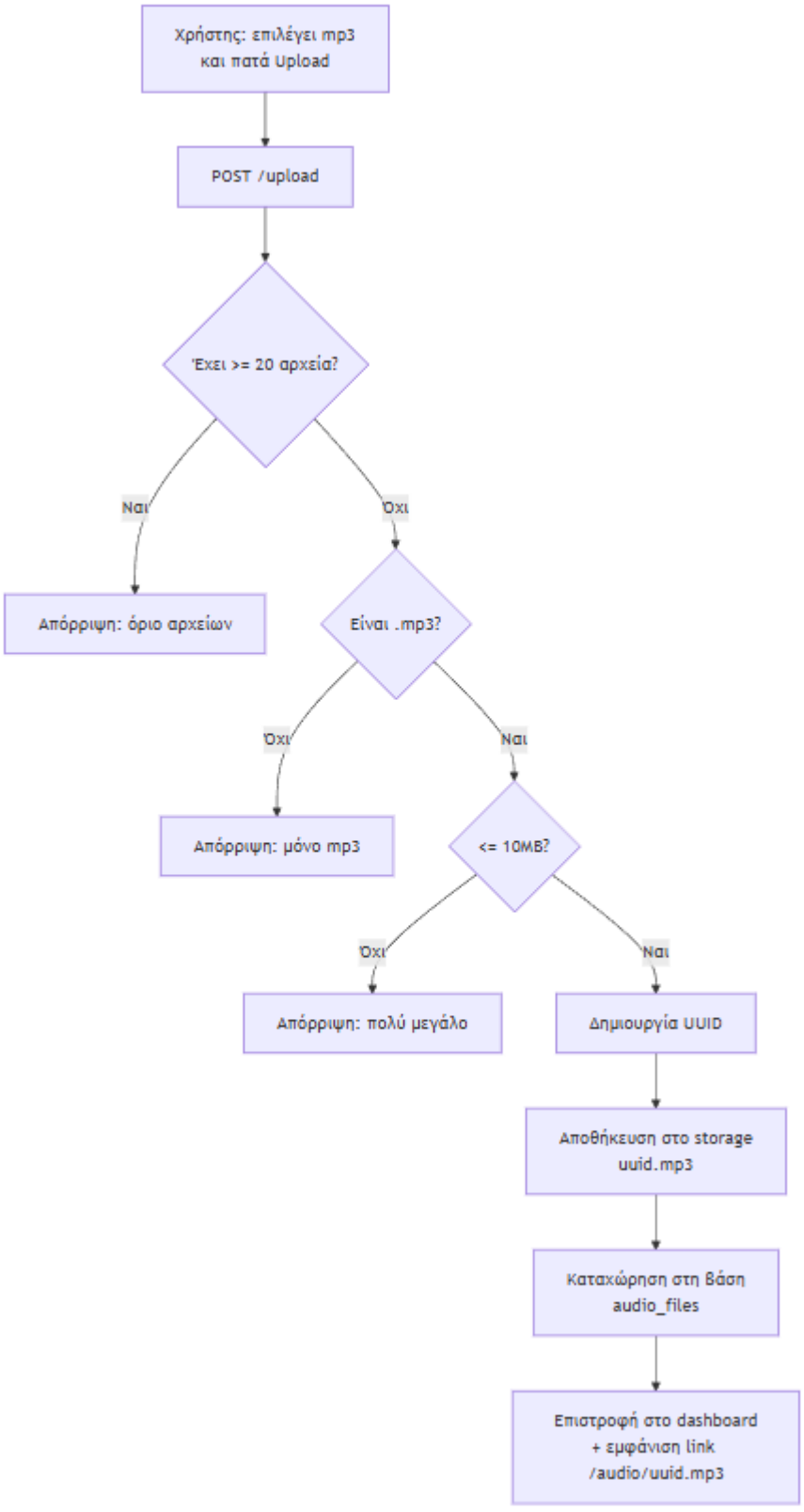


Εικόνα 4.2: Ροή claim λογαριασμού με email και κωδικό

Αν το email υπάρχει ήδη στη βάση, γίνεται κανονικό login. Αν δεν υπάρχει, το σύστημα ψάχνει pending χρήστη (email = NULL) που ταιριάζει ο κωδικός. Αν βρεθεί, ενημερώνει το email και ο χρήστης αποκτά πρόσβαση.

4.5 Σχεδίαση ροής upload και παραγωγής public mp3 link

Η λειτουργία upload πρέπει να είναι απλή για τον χρήστη, αλλά στο παρασκήνιο να γίνονται έλεγχοι (όρια/τύπος/μέγεθος) και να δημιουργείται το public URL με uuid.



Εικόνα 4.3: Ροή upload mp3 και παραγωγής public link

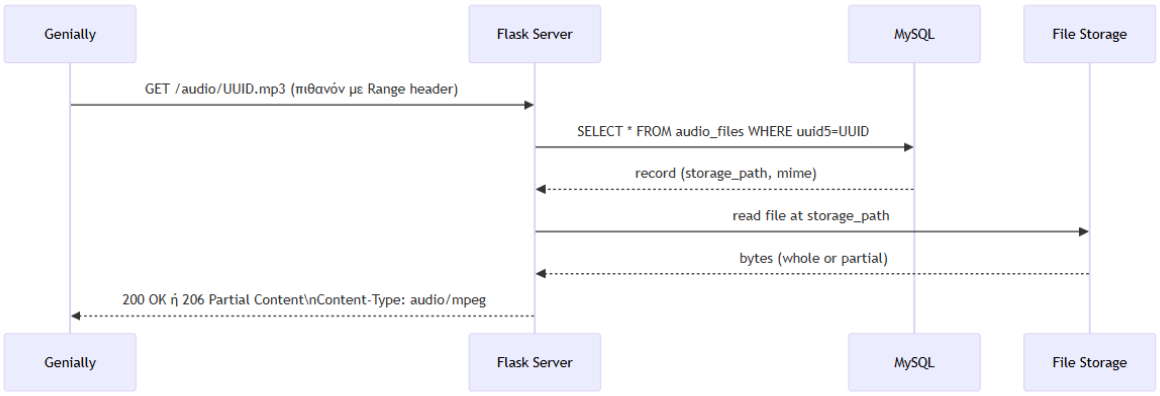
Το σύστημα ελέγχει αν ο χρήστης έχει ξεπεράσει το όριο των 20 αρχείων, αν το αρχείο είναι mp3 και

<=10MB, δημιουργεί μοναδικό uuid, αποθηκεύει το αρχείο στο filesystem και την εγγραφή στη βάση και επιστρέφει στον χρήστη το URL που θα χρησιμοποιήσει στο Genially.

4.6 Σχεδίαση δημόσιου endpoint /audio/<uuid>.mp3 (για Genially)

Το πιο κρίσιμο σημείο για να λειτουργήσει το Genially είναι να μπορεί να ζητήσει το mp3 σαν "κανονικό αρχείο". Αυτό σημαίνει:

- Να επιστρέφεται Content-Type: audio/mpeg
- Να υποστηρίζονται Range requests (Partial Content) για buffering/seek
- Να μην απαιτείται login (public πρόσβαση), αλλά να είναι "unlisted" μέσω uuid



Εικόνα 4.4: Ροή upload mp3 και παραγωγής public link - Ροή εξυπηρέτησης mp3 προς Genially

Το Genially στέλνει GET στο /audio/<uuid>.mp3. Ο server βρίσκει την εγγραφή στη βάση, εντοπίζει το αρχείο στο filesystem και επιστρέφει το αρχείο είτε ολόκληρο είτε τμηματικά (byte-range), επιτρέποντας ομαλή αναπαραγωγή.

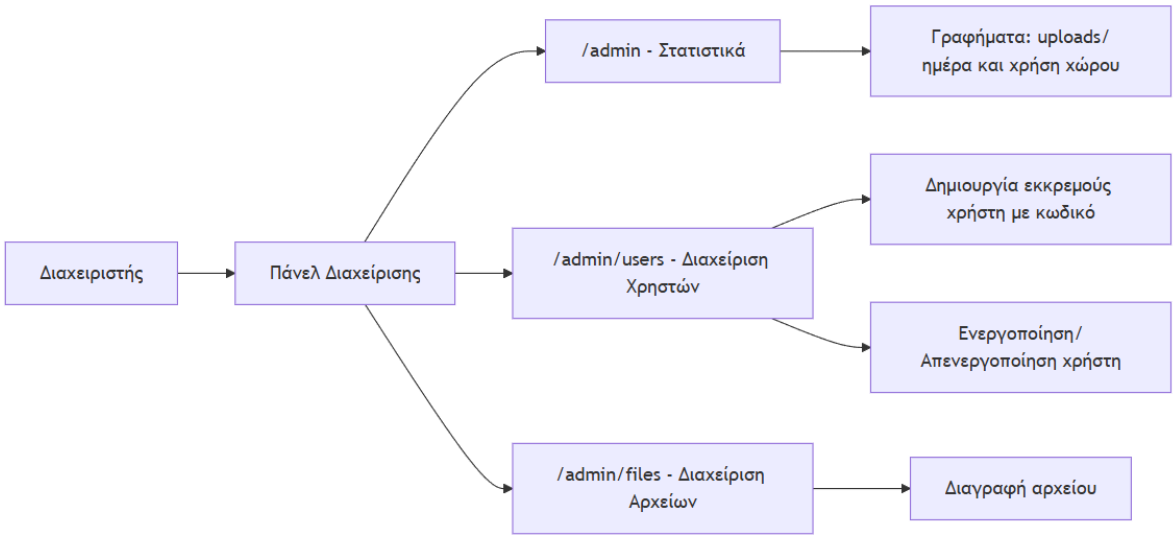
4.7 Σχεδίαση admin υποσυστήματος (σε ξεχωριστές σελίδες)

Ο admin έχει ξεχωριστό "κομμάτι" εφαρμογής, ώστε:

- να μην μπερδεύεται ο απλός χρήστης,
- να προστατεύονται οι διαδρομές με role check,
- και να υπάρχει πιο οργανωμένη διαχείριση.

Οι βασικές admin σελίδες είναι:

- /admin → συνοπτικά στατιστικά και γραφήματα
- /admin/users → χρήστες (pending/ενεργοί), κωδικοί, ενεργοποίηση/απενεργοποίηση
- /admin/files → όλα τα αρχεία, φίλτρα, διαγραφή



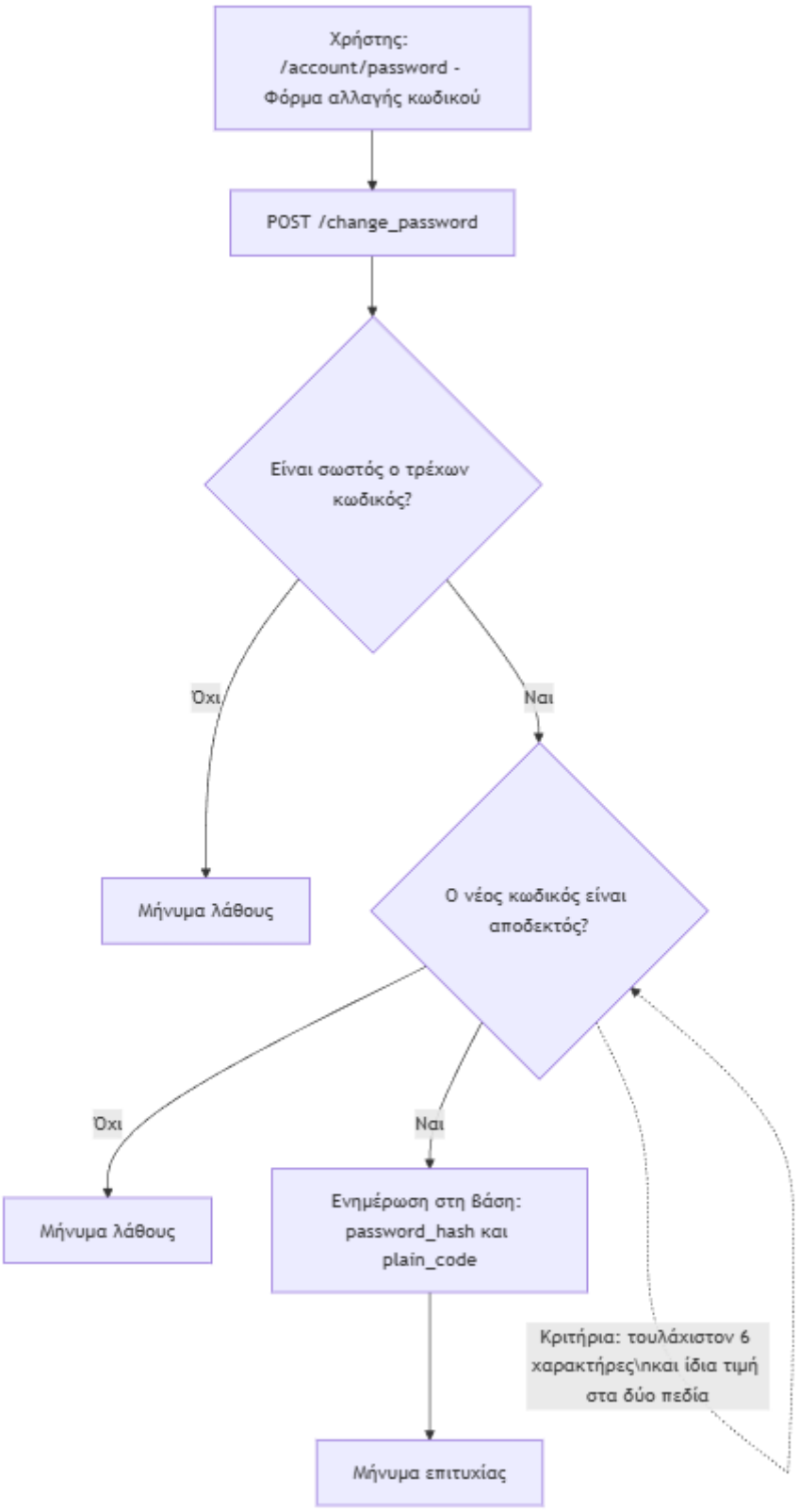
Εικόνα 4.5: Πλοήγηση και ενότητες του admin περιβάλλοντος

Το admin περιβάλλον έχει τρεις βασικές ενότητες. Όλες προστατεύονται με έλεγχο ρόλου (admin_required). Από κάθε ενότητα ο admin μπορεί να εκτελεί βασικές ενέργειες.

4.8 Σχεδίαση σελίδας αλλαγής κωδικού (ξεχωριστή σελίδα)

Η αλλαγή κωδικού δεν τοποθετείται στο dashboard για να μην "φορτώνει" την κύρια σελίδα του χρήστη. Για αυτό προβλέπεται μια απλή σελίδα:

- /account/password (GET → εμφανίζει φόρμα)
- /change_password (POST → εκτελεί αλλαγή)



Εικόνα 4.6: Ροή αλλαγής κωδικού χρήστη

Ο χρήστης εισάγει τον τρέχοντα κωδικό και τον νέο κωδικό δύο φορές. Ο server ελέγχει ότι ο τρέχων είναι σωστός, ότι ο νέος είναι έγκυρος και ενημερώνει τη βάση.

Κεφάλαιο 5ο: Σχεδίαση βάσης δεδομένων και μοντέλο δεδομένων

5.1 Εισαγωγή

Σε αυτή την πλατφόρμα η βάση δεδομένων είναι απαραίτητη γιατί δεν αρκεί απλά να αποθηκεύονται mp3 αρχεία στον δίσκο. Χρειαζόμαστε οργανωμένη πληροφορία για:

- τους χρήστες (ρόλος, κατάσταση ενεργού/ανενεργού, email ή pending),
- τα αρχεία (uuid, ονόματα, μέγεθος, ημερομηνία upload),
- τη σύνδεση "ποιος χρήστης ανέβασε ποιο αρχείο".

5.2 Οντότητες και πίνακες

Το σύστημα στηρίζεται σε δύο βασικούς πίνακες:

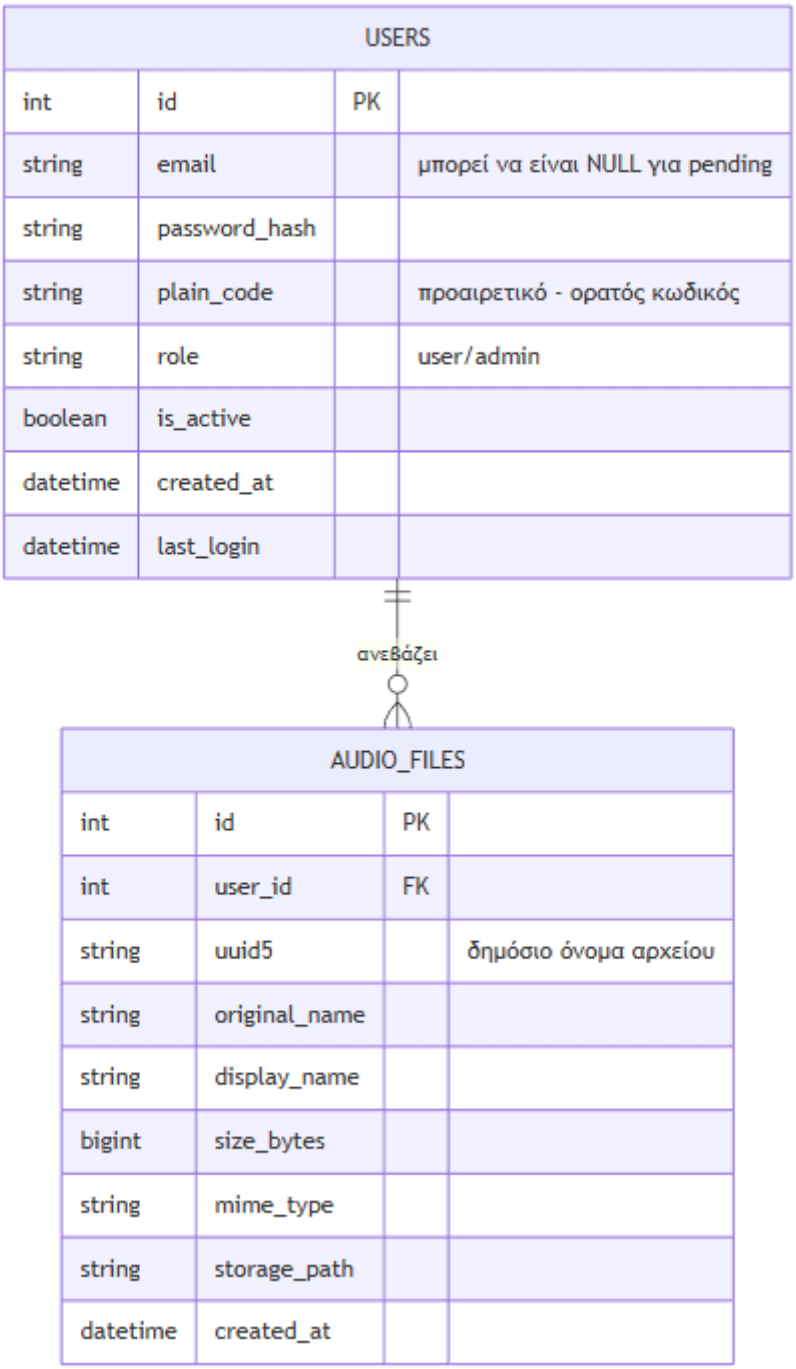
1. **users**: αποθηκεύει τους χρήστες της πλατφόρμας (φοιτητές/χρήστες και admin).
2. **audio_files**: αποθηκεύει τα αρχεία ήχου (μεταδεδομένα) και τη σύνδεσή τους με χρήστη.

Η σχέση είναι:

- Ένας χρήστης μπορεί να έχει πολλά αρχεία
- Κάθε αρχείο ανήκει σε έναν χρήστη

5.3 ER διάγραμμα (Entity–Relationship)

Παρακάτω φαίνεται το ER διάγραμμα με τις βασικές οντότητες και τα πεδία τους.



Εικόνα 5.1: ER διάγραμμα της βάσης δεδομένων (Users–AudioFiles)

Το διάγραμμα δείχνει τους δύο πίνακες (users και audio_files) και τη σχέση 1-προς-πολλά: ένας χρήστης μπορεί να ανεβάσει πολλά αρχεία, ενώ κάθε αρχείο συνδέεται με έναν χρήστη μέσω του user_id.

5.4 Πίνακας users (περιγραφή πεδίων)

Ο πίνακας users κρατάει την πληροφορία ταυτότητας και πρόσβασης.

- id: Μοναδικό αναγνωριστικό χρήστη (primary key).
- email: Email χρήστη. Μπορεί να είναι NULL όταν ο λογαριασμός είναι pending (δημιουργημένος από admin, αλλά δεν έχει "δεθεί" ακόμα με email).
- password_hash: Hash του κωδικού πρόσβασης (χρησιμοποιείται για έλεγχο login).
- plain_code: Προαιρετικό πεδίο που κρατάει τον κωδικό σε απλή μορφή για προβολή από admin (αυτό έγινε με βάση το συγκεκριμένο σενάριο χρήσης).
- role: Ο ρόλος του χρήστη (user ή admin).
- is_active: Δείχνει αν ο χρήστης είναι ενεργός. Αν είναι false, δεν επιτρέπεται login.
- created_at: Ημερομηνία δημιουργίας λογαριασμού.
- last_login: Τελευταία επιτυχημένη είσοδος.

Σημείωση για το email = NULL

Το ότι το email μπορεί να είναι NULL είναι κομβικό για το "claim flow":

- ο admin δημιουργεί χρήστες χωρίς email,
- ο φοιτητής μπαίνει με email + κωδικό και τότε συμπληρώνεται το email στη βάση.

5.5 Πίνακας audio_files (περιγραφή πεδίων)

Ο πίνακας audio_files περιγράφει κάθε αρχείο ήχου που ανέβηκε.

- id: Μοναδικό αναγνωριστικό αρχείου (primary key).
- user_id: Αναφορά στο users.id για το ποιος ανέβασε το αρχείο.
- uuid5: Το βασικό δημόσιο αναγνωριστικό του αρχείου. Είναι το όνομα που χρησιμοποιείται στο URL και στο filesystem: uuid5.mp3.
- original_name: Το αρχικό όνομα του αρχείου που ανέβασε ο χρήστης.
- display_name: Το όνομα που βλέπει ο χρήστης στο dashboard (μπορεί να αλλάζει με rename).
- size_bytes: Μέγεθος αρχείου σε bytes (χρήσιμο για στατιστικά και όρια).
- mime_type: Ο τύπος περιεχομένου (π.χ. audio/mpeg).
- storage_path: Πλήρης διαδρομή στο δίσκο του server (πού αποθηκεύτηκε).
- created_at: Ημερομηνία/ώρα upload.

5.6 Κανόνες και περιορισμοί σε επίπεδο δεδομένων

Μοναδικότητα email

Το email είναι μοναδικό όταν υπάρχει (δηλαδή δεν πρέπει να υπάρχουν δύο χρήστες με το ίδιο email). Όμως επειδή επιτρέπεται NULL, μπορούν να υπάρχουν πολλοί pending χρήστες με NULL email.

Μοναδικότητα uuid5

Το uuid5 πρέπει να είναι μοναδικό ανά αρχείο ώστε να μη δημιουργούνται συγκρούσεις στα public URLs.

Όρια (20 αρχεία και 10MB)

Τα όρια αυτά εφαρμόζονται κυρίως σε επίπεδο εφαρμογής (Flask), αλλά η βάση δίνει τα απαραίτητα στοιχεία:

- με count των αρχείων ανά χρήστη,
- με sum του μεγέθους ανά χρήστη,
- και με αποθήκευση size_bytes.

Για να λειτουργεί καλά το admin panel, η βάση επιτρέπει εύκολα ερωτήματα όπως:

- Πόσα αρχεία έχει κάθε χρήστης;
- Πόσο χώρο καταναλώνει κάθε χρήστης;
- Πόσα uploads έγιναν ανά ημέρα τις τελευταίες 30 ημέρες;
- Ποια είναι τα τελευταία 20 αρχεία που ανέβηκαν;

Αυτά υλοποιούνται με COUNT, SUM, GROUP BY και ταξινόμηση ORDER BY created_at.

Κεφάλαιο 6ο: Υλοποίηση της πλατφόρμας

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται η υλοποίηση της πλατφόρμας σε επίπεδο κώδικα και δομής project. Στόχος δεν είναι να φανεί καθαρά πώς υλοποιήθηκαν οι βασικές λειτουργίες: σύνδεση χρήστη, ενεργοποίηση λογαριασμού με κωδικό (claim), μεταφόρτωση mp3 με όρια, δημιουργία public link για Genially, διαχείριση αρχείων (rename/delete), σελίδα αλλαγής κωδικού, καθώς και το admin περιβάλλον με στατιστικά.

Η υλοποίηση έγινε με Python/Flask και MySQL, ενώ για το UI χρησιμοποιήθηκαν Bootstrap templates. Για τα admin γραφήματα χρησιμοποιήθηκε Chart.js. Επιπλέον, υλοποιήθηκε δημόσιο endpoint serving αρχείων ήχου που υποστηρίζει byte-range requests για καλύτερη συμβατότητα με players όπως αυτούς που χρησιμοποιεί το Genially.

6.2 Δομή του project (φάκελοι και αρχεία)

Η εφαρμογή οργανώθηκε με τρόπο που να είναι σχετικά "καθαρή" και διαχειρίσιμη. Μια ενδεικτική δομή είναι:

- app.py: κύριο αρχείο εφαρμογής (routes, login, admin, endpoints)
- models.py: ορισμός πινάκων/μοντέλων (User, AudioFile)
- config.py: ρυθμίσεις (DB URI, upload folder, limits, base URL)
- utils.py: βοηθητικές συναρτήσεις (uuid, δημιουργία φακέλων, byte-range responses)
- templates/: HTML templates (login, dashboard, admin pages, password page)
- static/: στατικά αρχεία (css/js, Chart.js usage κ.λπ.)
- uploads/: αποθήκευση mp3 στο filesystem (ανά χρήστη)

Η επιλογή αυτή βοηθάει ώστε:

- οι ρυθμίσεις να μην είναι "σκορπισμένες",
- ο κώδικας να μην γίνεται τεράστιο μονοκόμματο,
- και να υπάρχει ξεκάθαρη διάκριση μεταξύ UI και backend.

6.3 Ρυθμίσεις και βασικές παράμετροι λειτουργίας

Στο config.py ορίζονται μερικές βασικές σταθερές που επηρεάζουν όλη την εφαρμογή:

- UPLOAD_FOLDER: φάκελος αποθήκευσης των mp3

- MAX_CONTENT_LENGTH: μέγιστο μέγεθος request (10MB)
- MAX_FILES_PER_USER: όριο αρχείων ανά χρήστη (20)
- BASE_URL: η "βάση" του public link (για να εμφανίζεται σωστά στον χρήστη)
- ADMIN_PAGE_SIZE: pagination στις admin λίστες

Αυτό επιτρέπει να αλλάξουν εύκολα τα όρια ή ο τρόπος λειτουργίας χωρίς αλλαγές παντού στον κώδικα.

6.4 Υλοποίηση εισόδου (login) και συνεδρίας χρήστη

Για τη διαχείριση login χρησιμοποιήθηκε Flask-Login. Αυτό δίνει:

- sessions,
- @login_required για προστασία σελίδων,
- current_user για πρόσβαση στον τρέχοντα χρήστη.

Η σελίδα /login εμφανίζει τη φόρμα, ενώ το POST /login εκτελεί τον έλεγχο στοιχείων.

6.5 Υλοποίηση "Claim" λογαριασμού (email + κωδικός)

Η πιο σημαντική διαφοροποίηση από μια τυπική εφαρμογή είναι ότι:

- δεν υπάρχει register,
- ο admin δημιουργεί "pending" χρήστες με email = NULL,
- και ο φοιτητής ενεργοποιεί τον λογαριασμό με email + κωδικό.

Η λογική έχει δύο μονοπάτια:

1. **Κανονικό login**
    - Αν υπάρχει ήδη χρήστης με αυτό το email, γίνεται έλεγχος κωδικού.
2. **Claim**
    - Αν δεν υπάρχει email στη βάση, τότε γίνεται αναζήτηση σε pending χρήστες (email = NULL) που ταιριάζει ο κωδικός.
    - Αν βρεθεί, συμπληρώνεται το email και γίνεται login.

Έτσι ο χρήστης "γράφεται" στο σύστημα χωρίς να χρειάζεται ξεχωριστή διαδικασία εγγραφής.

6.6 Upload mp3 με όρια (10MB, 20 αρχεία)

Η μεταφόρτωση γίνεται από το dashboard σε POST endpoint /upload.

Κατά το upload ελέγχονται:

- **Πλήθος αρχείων**: αν ο χρήστης έχει ήδη 20, απορρίπτεται.
- **Τύπος αρχείου**: επιτρέπεται μόνο .mp3.

- **Μέγεθος**: το request κόβεται αν είναι πάνω από 10MB.

Αν όλα είναι σωστά:

- δημιουργείται ένα μοναδικό uuid5 (5 χαρακτήρες),
- το αρχείο αποθηκεύεται ως uuid5.mp3 στο φάκελο του χρήστη,
- γίνεται εγγραφή στη βάση audio_files,
- και ο χρήστης βλέπει το public link.

6.7 Δημιουργία public link και σταθερό URL για Genially

Το link που παίρνει ο χρήστης είναι της μορφής:

BASE_URL/audio/<uuid>.mp3

Το σημαντικό εδώ είναι ότι:

- ο χρήστης μπορεί να αλλάξει το display name,
- αλλά το uuid δεν αλλάζει,
- άρα το link που έχει μπει στο Genially δεν "σπάει".

Αυτό είναι κρίσιμο γιατί σε εκπαιδευτικό υλικό είναι πιθανό να έχουν ήδη δημοσιευτεί/μοιραστεί οι σύνδεσμοι.

6.8 Serving mp3 με byte-range υποστήριξη

Το endpoint /audio/<uuid>.mp3 είναι δημόσιο (χωρίς login) ώστε να μπορεί να το ζητήσει το Genially.

Η υλοποίηση περιλαμβάνει:

- αναζήτηση στη βάση με βάση το uuid,
- ανάγνωση από το storage_path,
- επιστροφή με Content-Type: audio/mpeg,
- και υποστήριξη HTTP Range ώστε να παίζει σωστά και με seek/buffering.

Στην πράξη αυτό βοηθάει πολύ στη συμβατότητα, γιατί πολλοί players περιμένουν 206 Partial Content όταν κάνουν seek.

6.9 Rename και delete αρχείων

Rename

Ο χρήστης μπορεί να αλλάξει το display_name από το dashboard. Η αλλαγή:

- ενημερώνει μόνο τη βάση
- δεν επηρεάζει το uuid ούτε το αρχείο στο filesystem

Delete

Η διαγραφή:

- αφαιρεί το record από τη βάση
- διαγράφει και το mp3 από τον δίσκο
- και έτσι "ελευθερώνεται" χώρος και slot (για το όριο των 20 αρχείων)

6.10 Αλλαγή κωδικού σε ξεχωριστή σελίδα

Υλοποιήθηκε ξεχωριστή σελίδα:

- GET /account/password → εμφανίζει φόρμα
- POST /change_password → εκτελεί αλλαγή

Ο έλεγχος περιλαμβάνει:

- σωστό τρέχοντα κωδικό
- νέο κωδικό με ελάχιστο μήκος
- επανάληψη νέου κωδικού (να ταιριάζουν)

Με επιτυχία ενημερώνεται:

- password_hash (για login)
- και plain_code ώστε να εμφανίζεται στον admin.

6.11 Admin περιβάλλον (ξεχωριστές σελίδες)

Το admin κομμάτι προστατεύεται με role check (admin_required) και περιλαμβάνει:

/admin – Στατιστικά

Εμφανίζει συνοπτικά:

- πλήθος χρηστών/αρχείων,
- συνολικό χώρο,
- uploads ανά ημέρα (π.χ. τελευταίες 30 ημέρες),
- top χρήστες ανά κατανάλωση χώρου,
- πρόσφατα uploads.

Τα γραφήματα γίνονται με Chart.js και τα δεδομένα προέρχονται από SQL queries.

/admin/users – Χρήστες και pending λογαριασμοί

Εδώ ο admin μπορεί:

- να δει όλους τους χρήστες (και pending),
- να δει τον κωδικό (plain_code),
- να ενεργοποιήσει/απενεργοποιήσει χρήστες,
- να δημιουργήσει νέους pending χρήστες με κωδικό (χωρίς email).

/admin/files – Όλα τα αρχεία

Εδώ ο admin μπορεί:

- να δει όλα τα αρχεία όλων των χρηστών,
- να κάνει αναζήτηση (με uuid ή display name),
- να φιλτράρει ανά χρήστη,
- και να διαγράφει αρχεία.

Κεφάλαιο 7ο: Δοκιμή

7.1 Εισαγωγή

Υπάρχουν κρίσιμα σημεία που αν δεν δουλέψουν σωστά, το σύστημα είτε θα γίνεται δύσχρηστο είτε θα σπάει σε πραγματικές συνθήκες χρήσης. Για παράδειγμα:

- Αν δεν δουλεύει σωστά το δημόσιο link, το Genially δεν θα μπορεί να παίξει τον ήχο.
- Αν δεν εφαρμόζονται σωστά τα όρια (10MB, 20 αρχεία), ο server μπορεί να γεμίσει.
- Αν υπάρχει λάθος σε authentication/roles, μπορεί ένας απλός χρήστης να δει admin δεδομένα.
- Αν δεν υπάρχει CSRF προστασία στα POST, ανοίγει "τρύπα" ασφάλειας.
- Αν δεν υπάρχει σωστό handling σε edge cases (π.χ. delete ενός αρχείου που λείπει), θα πέφτει error.

Οπότε, σε αυτό το κεφάλαιο παρουσιάζονται δοκιμές με πραγματικά σενάρια χρήσης, ώστε να φανεί ότι το σύστημα είναι σταθερό και λειτουργικό.

7.2 Προσέγγιση δοκιμών

Οι δοκιμές που έγιναν μπορούν να χωριστούν σε τρεις κατηγορίες:

1. **Δοκιμές βασικής λειτουργίας (λειτουργικές)**
    - login, claim, upload, link, rename, delete, admin pages.
2. **Δοκιμές ορίων και "δύσκολων περιπτώσεων" (edge cases)**
    - υπέρβαση μεγέθους, 21ο αρχείο, λάθος επέκταση, διπλό email, invalid uuid.
3. **Δοκιμές ασφάλειας/πρόσβασης**
    - CSRF, πρόσβαση σε admin endpoints από απλό χρήστη, πρόσβαση σε αρχεία άλλου χρήστη.

Σημαντικό: οι δοκιμές έγιναν με browser (σαν πραγματικός χρήστης), αλλά και με άμεσο άνοιγμα URL (για το Genially public endpoint).

7.3 Σενάρια δοκιμών χρήστη (User)

Δοκιμή 7.1 – Κανονικό login με σωστά στοιχεία

Στόχος: Να ελεγχθεί ότι ένας υπάρχων χρήστης μπορεί να συνδεθεί.

Βήματα:

Άνοιγμα /login

Εισαγωγή email και σωστού κωδικού

Υποβολή φόρμας

Δοκιμή 7.2 – Claim λογαριασμού (email + κωδικός) για pending χρήστη

Στόχος: Να ελεγχθεί η λειτουργία ενεργοποίησης λογαριασμού χωρίς email.

Προϋπόθεση: Υπάρχει pending λογαριασμός με email = NULL και κωδικό που έδωσε ο admin.

Βήματα:

Άνοιγμα /login

Εισαγωγή ενός νέου email (που δεν υπάρχει στη βάση)

Εισαγωγή του κωδικού πρόσβασης που δόθηκε

Υποβολή φόρμας

Αναμενόμενο αποτέλεσμα:

Ο λογαριασμός "δένεται" με το email. Ο χρήστης κάνει login και πηγαίνει /dashboard.

Στη βάση, το email πλέον δεν είναι NULL.

Δοκιμή 7.3 – Upload mp3 (εντός ορίων)

Στόχος: Να ελεγχθεί ότι γίνεται σωστό upload mp3.

Βήματα:

Στο /dashboard, επιλογή mp3 αρχείου (<10MB)

Upload

Αναμενόμενο αποτέλεσμα:

Το αρχείο αποθηκεύεται στον server. Δημιουργείται εγγραφή στον πίνακα audio_files. Εμφανίζεται στον χρήστη το link /audio/<uuid>.mp3.

Δοκιμή 7.4 – Upload αρχείου πάνω από 10MB

Στόχος: Να ελεγχθεί το όριο μεγέθους.

Βήματα:

Επιλογή mp3 αρχείου >10MB

Upload

Αναμενόμενο αποτέλεσμα:

Το upload απορρίπτεται.

Εμφανίζεται μήνυμα ότι το αρχείο είναι πολύ μεγάλο (ή error 413).

Δοκιμή 7.5 – Upload μη επιτρεπτού τύπου αρχείου

Στόχος: Να ελεγχθεί ότι επιτρέπονται μόνο mp3.

Βήματα:

Επιλογή αρχείου .wav ή .txt

Upload

Αναμενόμενο αποτέλεσμα:

Απόρριψη upload. Μήνυμα ότι επιτρέπονται μόνο mp3.

Δοκιμή 7.6 – Όριο 20 αρχείων ανά χρήστη

Στόχος: Να επιβεβαιωθεί ότι το σύστημα σταματά στο 20ο αρχείο.

Βήματα:

Upload μέχρι να φτάσει 20 αρχεία

Προσπάθεια για 21ο upload

Αναμενόμενο αποτέλεσμα:

Το 21ο upload απορρίπτεται.

Εμφανίζεται μήνυμα ότι ο χρήστης έφτασε το όριο.

Δοκιμή 7.7 – Rename αρχείου

Στόχος: Να ελεγχθεί ότι αλλάζει μόνο το display name.

Βήματα:

Επιλογή αρχείου στη λίστα

Rename σε νέο όνομα

Αναμενόμενο αποτέλεσμα:

Αλλάζει το display_name στη βάση. Το URL με uuid παραμένει ίδιο.

Δοκιμή 7.8 – Delete αρχείου

Στόχος: Να ελεγχθεί η σωστή διαγραφή.

Βήματα:

Επιλογή αρχείου στη λίστα

Delete

Αναμενόμενο αποτέλεσμα:

Διαγράφεται η εγγραφή από τη βάση.

Διαγράφεται το mp3 από τον δίσκο.

Μειώνεται το count αρχείων του χρήστη (άρα ελευθερώνεται slot).

Δοκιμή 7.9 – Αλλαγή κωδικού από τη σελίδα /account/password

Στόχος: Να ελεγχθεί η αλλαγή κωδικού και η ενημέρωση της βάσης.

Βήματα:

Άνοιγμα /account/password

Εισαγωγή τρέχοντος κωδικού

Εισαγωγή νέου κωδικού δύο φορές

Υποβολή

Αναμενόμενο αποτέλεσμα:

Μήνυμα επιτυχίας. Στη βάση ενημερώνεται το password_hash (και plain_code αν κρατιέται). Το νέο password δουλεύει στο επόμενο login.

7.4 Δοκιμές δημόσιου mp3 endpoint (για Genially)

Δοκιμή 7.10 – Άνοιγμα public link σε browser

Στόχος: Να ελεγχθεί ότι το link παίζει σαν αρχείο mp3.

Βήματα:

Αντιγραφή BASE_URL/audio/<uuid>.mp3

Άνοιγμα σε νέο tab

Αναμενόμενο αποτέλεσμα:

Το mp3 ξεκινά να παίζει ή κατεβαίνει. Το URL τελειώνει σε .mp3.

Δοκιμή 7.11 – Έλεγχος υποστήριξης byte-range (seek)

Στόχος: Να ελεγχθεί ότι ο server υποστηρίζει partial content.

Βήματα (πρακτικά):

Άνοιγμα του mp3 σε browser player

Μετακίνηση “μπάρας” στη μέση (seek)

Αναμενόμενο αποτέλεσμα:

Η αναπαραγωγή συνεχίζει ομαλά χωρίς να κολλάει.

Δοκιμή 7.12 – Invalid uuid (μη υπαρκτό αρχείο)

Στόχος: Να ελεγχθεί σωστό error handling.

Βήματα:

Άνοιγμα /audio/XXXXX.mp3 με uuid που δεν υπάρχει

Αναμενόμενο αποτέλεσμα:

Επιστρέφεται 404 Not Found. Δεν “κρασάρει” η εφαρμογή.

7.5 Δοκιμές admin λειτουργιών

Δοκιμή 7.13 – Πρόσβαση admin σελίδων μόνο από admin

Στόχος: Να ελεγχθεί role protection.

Βήματα:

Login ως απλός χρήστης

Απόπειρα πρόσβασης στο /admin

Αναμενόμενο αποτέλεσμα:

Απόρριψη (403) ή redirect. Ο απλός χρήστης δεν βλέπει admin δεδομένα.

Δοκιμή 7.14 – Δημιουργία pending χρήστη με κωδικό

Στόχος: Να ελεγχθεί η λειτουργία /admin/users/create.

Βήματα:

Login ως admin

Άνοιγμα /admin/users

Δημιουργία νέου pending χρήστη με κωδικό

Αναμενόμενο αποτέλεσμα: Δημιουργείται νέος χρήστης με email = NULL.

Ο κωδικός φαίνεται (π.χ. plain_code) και μπορεί να δοθεί στον φοιτητή.

Δοκιμή 7.15 – Toggle ενεργό/ανενεργό χρήστη

Στόχος: Να ελεγχθεί ότι ο admin μπορεί να μπλοκάρει έναν χρήστη.

Βήματα:

Admin → /admin/users

Toggle σε inactive

Προσπάθεια login από τον συγκεκριμένο χρήστη

Αναμενόμενο αποτέλεσμα: Το login απορρίπτεται. Ο χρήστης δεν έχει πρόσβαση στο σύστημα.

Δοκιμή 7.16 – Διαγραφή αρχείου από admin

Στόχος: Να ελεγχθεί ότι ο admin μπορεί να καθαρίσει αρχεία.

Βήματα:

Admin → /admin/files

Επιλογή αρχείου και delete

Αναμενόμενο αποτέλεσμα:

Το αρχείο διαγράφεται από DB + disk. Δεν εμφανίζεται πλέον στη λίστα.

7.6 Δοκιμές ασφάλειας

Δοκιμή 7.17 – CSRF προστασία σε POST requests

Στόχος: Να επιβεβαιωθεί ότι χωρίς CSRF token απορρίπτονται οι POST φόρμες.

Βήματα:

Αφαίρεση token (ή αποστολή POST από εργαλείο χωρίς token)

Απόπειρα upload ή delete

Αναμενόμενο αποτέλεσμα: 400 Bad Request με μήνυμα για CSRF token. Αυτό δείχνει ότι το σύστημα δεν δέχεται "τυφλά" POST.

Δοκιμή 7.18 – Πρόσβαση σε αρχείο άλλου χρήστη μέσω dashboard endpoints

Στόχος: Να ελεγχθεί ότι δεν μπορεί κάποιος να κάνει rename/delete αρχείο άλλου.

Βήματα:

Χρήστης Α βρίσκει ένα file_id άλλου χρήστη

Προσπαθεί να καλέσει /files/<id>/delete

Αναμενόμενο αποτέλεσμα:

404 ή απόρριψη. Καμία αλλαγή στα δεδομένα άλλου χρήστη.

7.7 Συνολική αποτίμηση δοκιμών

Με βάση τα παραπάνω σενάρια, η πλατφόρμα θεωρείται λειτουργική και σταθερή για το βασικό εκπαιδευτικό σενάριο:

- οι χρήστες μπορούν να ανεβάζουν και να παίρνουν links,
- τα links δουλεύουν δημόσια και είναι συμβατά με .mp3,
- εφαρμόζονται όρια χρήσης,
- ο admin έχει πλήρη έλεγχο και στατιστικά,
- και υπάρχουν βασικά μέτρα προστασίας (CSRF, hashing, role check).

Κεφάλαιο 8ο: Παρουσίαση εφαρμογής και ενδεικτική χρήση

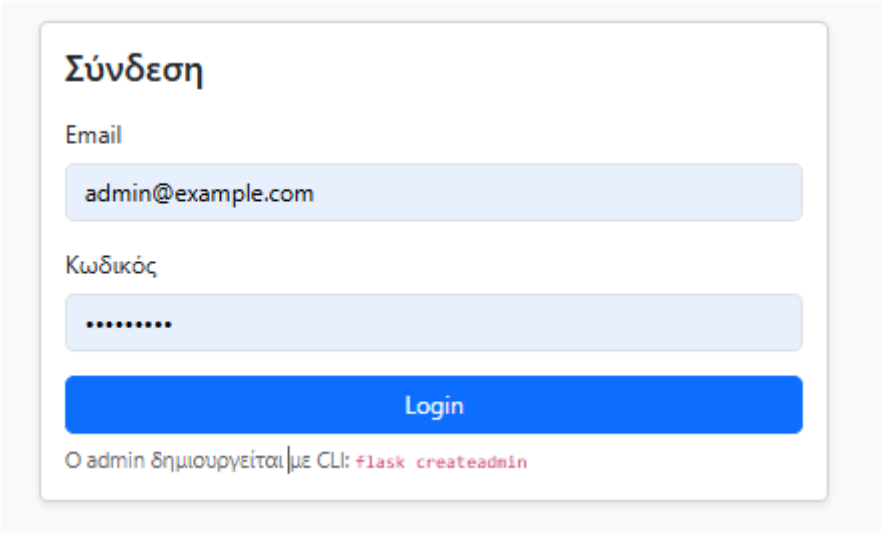
Εδώ παρουσιάζεται η εφαρμογή όπως τη βλέπει ο τελικός χρήστης (φοιτητής/εκπαιδευτικός) και ο διαχειριστής (admin). Στόχος είναι να φανεί στην πράξη πώς χρησιμοποιείται η πλατφόρμα βήμα-βήμα: από την είσοδο στο σύστημα, μέχρι το ανέβασμα αρχείου και την ενσωμάτωση του link στο Genially. Επίσης παρουσιάζονται οι βασικές δυνατότητες οργάνωσης (rename/delete) και το admin περιβάλλον με στατιστικά, ώστε να φαίνεται ότι το σύστημα μπορεί να λειτουργήσει "ομαδικά" σε εκπαιδευτικό πλαίσιο με πολλούς χρήστες.

8.1 Σελίδα σύνδεσης χρήστη

Η πρώτη οθόνη που βλέπει κάποιος είναι η σελίδα login. Ο χρήστης εισάγει το email του και τον κωδικό πρόσβασης.

Υπάρχουν δύο σενάρια:

1. **Κανονικός χρήστης**: υπάρχει ήδη στη βάση με το email του → γίνεται login.
2. **Νέος χρήστης με κωδικό**: ο admin έχει δημιουργήσει pending λογαριασμό (χωρίς email) → ο χρήστης γράφει email + κωδικό και έτσι ενεργοποιείται ο λογαριασμός (claim).



Εικόνα 8.1: Σελίδα σύνδεσης (Login)

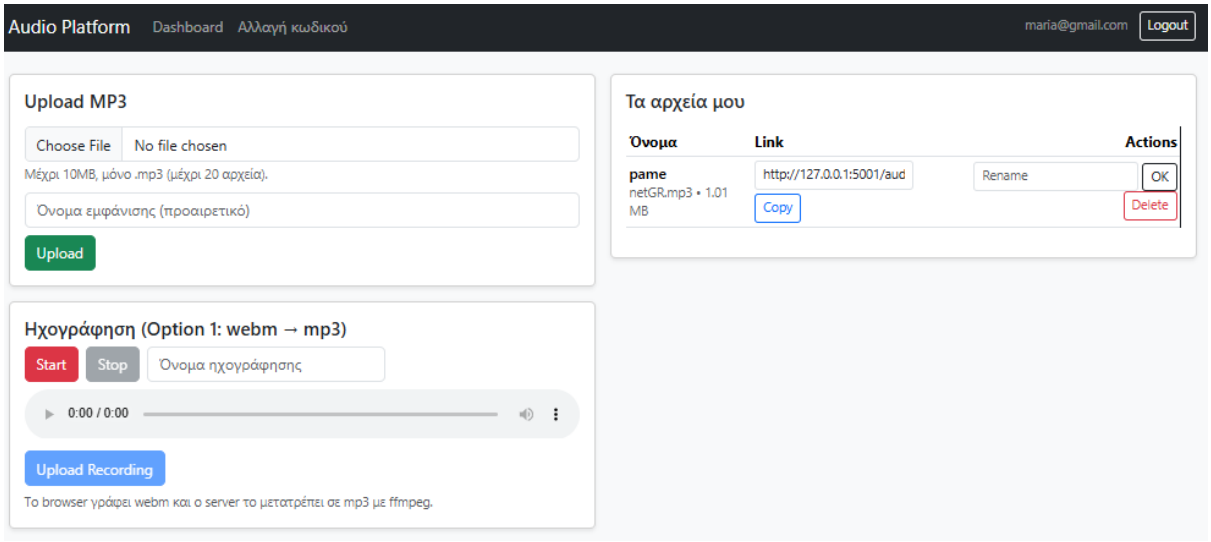
Η σελίδα login επιτρέπει είσοδο με email και κωδικό. Αν το email δεν υπάρχει, τότε γίνεται έλεγχος για pending λογαριασμό με τον συγκεκριμένο κωδικό ώστε να ενεργοποιηθεί ο χρήστης.

8.2 Dashboard χρήστη (λίστα αρχείων και upload)

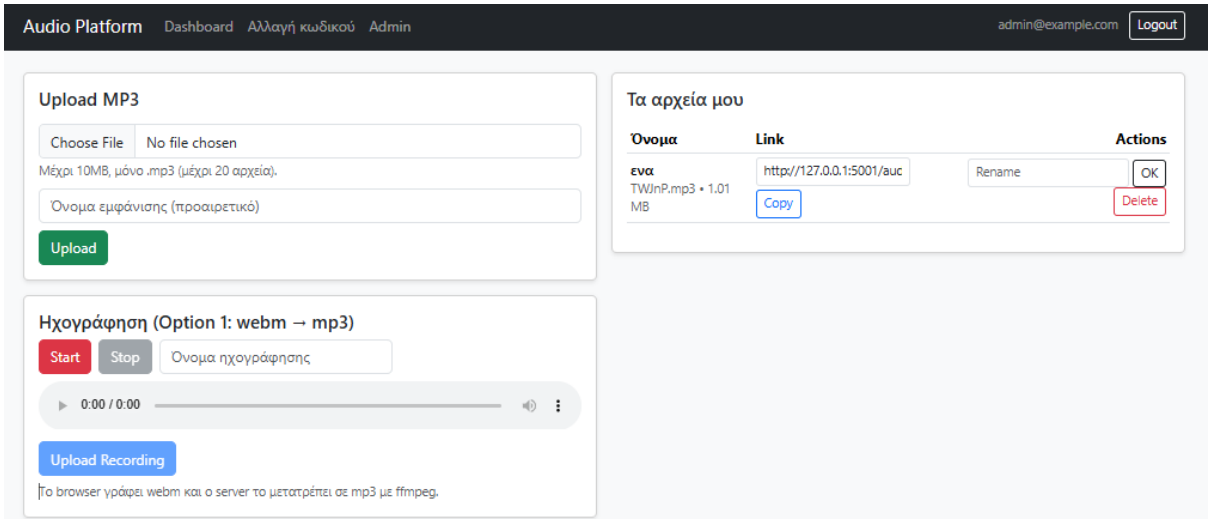
Μετά το login, ο χρήστης μεταφέρεται στο dashboard. Εκεί είναι συγκεντρωμένες οι βασικές λειτουργίες:

- Upload mp3
- Προβολή λίστας αρχείων
- Αντιγραφή public link για Genially
- Rename (όνομα εμφάνισης)
- Delete αρχείου

Το dashboard σχεδιάστηκε ώστε να είναι απλό και "καθαρό", ώστε να μην μπερδεύεται ο χρήστης. Στην εκπαιδευτική χρήση, αυτό είναι σημαντικό, γιατί οι χρήστες δεν είναι απαραίτητα εξοικειωμένοι με τεχνικές λεπτομέρειες.



(α)



(β)

Εικόνα 8.2: Dashboard χρήστη με φόρμα upload και λίστα αρχείων για (α) χρήστη (β) admin

Ο χρήστης ανεβάζει mp3 και βλέπει άμεσα τη λίστα αρχείων του. Για κάθε αρχείο εμφανίζεται το link /audio/<uuid>.mp3 για χρήση στο Genially, καθώς και επιλογές rename και delete.

8.3 Παράδειγμα δημιουργίας και χρήσης public link

Μετά από κάθε upload, η εφαρμογή δημιουργεί ένα μοναδικό όνομα αρχείου βασισμένο σε uuid 5 χαρακτήρων, π.χ.:

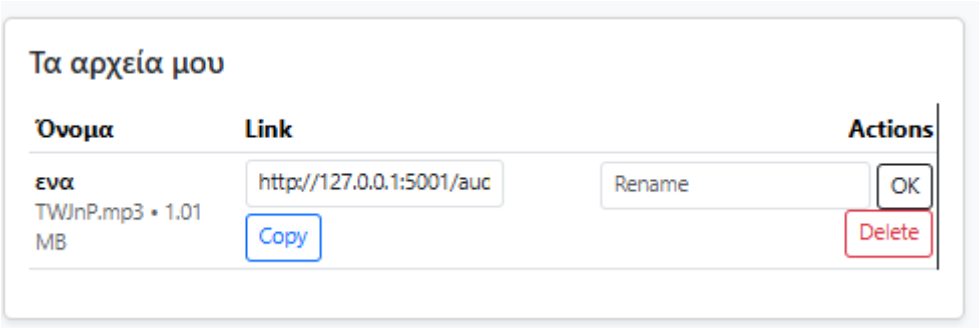
- A7xK2.mp3

Το public link προκύπτει ως:

- BASE_URL/audio/A7xK2.mp3

Αυτό το link είναι:

- δημόσιο (χωρίς login),
- τελειώνει σε .mp3,
- και οδηγεί απευθείας στο αρχείο ήχου.



Εικόνα 8.3: Εμφάνιση public link μετά το upload

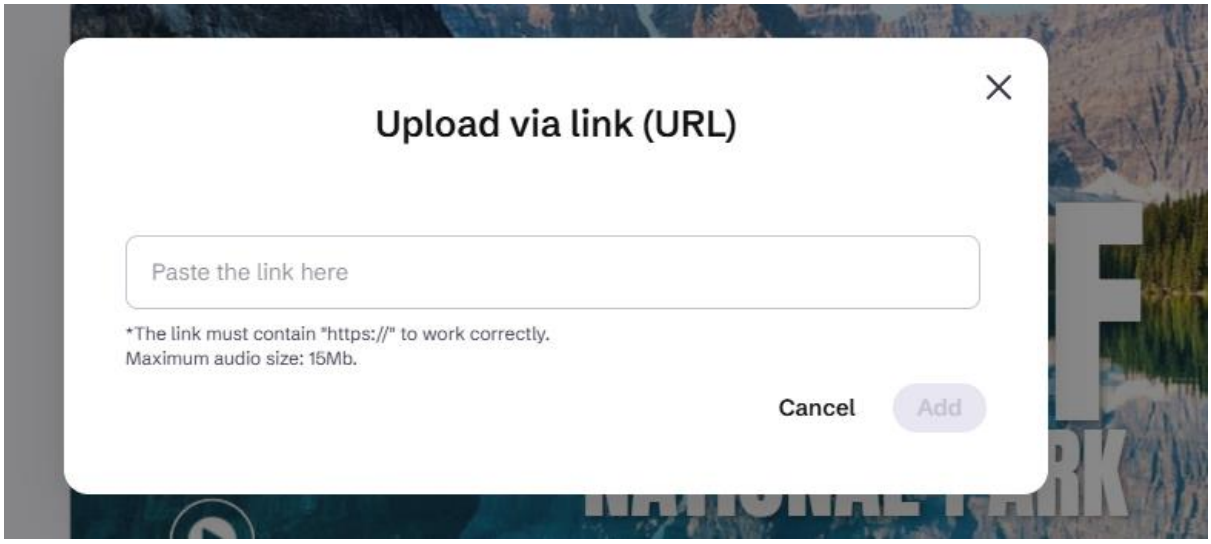
Μετά την επιτυχημένη μεταφόρτωση, το σύστημα εμφανίζει το άμεσο link του mp3. Ο χρήστης μπορεί να το αντιγράψει και να το χρησιμοποιήσει κατευθείαν στο Genially.

Ενσωμάτωση στο Genially

Η χρήση στο Genially είναι η "τελική" αξιοποίηση της πλατφόρμας. Η διαδικασία, σε γενικές γραμμές, είναι:

1. Ο χρήστης ανοίγει το project του στο Genially.
2. Επιλέγει το σημείο που θέλει να προσθέσει ήχο.
3. Επιλέγει προσθήκη ήχου μέσω URL.
4. Επικολλά το link που πήρε από την πλατφόρμα (π.χ. .../audio/A7xK2.mp3).
5. Ο ήχος φορτώνεται και μπορεί να αναπαραχθεί μέσα στο διαδραστικό υλικό.

Το κρίσιμο εδώ είναι ότι το link είναι πραγματικά “direct mp3”, άρα το Genially δεν χρειάζεται κάτι άλλο από το να κάνει ένα απλό GET.



Εικόνα 8.4: Εισαγωγή ήχου στο Genially μέσω URL

Ο χρήστης επικολλά το URL που τελειώνει σε .mp3. Το Genially φορτώνει τον ήχο από το public endpoint της πλατφόρμας και τον ενσωματώνει στο έργο.

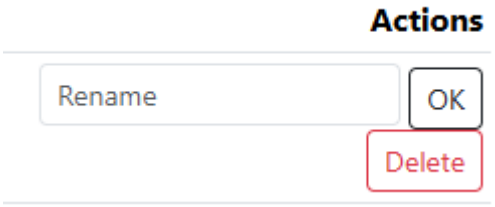
8.4 Rename αρχείου χωρίς να σπάει το Genially link

Ένα πρακτικό σημείο είναι ότι ο χρήστης μπορεί να θέλει να οργανώσει τη λίστα του με ονόματα που βγάζουν νόημα (π.χ. “Οδηγίες_Άσκησης_1”). Αν άλλαζε το πραγματικό filename, θα χαλούσαν τα links στο Genially.

Για αυτό:

- το rename αλλάζει μόνο το display_name (όνομα εμφάνισης),
- ενώ το αρχείο συνεχίζει να σερβίρεται ως uuid.mp3.

Έτσι, το link στο Genially παραμένει ίδιο και ασφαλές.



Εικόνα 8.5: Μετονομασία αρχείου (display name) χωρίς αλλαγή URL

Η εφαρμογή επιτρέπει οργάνωση μέσω rename, αλλά δεν αλλάζει το δημόσιο url. Έτσι, τα ήδη χρησιμοποιημένα links στο Genially δεν χρειάζονται καμία τροποποίηση.

8.5 Διαγραφή αρχείου και απελευθέρωση χώρου/ορίων

Η διαγραφή αρχείου γίνεται από το dashboard και:

- αφαιρεί το record από τη βάση,
- διαγράφει το αρχείο από το filesystem,
- και μειώνει τον συνολικό αριθμό αρχείων του χρήστη (άρα βοηθάει στο όριο των 20).

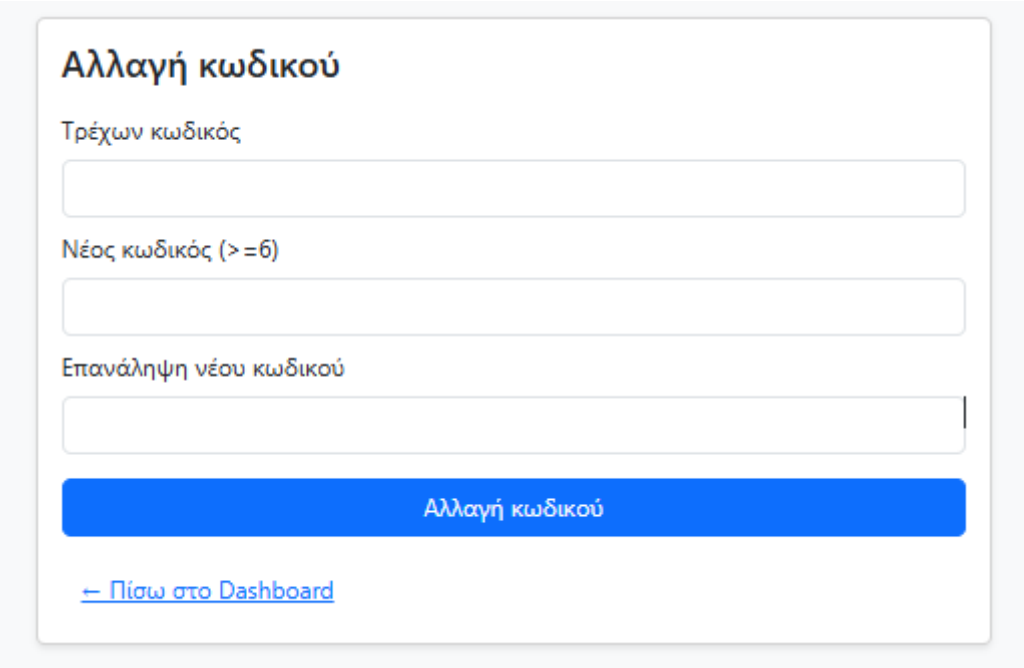
Ο χρήστης μπορεί να διαγράψει αρχεία που δεν χρειάζεται. Αυτό βοηθά στην οργάνωση, αλλά και στην τήρηση των ορίων (20 αρχεία ανά χρήστη).

8.6 Σελίδα αλλαγής κωδικού (ξεχωριστή)

Η αλλαγή κωδικού τοποθετήθηκε σε ξεχωριστή σελίδα (/account/password) ώστε ο χρήστης να έχει ένα "καθαρό" σημείο μόνο για αυτό.

Η φόρμα ζητά:

- τρέχον κωδικό
- νέο κωδικό
- επανάληψη νέου κωδικού



Εικόνα 8.6: Σελίδα αλλαγής κωδικού χρήστη

Ο χρήστης αλλάζει τον κωδικό του με επιβεβαίωση του τρέχοντος. Με επιτυχία εμφανίζεται μήνυμα και ο νέος κωδικός ισχύει από το επόμενο login.

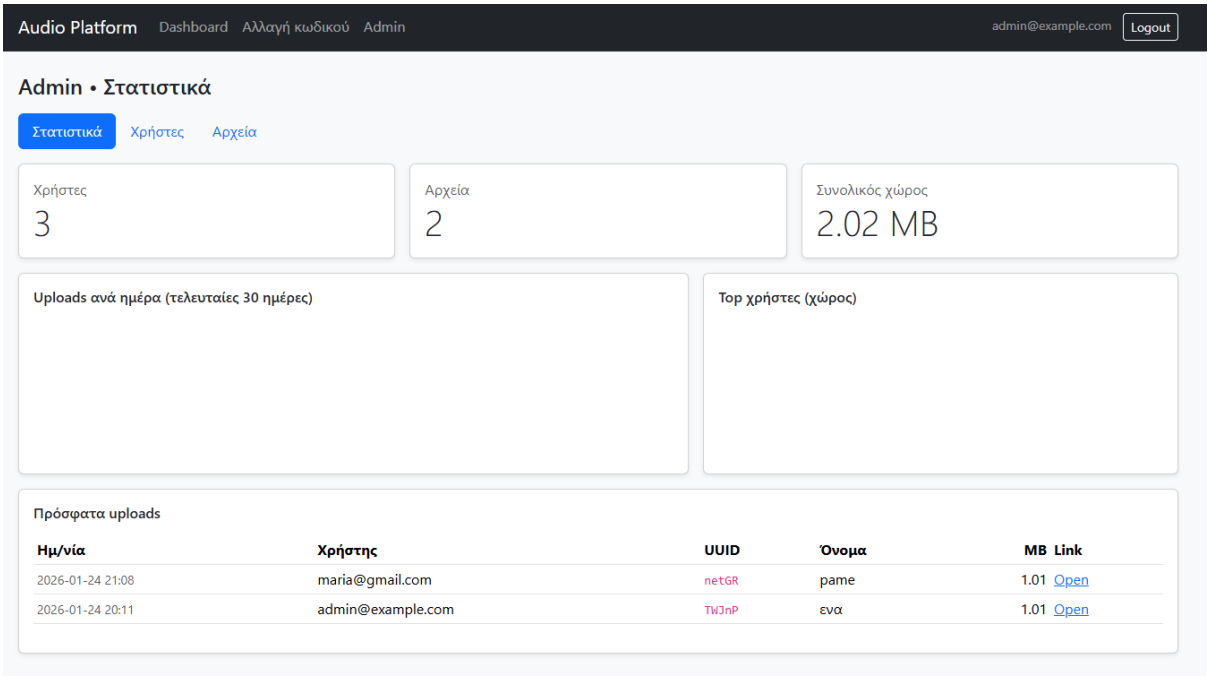
8.7 Admin περιβάλλον (εποπτεία και διαχείριση)

Το admin περιβάλλον είναι ξεχωριστό και προσβάσιμο μόνο από χρήστη με ρόλο admin. Στόχος του είναι η εποπτεία του συστήματος, ειδικά όταν χρησιμοποιείται από πολλούς φοιτητές.

8.7.1 Admin Στατιστικά (/admin)

Στη σελίδα αυτή εμφανίζονται:

- συνολικός αριθμός χρηστών,
- συνολικός αριθμός αρχείων,
- συνολική χρήση χώρου,
- uploads ανά ημέρα (γράφημα),
- top χρήστες σε χρήση χώρου,
- πρόσφατα uploads.



Εικόνα 8.7: Admin Dashboard με στατιστικά και γραφήματα

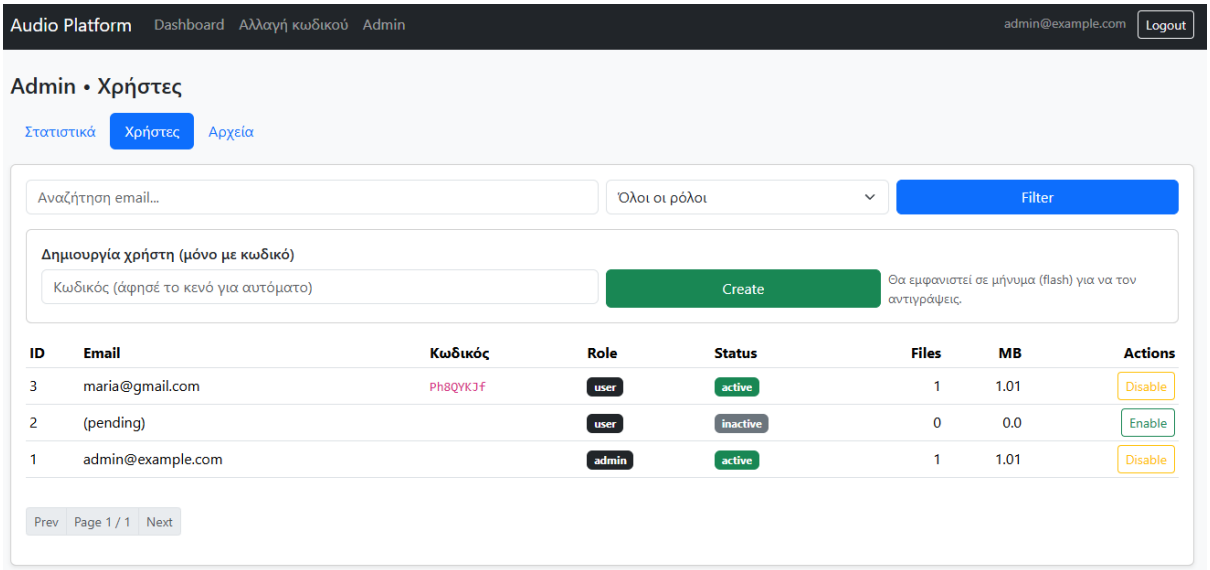
Ο admin βλέπει συνοπτικά τη χρήση της πλατφόρμας (πόσα αρχεία/πόσος χώρος) και εντοπίζει εύκολα περιπτώσεις μεγάλης χρήσης ή αυξημένων uploads.

8.7.2 Διαχείριση χρηστών (/admin/users)

Εδώ ο admin:

- βλέπει όλους τους χρήστες,

- ξεχωρίζει τους pending (email κενό/NULL),
- μπορεί να δημιουργήσει pending χρήστες με κωδικό,
- βλέπει τον κωδικό (plain_code) για να τον δώσει στον φοιτητή,
- και μπορεί να ενεργοποιήσει/απενεργοποιήσει χρήστες.



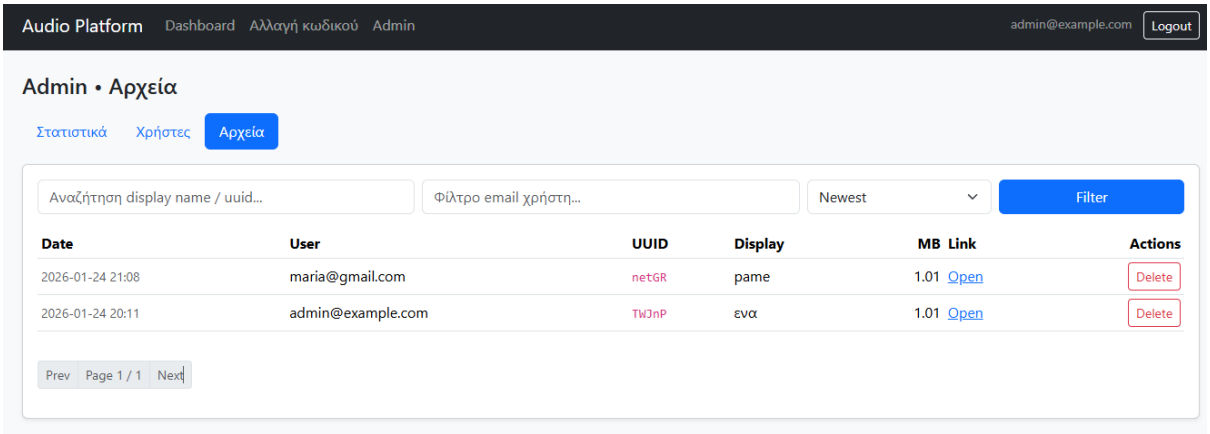
Εικόνα 8.8: Λίστα χρηστών και δημιουργία pending χρήστη με κωδικό

Η σελίδα χρηστών επιτρέπει στον admin να δημιουργεί λογαριασμούς με κωδικό, χωρίς email, και να ελέγχει την κατάσταση ενεργού/ανενεργού.

8.7.3 Διαχείριση αρχείων (/admin/files)

Εδώ ο admin:

- βλέπει όλα τα αρχεία όλων των χρηστών,
- κάνει αναζήτηση/φιλτράρισμα,
- και μπορεί να διαγράψει αρχεία αν χρειαστεί.



Εικόνα 8.9: Λίστα αρχείων (admin) με φίλτρα και δυνατότητα διαγραφής

Ο admin έχει κεντρική εικόνα για όλα τα mp3 που έχουν ανέβει και μπορεί να επέμβει σε περίπτωση λάθους ή κατάχρησης.

Κεφάλαιο 9ο: Συμπεράσματα και μελλοντικές επεκτάσεις

Η βασική επιτυχία της εργασίας είναι ότι αντιμετωπίστηκε ένα πολύ πρακτικό εμπόδιο: η δυσκολία ενσωμάτωσης mp3 στο Genially όταν δεν επιτρέπεται upload αρχείων ήχου, αλλά μόνο εισαγωγή μέσω URL που τελειώνει σε .mp3.

Με την πλατφόρμα που υλοποιήθηκε, ο χρήστης μπορεί:

- να ανεβάσει mp3 αρχεία με απλό τρόπο,
- να πάρει άμεσα ένα δημόσιο link της μορφής /audio/<uuid>.mp3,
- να χρησιμοποιήσει αυτό το link στο Genially χωρίς επιπλέον βήματα,
- να οργανώσει τα αρχεία του (rename/delete),
- και να λειτουργήσει μέσα σε συγκεκριμένα όρια (20 αρχεία ανά χρήστη, 10MB ανά αρχείο).

Από την πλευρά του admin, η πλατφόρμα δίνει:

- κεντρική διαχείριση χρηστών,
- δυνατότητα δημιουργίας λογαριασμών με κωδικό (pending χωρίς email),
- έλεγχο ενεργού/ανενεργού χρήστη,
- διαχείριση αρχείων όλων των χρηστών,
- καθώς και στατιστικά και συνολική εικόνα χρήσης.

Συνολικά, η εφαρμογή λειτουργεί σαν "ενδιάμεσο εργαλείο" που συμπληρώνει τα κενά της πλατφόρμας Genially στο κομμάτι των mp3.

9.1 Πλεονεκτήματα της προσέγγισης

Άμεση συμβατότητα με Genially

Το πιο σημαντικό είναι ότι ο στόχος επιτεύχθηκε με πολύ άμεσο τρόπο: δίνεται URL που τελειώνει σε .mp3 και εξυπηρετείται σωστά το αρχείο. Αυτό σημαίνει ότι ο χρήστης δεν χρειάζεται να ψάχνει τρίτες υπηρεσίες φιλοξενίας ή να κάνει "πατέντες" με links που δεν είναι direct.

Απλότητα χρήσης

Το interface κρατήθηκε "λιτό":

- upload → παίρνω link → το βάζω στο Genially.
  Η απλότητα είναι κρίσιμη σε εκπαιδευτικό πλαίσιο, γιατί οι χρήστες μπορεί να μην έχουν τεχνική εμπειρία.

Οργάνωση αρχείων και σταθερό URL

Η επιλογή να γίνεται rename μόνο στο display name (και όχι στο πραγματικό filename) αποδείχθηκε πολύ πρακτική, γιατί:

- ο χρήστης οργανώνει τη λίστα του,
- αλλά το link δεν αλλάζει ποτέ,
- άρα δεν σπάνε υπάρχοντα projects στο Genially.

Έλεγχος χρήσης (όρια)

Τα όρια 10MB και 20 αρχεία ανά χρήστη κρατάνε το σύστημα "μαζεμένο". Έτσι:

- προστατεύεται ο αποθηκευτικός χώρος,
- περιορίζεται η κατάχρηση,
- και η χρήση παραμένει κοντά στον εκπαιδευτικό σκοπό.

Admin εποπτεία

Το admin panel δίνει εικόνα της κατάστασης:

- ποιοι χρήστες υπάρχουν,
- πόσα αρχεία έχουν,
- πόσο χώρο καταναλώνουν,
- και ποια είναι η δραστηριότητα (uploads ανά ημέρα).
  Αυτό βοηθάει πολύ όταν το σύστημα χρησιμοποιείται σε μάθημα με πολλούς φοιτητές.

9.2 Δυσκολίες και σημεία που χρειάζονται προσοχή

Δημόσια πρόσβαση στα mp3 links

Για να μπορεί να τα "τραβάει" το Genially, τα mp3 links είναι δημόσια. Αυτό είναι αναγκαίο, αλλά δημιουργεί μια βασική πραγματικότητα:

- όποιος έχει το link μπορεί να ακούσει/κατεβάσει το αρχείο.

Στην πράξη, αυτό μειώνεται επειδή το uuid είναι δύσκολο να μαντευτεί, όμως δεν είναι "πλήρης ασφάλεια". Άρα χρειάζεται προσοχή αν κάποτε το σύστημα χρησιμοποιηθεί για ευαίσθητο περιεχόμενο.

Μικρό UUID (5 χαρακτήρες)

Το uuid 5 χαρακτήρων κάνει το link σύντομο και βολικό, αλλά θεωρητικά:

- όσο μικραίνει το token, αυξάνει η πιθανότητα κάποιος να το μαντέψει/σαρώσει με brute force.
  Για ένα μικρό εκπαιδευτικό περιβάλλον αυτό είναι συνήθως OK, αλλά σε μεγαλύτερη κλίμακα θα ήταν καλύτερο να είναι μεγαλύτερο.

Αποθήκευση plain_code

Η αποθήκευση του κωδικού σε απλή μορφή στη βάση (plain_code) έγινε για να καλύψει το συγκεκριμένο σενάριο όπου ο admin θέλει να βλέπει/μοιράζει κωδικούς εύκολα. Όμως, από άποψη ασφάλειας είναι ένα "ευαίσθητο" σημείο. Αν κάποιος αποκτήσει πρόσβαση στη βάση, βλέπει κωδικούς.

Για αυτό, σε μελλοντική έκδοση θα ήταν προτιμότερο:

- να εμφανίζεται ο κωδικός μόνο τη στιγμή δημιουργίας,
- ή να αποθηκεύεται με διαφορετικό τρόπο (π.χ. one-time tokens).

Διαχείριση χώρου σε δίσκο (scaling)

Η αποθήκευση σε τοπικό δίσκο είναι απλή και λειτουργική για διπλωματική/μικρή χρήση. Αν όμως το σύστημα μεγαλώσει:

- μπορεί να γεμίσει ο δίσκος,
- χρειάζεται backup,
- και ίσως χρειαστεί εξωτερικό storage.

9.3 Προτάσεις για μελλοντικές επεκτάσεις

Μεγαλύτερο token για τα public links

Αν χρειαστεί καλύτερη ασφάλεια, μπορεί το uuid να γίνει:

- 8–12 χαρακτήρες ή και περισσότερο,

  ώστε να μειωθεί δραστικά η πιθανότητα brute force.

Προσωρινά links ή υπογραφή (signed URLs)

Σε πιο "επαγγελματικό" σενάριο, θα μπορούσαν να χρησιμοποιηθούν signed URLs που:

- λήγουν μετά από κάποιο χρόνο,
- ή απαιτούν token που δημιουργείται δυναμικά.

  Αυτό όμως είναι πιο δύσκολο με Genially, γιατί το Genially πρέπει να μπορεί να ζητήσει το αρχείο χωρίς να σπάει η πρόσβαση.

Περιορισμός ρυθμού (rate limiting)

Για προστασία από κατάχρηση ή brute force σε public endpoints, μπορεί να προστεθεί:

- rate limiting ανά IP,
- logging για ύποπτες προσπάθειες,
- βασικές πολιτικές anti-abuse.

Βελτίωση μηχανισμού κωδικών (χωρίς plain_code)

Μια καλύτερη πρακτική θα ήταν:

- ο admin να δημιουργεί invite codes,
- ο κωδικός να εμφανίζεται μία φορά,

- και να μην αποθηκεύεται σε απλή μορφή.

  Ή να χρησιμοποιούνται προσωρινά tokens ενεργοποίησης που μετά ακυρώνονται.

Βελτίωση ηχογράφησης μέσα από browser

Αν αξιοποιηθεί το recording:

- μπορεί να προστεθεί preview,
- δυνατότητα pause/resume,
- και αυτόματη ονομασία ηχογράφησης.

Επίσης μπορεί να εξεταστεί αν ο browser παράγει καλύτερο format πριν τη μετατροπή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://genially.com/

[2] https://help.genially.com/en_us/insert-an-audio-from-a-url-in-genially-rJTs9PSni

[3] https://www.w3schools.com/python/

[4] https://www.geeksforgeeks.org/python/flask-tutorial/

[5] https://www.w3schools.com/MySQL/default.asp

[6] https://getbootstrap.com/

[7] https://www.chartjs.org/

[8] https://www.ffmpeg.org/

[9] https://en.wikipedia.org/wiki/Universally_unique_identifier

[10] https://www.w3schools.com/html/

ΠΑΡΑΡΤΗΜΑ Α

```
import os
from datetime import datetime, timedelta, date
from functools import wraps

from flask import Flask, render_template, redirect, url_for, request, flash, abort, jsonify
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from flask_wtf.csrf import CSRFProtect
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename

from sqlalchemy import create_engine, select, func, text
from sqlalchemy.orm import sessionmaker

from config import Config
from models import Base, User, AudioFile
from utils import gen_uuid5, ensure_dir, run_ffmpeg_to_mp3, ranged_file_response

import secrets
import string

BASE62 = string.ascii_letters + string.digits
ALLOWED_EXT = {".mp3"}

def create_app():
    app = Flask(    name    )
    app.config.from_object(Config)

    ensure_dir(app.config["UPLOAD_FOLDER"])

    engine = create_engine(app.config["SQLALCHEMY_DATABASE_URI"], pool_pre_ping=True)
    SessionLocal = sessionmaker(bind=engine, autoflush=False, autocommit=False)

    # CSRF for all POST requests (forms & fetch where applicable)
    csrf = CSRFProtect(app)

    login_manager = LoginManager()
    login_manager.login_view = "login"
    login_manager.init_app(app)

    @login_manager.user_loader
    def load_user(user_id: str):
        with SessionLocal() as db:
            return db.get(User, int(user_id))

    def db_session():
        return SessionLocal()

    def admin_required(fn):
        @wraps(fn)
        def wrapper(*args, **kwargs):
            if not current_user.is_authenticated:
                return redirect(url_for("login"))
            if current_user.role != "admin":
                abort(403)
            return fn(*args, **kwargs)
        return wrapper

    # ---------- Helpers: invites / pending users ----------
    def gen_invite_code(n: int = 8) -> str:
        return "".join(secrets.choice(BASE62) for _ in range(n))
```

```
def find_pending_by_code(db, code: str):
    """Find a pending (email is NULL) active user account that matches the given code."""
    if not code:
        return None
    candidates = db.scalars(
        select(User).where(User.email.is_(None), User.role == "user", User.is_active == True)  # noqa: E712
    ).all()
    for u in candidates:
        if check_password_hash(u.password_hash, code):
            return u
    return None

# ---------- CLI helpers ----------
@app.cli.command("initdb")
def initdb():
    Base.metadata.create_all(engine)
    print("DB tables ensured.")

@app.cli.command("createadmin")
def createadmin():
    email = os.environ.get("ADMIN_EMAIL", "admin@example.com").strip().lower()
    pwd = os.environ.get("ADMIN_PASSWORD", "admin1234")
    with db_session() as db:
        exists = db.scalar(select(User).where(User.email == email))
        if exists:
            print("Admin already exists.")
            return
        u = User(
            email=email,
            password_hash=generate_password_hash(pwd),
            plain_code=pwd,  # visible in DB/admin table (as requested)
            role="admin",
            is_active=True,
        )
        db.add(u)
        db.commit()
        print(f"Admin created: {email} / {pwd}")

# ---------- Auth ----------
@app.get("/login")
def login():
    return render_template("login.html")

@app.post("/login")
def login_post():
    email = request.form.get("email", "").strip().lower()
    password = request.form.get("password", "")

    if not email:
        flash("Το email είναι υποχρεωτικό.", "warning")
        return redirect(url_for("login"))

    with db_session() as db:
        user = db.scalar(select(User).where(User.email == email))

        # 1) Normal login
        if user:
            if (not user.is_active) or (not check_password_hash(user.password_hash, password)):
                flash("Λάθος στοιχεία ή ανενεργός λογαριασμός.", "danger")
                return redirect(url_for("login"))

            user.last_login = datetime.utcnow()
            db.commit()
            login_user(user)
            return redirect(url_for("dashboard"))

        # 2) Claim flow: email not found -> try claim a pending account by code
```

```
        pending = find_pending_by_code(db, password)
        if not pending:
            flash("Λάθος στοιχεία. (Δεν βρέθηκε κωδικός πρόσβασης για εγγραφή)", "danger")
            return redirect(url_for("login"))

        # Double-check email uniqueness (race condition)
        exists = db.scalar(select(User).where(User.email == email))
        if exists:
            flash("Αυτό το email υπάρχει ήδη.", "danger")
            return redirect(url_for("login"))

        pending.email = email
        pending.last_login = datetime.utcnow()
        db.commit()

        login_user(pending)
        flash("Ο λογαριασμός ενεργοποιήθηκε επιτυχώς!", "success")
        return redirect(url_for("dashboard"))

@app.get("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for("login"))

# ---------- User dashboard ----------
@app.get("/")
def home():
    if current_user.is_authenticated:
        return redirect(url_for("dashboard"))
    return redirect(url_for("login"))

@app.get("/account/password")
@login_required
def account_password():
    return render_template("account_password.html")

@app.get("/dashboard")
@login_required
def dashboard():
    with db_session() as db:
        files = db.scalars(
            select(AudioFile)
            .where(AudioFile.user_id == current_user.id)
            .order  by(AudioFile.created  at.desc())
        ).all()
    return render_template("dashboard.html", files=files, base_url=app.config["BASE_URL"])

def user_file_count(db, user_id: int):
    return db.scalar(select(func.count()).select_from(AudioFile).where(AudioFile.user_id == user_id))

def generate_unique_uuid5(db):
    for _ in range(80):
        u = gen_uuid5()
        exists = db.scalar(select(AudioFile).where(AudioFile.uuid5 == u))
        if not exists:
            return u
    raise RuntimeError("Could not generate unique uuid5")

def save_mp3_for_user(db, file_storage, original_name: str, display_name: str):
    count = user_file_count(db, current_user.id)
    if count >= app.config["MAX_FILES_PER_USER"]:
        abort(400, description="Εχεις φτάσει το όριο των 20 αρχείων.")

    filename = secure_filename(original_name)
    ext = os.path.splitext(filename.lower())[1]
    if ext not in ALLOWED_EXT:
```

```
        abort(400, description="Επιτρέπονται μόνο αρχεία .mp3")

    if request.content_length and request.content_length > app.config["MAX_CONTENT_LENGTH"]:
        abort(413)

    uuid5 = generate_unique_uuid5(db)

    user_dir = os.path.join(app.config["UPLOAD_FOLDER"], f"u{current_user.id}")
    ensure_dir(user_dir)

    out_path = os.path.join(user_dir, f"{uuid5}.mp3")
    file_storage.save(out_path)
    size_bytes = os.path.getsize(out_path)

    rec = AudioFile(
        user_id=current_user.id,
        uuid5=uuid5,
        original_name=filename,
        display_name=(display_name or filename)[:255],
        size_bytes=size_bytes,
        mime_type="audio/mpeg",
        storage_path=out_path,
    )
    db.add(rec)
    db.commit()
    return rec

@app.post("/upload")
@login_required
def upload():
    f = request.files.get("file")
    if not f or f.filename == "":
        flash("Διάλεξε ένα αρχείο.", "warning")
        return redirect(url_for("dashboard"))

    display_name = request.form.get("display_name", "").strip()
    try:
        with db_session() as db:
            rec = save_mp3_for_user(db, f, f.filename, display_name)
            flash("Ανέβηκε επιτυχώς!", "success")
            flash(f"Link: {app.config['BASE_URL']}/audio/{rec.uuid5}.mp3", "info")
    except Exception as e:
        flash(f"Σφάλμα upload: {str(e)}", "danger")

    return redirect(url_for("dashboard"))

@app.post("/files/<int:file_id>/rename")
@login_required
def rename(file_id):
    new_name = request.form.get("new_name", "").strip()
    if not new_name:
        flash("Δώσε νέο όνομα.", "warning")
        return redirect(url_for("dashboard"))

    with db_session() as db:
        rec = db.get(AudioFile, file_id)
        if not rec or rec.user_id != current_user.id:
            abort(404)
        rec.display_name = new_name[:255]
        db.commit()

    flash("Το όνομα άλλαξε.", "success")
    return redirect(url_for("dashboard"))

@app.post("/files/<int:file_id>/delete")
@login_required
def delete(file_id):
```

```
    with db_session() as db:
        rec = db.get(AudioFile, file_id)
        if not rec or rec.user_id != current_user.id:
            abort(404)
        path = rec.storage_path
        db.delete(rec)
        db.commit()

    try:
        if path and os.path.exists(path):
            os.remove(path)
    except Exception:
        pass

    flash("Διαγράφηκε.", "success")
    return redirect(url_for("dashboard"))

# ---------- Change password (updates plain_code too, as requested) ----------
@app.post("/change  password")
@login_required
def change_password():
    current_pw = request.form.get("current_password", "")
    new_pw = request.form.get("new_password", "")
    new_pw2 = request.form.get("new_password2", "")

    if not new_pw or len(new_pw) < 6:
        flash("Ο νέος κωδικός πρέπει να έχει τουλάχιστον 6 χαρακτήρες.", "warning")
        return redirect(url_for("dashboard"))

    if new_pw != new_pw2:
        flash("Οι νέοι κωδικοί δεν ταιριάζουν.", "warning")
        return redirect(url_for("dashboard"))

    with db_session() as db:
        u = db.get(User, current_user.id)
        if not u:
            abort(404)

        if not check_password_hash(u.password_hash, current_pw):
            flash("Λάθος τρέχων κωδικός.", "danger")
            return redirect(url_for("dashboard"))

        u.password_hash = generate_password_hash(new_pw)
        u.plain_code = new_pw  # stored in DB for admin visibility (per requirement)
        db.commit()

    flash("Ο κωδικός άλλαξε επιτυχώς.", "success")
    return redirect(url_for("dashboard"))

# ---------- Public mp3 serving (Genially) with byte-range ----------
@app.get("/audio/<uuid5>.mp3")
def serve_audio(uuid5):
    with db_session() as db:
        rec = db.scalar(select(AudioFile).where(AudioFile.uuid5 == uuid5))
        if not rec:
            abort(404)

    if not os.path.exists(rec.storage_path):
        abort(404)

    return ranged_file_response(rec.storage_path, "audio/mpeg", f"{uuid5}.mp3")

# ---------- Recording option 1: webm -> ffmpeg -> mp3 ----------
@app.post("/record_upload")
@login_required
def record_upload():
    blob = request.files.get("blob")
```

```
    display_name = (request.form.get("display_name") or "recording").strip()

    if not blob:
        return jsonify({"ok": False, "error": "No blob"}), 400

    if request.content_length and request.content_length > app.config["MAX_CONTENT_LENGTH"]:
        return jsonify({"ok": False, "error": "Too large"}), 413

    with db_session() as db:
        count = user_file_count(db, current_user.id)
        if count >= app.config["MAX_FILES_PER_USER"]:
            return jsonify({"ok": False, "error": "Limit 20 files reached"}), 400

        uuid5 = generate_unique_uuid5(db)

        user_dir = os.path.join(app.config["UPLOAD_FOLDER"], f"u{current_user.id}")
        ensure_dir(user_dir)

        temp  webm = os.path.join(user  dir, f"{uuid5}.webm")
        out_mp3 = os.path.join(user_dir, f"{uuid5}.mp3")

        blob.save(temp_webm)

        try:
            run_ffmpeg_to_mp3(app.config["FFMPEG_BIN"], temp_webm, out_mp3)
        except Exception as e:
            try:
                if os.path.exists(temp_webm):
                    os.remove(temp_webm)
            except Exception:
                pass
            return jsonify({"ok": False, "error": str(e)}), 500
        finally:
            try:
                if os.path.exists(temp_webm):
                    os.remove(temp_webm)
            except Exception:
                pass

        size_bytes = os.path.getsize(out_mp3)

        rec = AudioFile(
            user_id=current_user.id,
            uuid5=uuid5,
            original  name="recording.webm",
            display_name=display_name[:255],
            size_bytes=size_bytes,
            mime_type="audio/mpeg",
            storage_path=out_mp3,
        )
        db.add(rec)
        db.commit()

    link = f"{app.config['BASE_URL']}/audio/{uuid5}.mp3"
    return jsonify({"ok": True, "uuid5": uuid5, "link": link})

# ---------- Admin: create pending users (code only, empty email) ----------
@app.post("/admin/users/create")
@login_required
@admin_required
def admin_create_user_invite():
    code = (request.form.get("code") or "").strip()
    if not code:
        code = gen_invite_code(8)

    with db_session() as db:
        u = User(
```

```
            email=None,
            password_hash=generate_password_hash(code),
            plain_code=code,
            role="user",
            is_active=True,
        )
        db.add(u)
        db.commit()

    flash(f"Δημιουργήθηκε κωδικός χρήστη: {code}", "success")
    return redirect(url_for("admin_users"))

# ---------- Admin: stats ----------
@app.get("/admin")
@login_required
@admin_required
def admin_index():
    with db_session() as db:
        users_total = db.scalar(select(func.count()).select_from(User))
        files_total = db.scalar(select(func.count()).select_from(AudioFile))
        bytes_total = db.scalar(select(func.coalesce(func.sum(AudioFile.size_bytes), 0)))
        total_mb = round((bytes_total or 0) / 1024 / 1024, 2)

        rows = db.execute(text("""
            SELECT DATE(created_at) as d, COUNT(*) as c
            FROM audio_files
            WHERE created_at >= (CURRENT_DATE - INTERVAL 29 DAY)
            GROUP BY DATE(created_at)
            ORDER BY d ASC
        """)).all()

        day_map = {str(r.d): int(r.c) for r in rows}
        labels = []
        values = []
        for i in range(30):
            dd = (date.today() - timedelta(days=29 - i))
            s = dd.isoformat()
            labels.append(s)
            values.append(day_map.get(s, 0))

        top = db.execute(text("""
            SELECT u.email as email, COALESCE(SUM(a.size_bytes),0) as b
            FROM users u
            LEFT JOIN audio_files a ON a.user_id = u.id
            GROUP BY u.id
            ORDER BY b DESC
            LIMIT 7
        """)).all()
        top_labels = [r.email or "(pending)" for r in top]
        top_values = [round((r.b or 0) / 1024 / 1024, 2) for r in top]

        recent = db.execute(text("""
            SELECT a.id as id, a.uuid5 as uuid5, a.display_name as display_name, a.created_at as created_at,
                   a.size_bytes as size_bytes, u.email as email
            FROM audio_files a
            JOIN users u ON u.id = a.user_id
            ORDER BY a.created_at DESC
            LIMIT 20
        """)).all()

    recent_view = [{
        "id": r.id,
        "uuid5": r.uuid5,
        "display_name": r.display_name,
        "created_at": r.created_at.strftime("%Y-%m-%d %H:%M") if hasattr(r.created_at, "strftime") else str(r.created_at),
        "mb": round((r.size_bytes or 0) / 1024 / 1024, 2),
        "email": r.email or "(pending)",
```

```
    } for r in recent]

    return render_template(
        "admin/index.html",
        active="stats",
        base_url=app.config["BASE_URL"],
        totals={"users": users_total, "files": files_total, "total_mb": total_mb},
        uploads={"labels": labels, "values": values},
        top_space={"labels": top_labels, "values": top_values},
        recent=recent_view,
    )

# ---------- Admin: users ----------
@app.get("/admin/users")
@login_required
@admin_required
def admin_users():
    page = max(1, int(request.args.get("page", "1")))
    q = (request.args.get("q") or "").strip().lower()
    role = (request.args.get("role") or "").strip()

    limit = app.config["ADMIN_PAGE_SIZE"]
    offset = (page - 1) * limit

    where = []
    params = {}
    if q:
        where.append("LOWER(COALESCE(u.email,'')) LIKE :q")
        params["q"] = f"%{q}%"
    if role in ("user", "admin"):
        where.append("u.role = :role")
        params["role"] = role

    where_sql = ("WHERE " + " AND ".join(where)) if where else ""

    with db_session() as db:
        total = db.execute(text(f"SELECT COUNT(*) FROM users u {where_sql}"), params).scalar() or 0
        pages = max(1, (total + limit - 1) // limit)

        rows = db.execute(text(f"""
            SELECT u.id, u.email, u.role, u.is_active, u.plain_code,
                (SELECT COUNT(*) FROM audio_files a WHERE a.user_id=u.id) as file_count,
                (SELECT COALESCE(SUM(a.size_bytes),0) FROM audio_files a WHERE a.user_id=u.id) as bytes_sum
            FROM users u
            {where_sql}
            ORDER BY u.id DESC
            LIMIT :limit OFFSET :offset
        """), {**params, "limit": limit, "offset": offset}).all()

    users = [{
        "id": r.id,
        "email": r.email if r.email else "(pending)",
        "role": r.role,
        "plain_code": r.plain_code or "",
        "is_active": bool(r.is_active),
        "file_count": int(r.file_count or 0),
        "mb": round((r.bytes_sum or 0) / 1024 / 1024, 2),
    } for r in rows]

    return render_template(
        "admin/users.html",
        active="users",
        users=users,
        page=page,
        pages=pages,
        q=q,
        role=role,
```

```
    )

@app.post("/admin/users/<int:user_id>/toggle")
@login_required
@admin_required
def admin_toggle_user(user_id):
    next_url = request.form.get("next") or url_for("admin_users")
    if user_id == current_user.id:
        flash("Δεν μπορείς να απενεργοποιήσεις τον εαυτό σου.", "warning")
        return redirect(next_url)

    with db_session() as db:
        u = db.get(User, user_id)
        if not u:
            abort(404)
        u.is_active = not bool(u.is_active)
        db.commit()

    flash("Εγινε αλλαγή κατάστασης.", "success")
    return redirect(next_url)

# ---------- Admin: files ----------
@app.get("/admin/files")
@login_required
@admin_required
def admin_files():
    page = max(1, int(request.args.get("page", "1")))
    q = (request.args.get("q") or "").strip().lower()
    user = (request.args.get("user") or "").strip().lower()
    sort = (request.args.get("sort") or "new").strip()

    limit = app.config["ADMIN_PAGE_SIZE"]
    offset = (page - 1) * limit

    where = []
    params = {}
    if q:
        where.append("(LOWER(a.display_name) LIKE :q OR LOWER(a.uuid5) LIKE :q)")
        params["q"] = f"%{q}%"
    if user:
        where.append("LOWER(COALESCE(u.email,")) LIKE :ue")
        params["ue"] = f"%{user}%"

    where_sql = ("WHERE " + " AND ".join(where)) if where else ""

    order_sql = "ORDER BY a.created_at DESC"
    if sort == "old":
        order_sql = "ORDER BY a.created_at ASC"
    elif sort == "big":
        order_sql = "ORDER BY a.size_bytes DESC"

    with db_session() as db:
        total = db.execute(text(f"""
            SELECT COUNT(*)
            FROM audio_files a
            JOIN users u ON u.id = a.user_id
            {where_sql}
        """), params).scalar() or 0
        pages = max(1, (total + limit - 1) // limit)

        rows = db.execute(text(f"""
            SELECT a.id, a.uuid5, a.display_name, a.created_at, a.size_bytes, u.email
            FROM audio_files a
            JOIN users u ON u.id = a.user_id
            {where_sql}
            {order_sql}
            LIMIT :limit OFFSET :offset
```

```
            """), {**params, "limit": limit, "offset": offset}).all()

        files = [{
            "id": r.id,
            "uuid5": r.uuid5,
            "display_name": r.display_name,
            "created_at": r.created_at.strftime("%Y-%m-%d %H:%M") if hasattr(r.created_at, "strftime") else str(r.created_at),
            "mb": round((r.size_bytes or 0) / 1024 / 1024, 2),
            "email": r.email or "(pending)",
        } for r in rows]

        return render_template(
            "admin/files.html",
            active="files",
            base_url=app.config["BASE_URL"],
            files=files,
            page=page,
            pages=pages,
            q=q,
            user=user,
            sort=sort,
        )

    @app.post("/admin/files/<int:file_id>/delete")
    @login_required
    @admin_required
    def admin_delete_file(file_id):
        next_url = request.form.get("next") or url_for("admin_files")
        with db_session() as db:
            rec = db.get(AudioFile, file_id)
            if not rec:
                abort(404)
            path = rec.storage_path
            db.delete(rec)
            db.commit()

        try:
            if path and os.path.exists(path):
                os.remove(path)
        except Exception:
            pass

        flash("Το αρχείο διαγράφηκε.", "success")
        return redirect(next_url)

    return app

app = create_app()

if __name__ == "__main__":
    app.run(debug=True, port=5001)
```