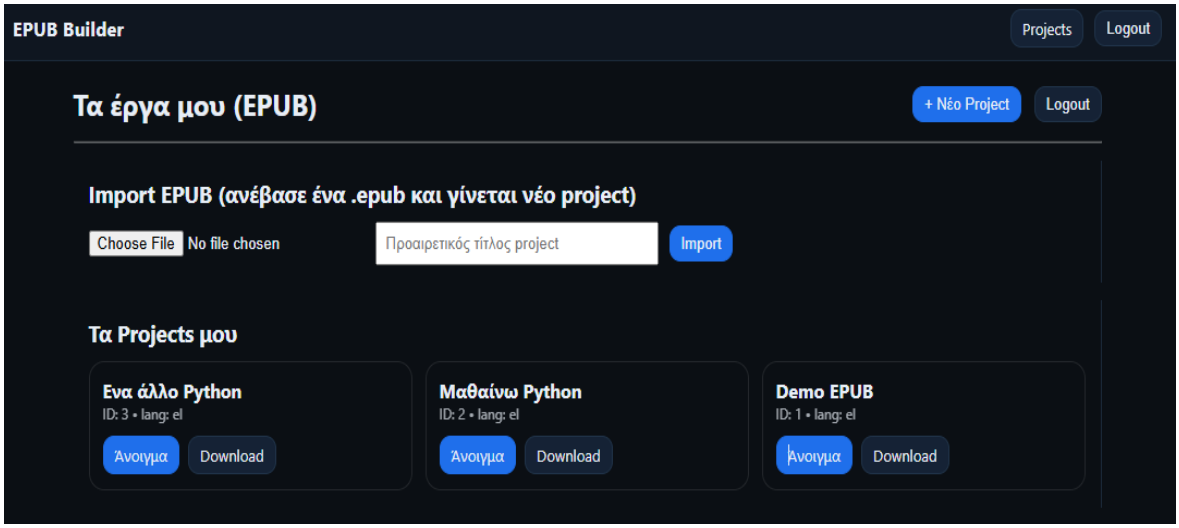
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ενιαίο σύστημα διαχείρισης περιεχομένου epub»



Φοιτητής

ΑΠΟΣΤΟΛΟΣ ΣΑΝΔΑΛΙΔΗΣ

185271

Επιβλέπων

Δρ. Κυριάκος Τσιακμάκης

ΙΑΝ 2025

Ενιαίο σύστημα διαχείρισης περιεχομένου epub

Κωδικός: **25339**

Φοιτητής: ΑΠΟΣΤΟΛΟΣ ΣΑΝΔΑΛΙΔΗΣ

Εισηγητής: Δρ Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 31-10-2025

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 πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της πτυχιακής 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 πτυχιακή εργασία παρουσιάζει την ανάπτυξη ενός ενιαίου συστήματος διαχείρισης περιεχομένου για ηλεκτρονικά βιβλία EPUB. Το σύστημα υλοποιείται ως διαδικτυακή εφαρμογή, όπου κάθε χρήστης συνδέεται με λογαριασμό και διαχειρίζεται τα δικά του έργα (projects). Παρέχεται περιβάλλον εργασίας τριών panels όπου αριστερά εμφανίζονται τα αρχεία του βιβλίου με δυνατότητα δημιουργίας, ανεβάσματος, μετονομασίας, διπλασιασμού, διαγραφής και αλλαγής σειράς μέσω drag & drop, στο κέντρο υπάρχει editor τόσο σε code mode (XHTML/CSS) όσο και σε rich mode και δεξιά εμφανίζεται live προεπισκόπηση του αποτελέσματος. Τα κείμενα αποθηκεύονται στη MySQL, ενώ τα binary αρχεία (π.χ. εικόνες) αποθηκεύονται στον δίσκο. Η εξαγωγή ακολουθεί τη σειρά που ορίζει ο χρήστης και υποστηρίζεται παραγωγή και κατέβασμα αρχείων EPUB και PDF.

«Unified EPUB Content Management System»

Abstract

This thesis presents the development of a unified content management system for EPUB e-books. The system is implemented as a web application where each user logs in and manages personal projects (books). A three-panel workspace is provided: the left panel lists the book files and supports creating, uploading, renaming, duplicating, deleting, and reordering files via drag & drop, the center panel offers editing in both code mode (XHTML/CSS) and rich-text mode and the right panel displays a live preview of the rendered output. Text-based resources are stored in a MySQL database, while binary assets (e.g., images) are stored on disk. Export generation follows the user-defined order, enabling consistent reading flow without requiring manual package editing. The system supports building and downloading both EPUB and PDF outputs.

Ευχαριστίες

Θέλω να ευχαριστήσω όσους με βοήθησαν να φέρω εις πέρας αυτή την εργασία.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η ψηφιακή έκδοση βιβλίων και εκπαιδευτικού υλικού έχει εξελιχθεί ραγδαία τα τελευταία χρόνια, καθώς όλο και περισσότεροι χρήστες διαβάζουν περιεχόμενο από κινητές συσκευές, tablets και e-readers. Η ανάγκη για ευέλικτα, φορητά και συμβατά ηλεκτρονικά βιβλία έχει οδηγήσει στην καθιέρωση ανοικτών προτύπων, με κυρίαρχο το EPUB, το οποίο χρησιμοποιείται ευρέως από πλατφόρμες ανάγνωσης και εργαλεία διανομής. Σε αντίθεση με κλειστά formats, το EPUB βασίζεται σε τεχνολογίες ιστού (XHTML και CSS) και προσφέρει δυνατότητα προσαρμογής του περιεχομένου ανάλογα με τη συσκευή και τις προτιμήσεις του αναγνώστη. Ωστόσο, παρόλο που το πρότυπο είναι ιδιαίτερα διαδεδομένο, η παραγωγή και η σωστή δομή ενός EPUB δεν είναι πάντα εύκολη διαδικασία, ειδικά για άτομα που δεν έχουν εμπειρία στη συγγραφή κώδικα ή στη διαχείριση αρχείων [1-4].

Ένα σημαντικό πρόβλημα που εμφανίζεται στην πράξη είναι ότι πολλά εργαλεία δημιουργίας EPUB είτε είναι υπερβολικά “κλειστά” είτε απαιτούν εξειδικευμένες γνώσεις (δίνουν πλήρη έλεγχο αλλά αυξάνουν τη δυσκολία). Για παράδειγμα, ένας χρήστης που θέλει να δημιουργήσει εκπαιδευτικό υλικό, σημειώσεις ή ένα μικρό βιβλίο, συχνά χρειάζεται να διαχειριστεί πολλά διαφορετικά αρχεία (κεφάλαια σε XHTML, CSS για μορφοποίηση, εικόνες, πίνακες περιεχομένων, μεταδεδομένα) και να γνωρίζει πώς όλα αυτά συνδέονται μεταξύ τους μέσα σε ένα έγκυρο EPUB. Η οποιαδήποτε ασυνέπεια, όπως λάθος relative paths, ελλιπές manifest, προβληματική σειρά ή μη σωστή ενσωμάτωση εικόνων μπορεί να οδηγήσει σε βιβλίο που δεν ανοίγει, εμφανίζεται λάθος ή απορρίπτεται από συστήματα διανομής. Ενώ θεωρητικά το EPUB είναι απλό, πρακτικά απαιτεί έναν ολοκληρωμένο τρόπο διαχείρισης.

Στο πλαίσιο αυτό, η παρούσα πτυχιακή εργασία έχει ως στόχο τον σχεδιασμό και την υλοποίηση ενός “Ενιαίου Συστήματος Διαχείρισης Περιεχομένου EPUB”, δηλαδή μιας διαδικτυακής εφαρμογής που συγκεντρώνει όλα τα βασικά στάδια δημιουργίας και επεξεργασίας ενός βιβλίου σε ένα σημείο. Η φιλοσοφία του συστήματος είναι να λειτουργεί ως ένα CMS (Content Management System), ειδικά προσαρμοσμένο για EPUB. [5] Ο χρήστης δημιουργεί ένα έργο (project) που αντιστοιχεί σε ένα βιβλίο, βλέπει τα αρχεία του σε ένα περιβάλλον τύπου file manager, μπορεί να τα επεξεργαστεί με κώδικα ή με πλούσιο κειμενογράφο (rich editor), να ανεβάσει εικόνες και αρχεία, να ελέγχει άμεσα το αποτέλεσμα μέσω προεπισκόπησης, και τέλος να δημιουργεί εξαγώγιμο αρχείο EPUB για διανομή ή χρήση. Με αυτόν τον τρόπο επιδιώκεται να μειωθεί η πολυπλοκότητα χωρίς να “χαθεί” ο έλεγχος και η δυνατότητα λεπτομερούς μορφοποίησης που προσφέρει το πρότυπο.

Βασικός πυρήνας της εφαρμογής είναι η διαχείριση πολλαπλών έργων ανά χρήστη. Κάθε χρήστης, αφού συνδεθεί στο σύστημα, μπορεί να δει τα προσωπικά του έργα, να δημιουργήσει νέο βιβλίο ή να

εισάγει ένα υπάρχον EPUB που του έχει δοθεί (π.χ. από συνεργάτη ή εκδότη). Η δυνατότητα εισαγωγής είναι ιδιαίτερα σημαντική, διότι σε πραγματικές συνθήκες ο χρήστης συχνά δεν ξεκινά από το μηδέν· μπορεί να έχει ένα πρόχειρο EPUB που θέλει να διορθώσει, να προσαρμόσει ή να εμπλουτίσει. Με το import, το σύστημα δημιουργεί νέο project και "σπάει" το EPUB στα επιμέρους αρχεία του (XHTML, CSS, εικόνες), τα οποία γίνονται διαθέσιμα για επεξεργασία, χωρίς ο χρήστης να χρειάζεται χειροκίνητο unzip ή αναδιοργάνωση. Έτσι, το σύστημα λειτουργεί ως "χώρος εργασίας" για οποιοδήποτε EPUB, είτε αυτό έχει δημιουργηθεί από το ίδιο είτε προέρχεται από εξωτερική πηγή.

Ένα ακόμα κομβικό στοιχείο είναι το περιβάλλον επεξεργασίας και προεπισκόπησης. Η εργασία πάνω σε EPUB δεν είναι απλή συγγραφή κειμένου: περιλαμβάνει δομή κεφαλαίων, μορφοποίηση, διαχείριση εικόνων, υπερσυνδέσμων, καθώς και έλεγχο της τελικής εμφάνισης. Για αυτό το σύστημα υλοποιεί τριπλό περιβάλλον εργασίας: αριστερά εμφανίζεται η λίστα αρχείων του βιβλίου, στο κέντρο υπάρχει ο editor (με δυνατότητα τόσο "code mode" και "rich mode"), και δεξιά εμφανίζεται live preview του τρέχοντος κεφαλαίου. Με αυτόν τον τρόπο ο χρήστης έχει άμεσο feedback και κάθε αλλαγή που κάνει στο XHTML ή στο CSS μπορεί να ελεγχθεί άμεσα, περιορίζοντας τον χρόνο δοκιμών και τα λάθη. Παράλληλα, η προσθήκη λειτουργιών τύπου "δεξί κλικ" (rename, duplicate, delete) και drag-and-drop ταξινόμησης των κεφαλαίων, κάνει το σύστημα πιο κοντά σε εργαλεία διαχείρισης αρχείων που ήδη γνωρίζουν οι χρήστες.

Ιδιαίτερη σημασία στο EPUB έχει η σειρά ανάγνωσης (spine order). Ακόμα και αν όλα τα αρχεία υπάρχουν σωστά, αν η σειρά δεν είναι η επιθυμητή, το βιβλίο "διαβάζεται λάθος", κάτι που είναι κρίσιμο σε εκπαιδευτικό υλικό ή σε βιβλία με συγκεκριμένη ροή. Πολλά εργαλεία επιβάλλουν μια σειρά (π.χ. αλφαβητική) ή απαιτούν χειροκίνητη επεξεργασία του OPF. Στην παρούσα εργασία επιλέγεται μια πιο πρακτική προσέγγιση: η σειρά καθορίζεται από τον ίδιο τον χρήστη μέσω drag-and-drop στη λίστα των αρχείων, και αυτή η σειρά αποθηκεύεται στη βάση δεδομένων. Κατά την παραγωγή του EPUB, το σύστημα ανακατασκευάζει το spine με βάση αυτή τη σειρά, χωρίς να επιβάλλει συγκεκριμένες "ειδικές" σελίδες όπως index ή toc. Αυτό δίνει πλήρη ελευθερία στον χρήστη να οργανώσει το βιβλίο όπως θέλει, να προσθέσει ή να αφαιρέσει κεφάλαια και να δημιουργήσει τη δική του δομή, ενώ το σύστημα απλώς εγγυάται ότι το αποτέλεσμα θα εξαχθεί με συνέπεια.

Πέρα από το EPUB, στην πράξη πολλοί χρήστες χρειάζονται και εξαγωγή σε PDF, είτε για εκτύπωση είτε για διανομή σε πλατφόρμες που προτιμούν στατικά έγγραφα. Παρότι το EPUB και το PDF διαφέρουν φιλοσοφικά (reflowable vs fixed layout), η ύπαρξη και των δύο μορφών στο ίδιο σύστημα αυξάνει τη χρησιμότητά του. Έτσι, η εργασία εξετάζει και την παραγωγή PDF ως επιπλέον δυνατότητα εξαγωγής, με έμφαση στη σωστή απόδοση ελληνικών χαρακτήρων και στην ενσωμάτωση μορφοποίησης και εικόνων. Η πρόκληση εδώ είναι ότι διαφορετικές βιβλιοθήκες PDF υποστηρίζουν σε διαφορετικό βαθμό το CSS, ενώ απαιτείται και σωστή διαχείριση γραμματοσειρών ώστε να αποφεύγονται προβλήματα εμφάνισης.

Από πλευράς υλοποίησης, το σύστημα σχεδιάζεται ως web εφαρμογή με backend σε Python (Flask) και βάση δεδομένων MySQL. Η επιλογή αυτής της αρχιτεκτονικής δεν είναι τυχαία: η web υλοποίηση επιτρέπει πρόσβαση από οποιοδήποτε περιβάλλον χωρίς εγκατάσταση εξειδικευμένου editor, ενώ η MySQL προσφέρει αξιόπιστη αποθήκευση των projects και των αρχείων. Τα κείμενα (XHTML/CSS) αποθηκεύονται στη βάση για εύκολη αναζήτηση και διαχείριση, ενώ τα binary αρχεία (όπως εικόνες) αποθηκεύονται στον δίσκο σε οργανωμένη δομή ανά project, ώστε να είναι αποδοτική η διαχείριση μεγάλων αρχείων και να αποφεύγεται υπερφόρτωση της βάσης. Με αυτό το υβριδικό μοντέλο, επιτυγχάνεται ισορροπία ανάμεσα σε ευκολία διαχείρισης και πρακτικότητα αποθήκευσης.

Τέλος, η εργασία δεν περιορίζεται μόνο στην ανάπτυξη "ενός ακόμα εργαλείου", αλλά επιδιώκει να αποτυπώσει και τη λογική ενός ολοκληρωμένου workflow παραγωγής EPUB: από το αρχικό υλικό, στην οργάνωση των κεφαλαίων, στη μορφοποίηση, στη διαχείριση πόρων, και στην τελική εξαγωγή. Μέσα από αυτό το πρίσμα, το σύστημα λειτουργεί ως ένα ενιαίο περιβάλλον εργασίας που μπορεί να υποστηρίξει εκπαιδευτικούς, φοιτητές, δημιουργούς περιεχομένου ή μικρές ομάδες που θέλουν να παράγουν ψηφιακά βιβλία χωρίς να χρειάζεται να "πηδάνε" από εργαλείο σε εργαλείο. Η έμφαση δίνεται στην πρακτική χρηστικότητα, στη μείωση του τεχνικού κόστους και στην παροχή λειτουργιών που είναι κοντά στις πραγματικές ανάγκες δημιουργίας και επιμέλειας.

Στα επόμενα κεφάλαια θα παρουσιαστούν αρχικά το θεωρητικό υπόβαθρο του EPUB και τα βασικά δομικά του στοιχεία, στη συνέχεια οι απαιτήσεις του συστήματος και οι επιλογές σχεδίασης, και τέλος η υλοποίηση, η αξιολόγηση και τα συμπεράσματα. Με αυτόν τον τρόπο, η εργασία καλύπτει τόσο την τεχνική διάσταση της ανάπτυξης μιας web εφαρμογής όσο και την ειδική γνώση που απαιτείται για την ορθή διαχείριση του περιεχομένου ενός EPUB, καταλήγοντας σε ένα ολοκληρωμένο σύστημα που μπορεί να χρησιμοποιηθεί άμεσα σε πραγματικά σενάρια.

Κεφάλαιο 2ο: Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζεται το θεωρητικό πλαίσιο που απαιτείται για την κατανόηση και την τεκμηρίωση ενός ενιαίου συστήματος διαχείρισης EPUB. Αρχικά περιγράφεται το πρότυπο EPUB και ο τρόπος με τον οποίο οργανώνει το περιεχόμενο ενός ηλεκτρονικού βιβλίου. Στη συνέχεια αναλύεται η χρήση XHTML και CSS στο πλαίσιο των eBooks, καθώς και οι ιδιαιτερότητες σε σχέση με την κλασική ανάπτυξη ιστοσελίδων. Τέλος, γίνεται σύνδεση με την έννοια των συστημάτων διαχείρισης περιεχομένου (CMS) και εξηγείται γιατί μια CMS-προσέγγιση είναι κατάλληλη για την παραγωγή και συντήρηση EPUB έργων.

2.1 Το πρότυπο EPUB: Σκοπός και βασικές αρχές

Το EPUB (Electronic Publication) αποτελεί ένα από τα πιο διαδεδομένα πρότυπα ηλεκτρονικών βιβλίων. Η βασική ιδέα του είναι να προσφέρει ένα ανοικτό, φορητό και συμβατό format που να μπορεί να αναγνωστεί σε διαφορετικές συσκευές και εφαρμογές. Σε αντίθεση με το PDF, το οποίο είναι κατεξοχήν “στατικό” και σχεδιασμένο για εκτύπωση ή ακριβή αναπαράσταση σελίδων, το EPUB είναι κατά κανόνα reflowable: το περιεχόμενο προσαρμόζεται στην οθόνη, στο μέγεθος γραμματοσειράς και στις προτιμήσεις του χρήστη. Αυτό είναι ιδιαίτερα κρίσιμο σε e-readers και κινητές συσκευές, όπου το ίδιο κείμενο πρέπει να παραμένει ευανάγνωστο ανεξαρτήτως αν ο χρήστης διαβάζει σε οθόνη 6 ιντσών ή σε tablet.

Το EPUB δεν είναι απλώς “ένα αρχείο”. Στην πραγματικότητα είναι ένα συμπιεσμένο πακέτο (zip container) που περιλαμβάνει πολλαπλά επιμέρους αρχεία: XHTML για το κείμενο, CSS για μορφοποίηση, εικόνες, γραμματοσειρές, καθώς και αρχεία μεταδεδομένων και πλοήγησης. Η ύπαρξη αυτής της δομής επιτρέπει μεγάλη ευελιξία, αλλά ταυτόχρονα εισάγει πολυπλοκότητα: ένα λάθος σε κάποιο αρχείο ή μια λάθος σύνδεση μεταξύ τους μπορεί να επηρεάσει ολόκληρο το βιβλίο.

2.2 Εσωτερική δομή EPUB: Container, OPF, Manifest, Spine

Το EPUB (Electronic Publication) αποτελεί ένα από τα πιο διαδεδομένα πρότυπα ηλεκτρονικών βιβλίων. Η βασική ιδέα του είναι να προσφέρει ένα ανοικτό, φορητό και συμβατό format που να μπορεί να αναγνωστεί σε διαφορετικές συσκευές και εφαρμογές. Σε αντίθεση με το PDF, το οποίο είναι κατεξοχήν “στατικό” και σχεδιασμένο για εκτύπωση ή ακριβή αναπαράσταση σελίδων, το EPUB είναι κατά κανόνα reflowable: το περιεχόμενο προσαρμόζεται στην οθόνη, στο μέγεθος γραμματοσειράς και

στις προτιμήσεις του χρήστη. Αυτό είναι ιδιαίτερα κρίσιμο σε e-readers και κινητές συσκευές, όπου το ίδιο κείμενο πρέπει να παραμένει ευανάγνωστο ανεξαρτήτως αν ο χρήστης διαβάζει σε οθόνη 6 ιντσών ή σε tablet.

Το EPUB δεν είναι απλώς "ένα αρχείο". Στην πραγματικότητα είναι ένα συμπιεσμένο πακέτο (zip container) που περιλαμβάνει πολλαπλά επιμέρους αρχεία: XHTML για το κείμενο, CSS για μορφοποίηση, εικόνες, γραμματοσειρές, καθώς και αρχεία μεταδεδομένων και πλοήγησης. Η ύπαρξη αυτής της δομής επιτρέπει μεγάλη ευελιξία, αλλά ταυτόχρονα εισάγει πολυπλοκότητα: ένα λάθος σε κάποιο αρχείο ή μια λάθος σύνδεση μεταξύ τους μπορεί να επηρεάσει ολόκληρο το βιβλίο.

2.3 XHTML σε EPUB: κανόνες, περιορισμοί και συμβατότητα

Ένα EPUB οργανώνεται με τρόπο που θυμίζει "μικρή ιστοσελίδα" αλλά με συγκεκριμένους κανόνες. Σε υψηλό επίπεδο, υπάρχουν τρία κρίσιμα στοιχεία:

1. **Container (META-INF/container.xml)**

   Είναι το αρχείο που δείχνει πού βρίσκεται το βασικό αρχείο πακέτου (OPF). Χωρίς αυτό, ο reader δεν γνωρίζει ποιο αρχείο περιγράφει το περιεχόμενο.

2. **Package Document (OPF)**

   Το OPF (Open Packaging Format) είναι το "κεντρικό index" του EPUB [6]. Περιλαμβάνει:

   - **Metadata**: τίτλος, γλώσσα, δημιουργός, αναγνωριστικό κ.ά.
   - **Manifest**: λίστα όλων των αρχείων που περιλαμβάνει το βιβλίο (XHTML, CSS, εικόνες, fonts κ.λπ.), μαζί με το media-type τους.
   - **Spine**: τη σειρά ανάγνωσης του βιβλίου, δηλαδή ποια XHTML κεφάλαια εμφανίζονται και με ποια σειρά.

3. **Navigation (toc/nav)**

   Η πλοήγηση μπορεί να περιλαμβάνει NCX (κυρίως για παλαιότερη συμβατότητα) και ένα NAV document (XHTML) για EPUB3. Η ύπαρξη πίνακα περιεχομένων δεν είναι μόνο θέμα "ευκολίας", αλλά συχνά απαιτείται για να θεωρείται το EPUB πλήρες και σωστά δομημένο.

Το **manifest** και το **spine** είναι από τα πιο σημαντικά σημεία. Ενώ το manifest αφορά το "τι υπάρχει" μέσα στο EPUB, το spine αφορά το "πώς διαβάζεται". Για ένα σύστημα διαχείρισης, αυτά είναι κρίσιμα, γιατί πρέπει να υπάρχει συνέπεια μεταξύ αρχείων που ο χρήστης βλέπει/επεξεργάζεται και του τρόπου που θα "πακεταριστούν" στην τελική εξαγωγή.

2.4 CSS και eBook μορφοποίηση

Η μορφοποίηση στο EPUB γίνεται κυρίως με CSS. Αν και το CSS είναι το ίδιο concept με το web, δεν υποστηρίζονται πάντα όλες οι ιδιότητες και, κυρίως, η συμπεριφορά μπορεί να διαφέρει ανά reader. Για παράδειγμα:

- κάποιοι readers αγνοούν συγκεκριμένα properties,
- άλλοι επιβάλλουν δικά τους default styles,
- και συχνά υπάρχουν περιορισμοί σε fonts, margins ή advanced layout.

Παρ' όλα αυτά, το CSS είναι απαραίτητο για να δημιουργηθεί ένα αναγνώσιμο βιβλίο: τίτλοι, παράγραφοι, αποστάσεις, λίστες, blocks κώδικα (σε τεχνικά βιβλία όπως Python), καθώς και κανόνες για εικόνες. Ειδικά σε τεχνικό περιεχόμενο, η σωστή εμφάνιση code και pre είναι κρίσιμη, καθώς από εκεί εξαρτάται η κατανόηση παραδειγμάτων. Επιπλέον, η τυπογραφία (font-family, line-height) επηρεάζει σημαντικά την εμπειρία ανάγνωσης.

Στο πλαίσιο της εργασίας, το σύστημα διαχείρισης χρειάζεται να:

- επιτρέπει στον χρήστη να έχει ένα ή περισσότερα CSS αρχεία,
- εφαρμόζει σωστά αυτά τα styles στο preview,
- και να διασφαλίζει ότι κατά το build (EPUB/PDF) τα styles θα "περάσουν" στο τελικό αποτέλεσμα.

Ένα πρακτικό ζήτημα που εμφανίζεται συχνά είναι η σωστή σύνδεση CSS με XHTML (μέσω <link> ή μέσω injection). Στο preview μπορεί να γίνεται injection για άμεσο αποτέλεσμα, ενώ στο EPUB πρέπει να διατηρηθούν σωστά οι relative αναφορές και οι συνδέσεις, ώστε οι readers να φορτώνουν το CSS από το πακέτο.

2.5 Εικόνες και πόροι (assets): relative paths και αποθήκευση

Οι εικόνες και γενικά οι "πόροι" ενός EPUB (images, fonts, icons) αποτελούν ξεχωριστή κατηγορία αρχείων. Ενώ τα XHTML/CSS είναι κείμενο και μπορούν να αποθηκευτούν εύκολα σε βάση δεδομένων, οι εικόνες είναι binary δεδομένα. Αυτό δημιουργεί δύο προκλήσεις:

1. **Πού αποθηκεύονται;**

   Μια πρακτική λύση είναι να αποθηκεύονται οι εικόνες στον δίσκο, σε φάκελο ανά project (π.χ. uploads/<project_id>/...). Έτσι αποφεύγεται το "φούσκωμα" της βάσης και βελτιώνεται η απόδοση.

2. **Πώς συνδέονται μέσα στο XHTML;**

   Στο EPUB, οι εικόνες αναφέρονται με relative paths, π.χ. ../images/pic.png. Αν ο χρήστης μετακινήσει αρχεία ή αλλάξει δομή φακέλων, μπορεί να "σπάσουν" οι σύνδεσμοι. Γι' αυτό ένα ενιαίο σύστημα πρέπει να παρέχει ασφαλή τρόπο ανεβάσματος και να εισάγει σωστά paths μέσα στο περιεχόμενο.

Επιπλέον, μια σύγχρονη απαίτηση είναι η εισαγωγή εικόνων απευθείας από rich editor (π.χ. Summernote). Αυτό σημαίνει ότι ο χρήστης κάνει paste ή insert εικόνα, και το σύστημα αυτόματα κάνει upload και γράφει το σωστό <img src="...">. Αυτή η λειτουργία αυξάνει σημαντικά την ευχρηστία και μειώνει τον τεχνικό φόρτο.

2.6 CMS προσέγγιση για EPUB: γιατί "ενιαίο σύστημα"

Τα CMS συστήματα, όπως είναι γνωστά από το web (π.χ. WordPress, Drupal), οργανώνουν την παραγωγή περιεχομένου με βάση έννοιες όπως: έργα, σελίδες, ρόλοι, αποθήκευση, επεξεργασία και δημοσίευση. Παρότι το EPUB δεν είναι μια "online" ιστοσελίδα, έχει πολλά κοινά με τη διαχείριση περιεχομένου:

- υπάρχουν πολλαπλά "κομμάτια" περιεχομένου (κεφάλαια),
- υπάρχει δομή και πλοήγηση,
- υπάρχουν πόροι (images),
- και υπάρχει τελική "έκδοση" (export).

Η επιλογή ενός ενιαίου συστήματος διαχείρισης EPUB βασίζεται στο ότι ο χρήστης χρειάζεται μια ολοκληρωμένη διαδικασία: όχι απλά να γράψει κείμενο, αλλά να οργανώσει, να διορθώσει, να μορφοποιήσει και να εξάγει. Ένα CMS-style σύστημα επιτρέπει να γίνουν όλα αυτά μέσα από ένα συνεκτικό περιβάλλον. Επίσης, διευκολύνει την εξέλιξη: μπορούν να προστεθούν επιπλέον λειτουργίες όπως templates, αυτόματη δημιουργία περιεχομένων, validation, ιστορικό εκδόσεων, ή ακόμα και συνεργατική επεξεργασία.

Όλες οι παραπάνω έννοιες επηρεάζουν άμεσα τις σχεδιαστικές επιλογές της εργασίας. Η γνώση της δομής EPUB καθορίζει:

- πώς αποθηκεύονται τα αρχεία,
- πώς ορίζεται η σειρά ανάγνωσης (spine),
- πώς γίνεται η σύνδεση CSS και πόρων,
- και πώς υλοποιείται ασφαλής εισαγωγή/εξαγωγή.

Παράλληλα, οι περιορισμοί των readers και η ανάγκη συμβατότητας καθιστούν σημαντικό να υπάρχουν μηχανισμοί που αποτρέπουν σφάλματα (π.χ. κενά XHTML, broken links). Τέλος, η CMS φιλοσοφία οδηγεί σε ένα σύστημα που δεν είναι απλώς "ένα tool", αλλά μια πλατφόρμα εργασίας για διαρκή συντήρηση και παραγωγή EPUB έργων.

Στο επόμενο κεφάλαιο θα παρουσιαστεί η ανάλυση απαιτήσεων του συστήματος: οι ρόλοι χρηστών, τα use cases, οι λειτουργικές και μη λειτουργικές απαιτήσεις, καθώς και οι στόχοι που θα καθοδηγήσουν τον σχεδιασμό και την υλοποίηση.

Κεφάλαιο 3ο: Ανάλυση απαιτήσεων

Η ανάλυση απαιτήσεων αποτελεί κρίσιμο στάδιο για την επιτυχή ανάπτυξη ενός ενιαίου συστήματος διαχείρισης περιεχομένου EPUB. Στο κεφάλαιο αυτό αποτυπώνονται οι ανάγκες που καλείται να καλύψει το σύστημα, τόσο από πλευράς λειτουργιών (τι πρέπει να κάνει) όσο και από πλευράς ποιότητας/περιορισμών (πώς πρέπει να το κάνει). Η εργασία βασίζεται σε ένα πρακτικό σενάριο χρήσης: ένας χρήστης θέλει να δημιουργεί ή να τροποποιεί EPUB βιβλία με τρόπο απλό, χωρίς να χρειάζεται να γνωρίζει όλη τη θεωρία και την "χειροκίνητη" δομή του προτύπου, αλλά ταυτόχρονα να μπορεί να επεμβαίνει στο περιεχόμενο σε επίπεδο αρχείων (XHTML/CSS/images) όταν χρειάζεται. Επιπλέον, επειδή η εφαρμογή λειτουργεί ως web πλατφόρμα, είναι απαραίτητο να οριστούν ρόλοι χρήστη, δικαιώματα πρόσβασης, τρόποι αποθήκευσης και ασφαλείς διαδικασίες εξαγωγής.

3.1 Σκοπός και όρια του συστήματος

Ο σκοπός του συστήματος είναι η παροχή ενός ενιαίου περιβάλλοντος εργασίας για EPUB, το οποίο:

- οργανώνει το περιεχόμενο ενός βιβλίου σε projects,
- επιτρέπει επεξεργασία αρχείων,
- υποστηρίζει διαχείριση πόρων (εικόνες),
- προσφέρει προεπισκόπηση,
- και επιτρέπει παραγωγή τελικών αρχείων (EPUB και PDF).

Ταυτόχρονα, το σύστημα δεν στοχεύει να αντικαταστήσει πλήρως επαγγελματικά εκδοτικά εργαλεία με πολύπλοκες ροές (π.χ. InDesign workflows) ούτε να καλύψει όλα τα edge cases του προτύπου EPUB. Ο στόχος είναι να καλύπτει ρεαλιστικές ανάγκες δημιουργίας/επιμέλειας EPUB, με έμφαση στην ευχρηστία, στην ορθότητα και στη λειτουργικότητα.

3.2 Χρήστες και ρόλοι

Το σύστημα, στη βασική του μορφή, υποστηρίζει έναν κύριο ρόλο:

3.2.1 Χρήστης (User)

Ο χρήστης:

- κάνει login/logout,

- βλέπει μόνο τα δικά του projects,
- δημιουργεί νέο project (νέο EPUB βιβλίο),
- εισάγει υπάρχον EPUB και το μετατρέπει σε project,
- επεξεργάζεται αρχεία (XHTML/CSS),
- ανεβάζει εικόνες/πόρους,
- αλλάζει τη σειρά των κεφαλαίων,
- κατεβάζει το τελικό EPUB ή PDF.

Σημαντική απαίτηση είναι ότι κάθε χρήστης πρέπει να έχει πλήρη απομόνωση δεδομένων (data isolation). Δηλαδή, να μην μπορεί να δει ή να κατεβάσει project άλλου χρήστη, ακόμα και αν γνωρίζει το project_id (security by authorization, όχι μόνο by UI).

3.3 Σενάρια χρήσης (Use Cases)

Η καταγραφή use cases βοηθά στο να οριστεί η πραγματική ροή εργασίας του συστήματος.

UC1: Είσοδος χρήστη (Login)

- Ο χρήστης εισάγει email/κωδικό.
- Το σύστημα ελέγχει τα στοιχεία και επιτρέπει πρόσβαση στο dashboard.

UC2: Προβολή λίστας έργων (Projects list)

- Μετά το login, ο χρήστης βλέπει τα projects του.
- Κάθε project έχει επιλογή “Άνοιγμα” (edit) και “Download”.

UC3: Δημιουργία νέου project

- Ο χρήστης πατά “Νέο project”.
- Το σύστημα δημιουργεί κενό έργο (ή έργο με βασικό template).
- Ο χρήστης προσθέτει αρχεία και περιεχόμενο.

UC4: Εισαγωγή υπάρχοντος EPUB (Import)

- Ο χρήστης ανεβάζει ένα .epub.

- Το σύστημα δημιουργεί νέο project.
- Γίνεται unzip, ανάγνωση δομής, αποθήκευση αρχείων και πόρων.
- Το project ανοίγει έτοιμο προς επεξεργασία.

UC5: Διαχείριση αρχείων μέσα στο project

- Ο χρήστης βλέπει αριστερά τα αρχεία.
- Επιλέγει ένα αρχείο και το επεξεργάζεται στο κέντρο.
- Με δεξί κλικ στο αρχείο μπορεί:
    - να το μετονομάσει (rename),
    - να το διπλασιάσει (duplicate),
    - να το διαγράψει (delete).
- Με drag-and-drop μπορεί να αλλάξει τη σειρά των κεφαλαίων.

UC6: Επεξεργασία XHTML/CSS με preview

- Για XHTML: δυνατότητα επιλογής “Code” (CodeMirror) ή “Rich” (Summernote).
- Σε κάθε αλλαγή ενημερώνεται το preview δεξιά.
- Το CSS εφαρμόζεται στο preview (με injection/merge).

UC7: Upload εικόνων

- Ο χρήστης ανεβάζει εικόνα σε συγκεκριμένο φάκελο (π.χ. OEBPS/images).
- Στο rich editor, οι εικόνες μπορούν να ανεβαίνουν αυτόματα με paste/insert.
- Το σύστημα εισάγει το σωστό path στο XHTML (../images/...).

UC8: Build & Download EPUB

- Ο χρήστης πατά “Download EPUB”.
- Το σύστημα δημιουργεί EPUB package.
- Η σειρά spine βασίζεται στη σειρά που έχει ορίσει ο χρήστης.
- Κατεβαίνει έτοιμο .epub.

UC9: Export PDF

- Ο χρήστης πατά “Download PDF”.
- Το σύστημα παράγει PDF από το περιεχόμενο των XHTML.
- Πρέπει να υποστηρίζονται σωστά ελληνικοί χαρακτήρες και βασική μορφοποίηση.

3.4 Λειτουργικές απαιτήσεις

Παρακάτω συγκεντρώνονται οι βασικές λειτουργικές απαιτήσεις του συστήματος:

Διαχείριση χρηστών

- FR1: Το σύστημα παρέχει login/logout.
- FR2: Κάθε χρήστης έχει πρόσβαση μόνο στα δικά του projects.

Διαχείριση projects

- FR3: Δημιουργία νέου project.
- FR4: Λίστα projects (αναζήτηση/φίλτρα μπορούν να προστεθούν).
- FR5: Import EPUB σε νέο project.

Διαχείριση αρχείων EPUB

- FR6: Προβολή λίστας αρχείων project σε panel.
- FR7: Επιλογή αρχείου και φόρτωση περιεχομένου στον editor.
- FR8: Save αλλαγών σε text αρχεία.
- FR9: Upload binary αρχείων (εικόνες).
- FR10: Rename, duplicate, delete αρχείων (με context menu).
- FR11: Drag-and-drop reorder αρχείων και αποθήκευση της σειράς.

Προεπισκόπηση (Preview)

- FR12: Live preview για XHTML.
- FR13: Το preview πρέπει να εφαρμόζει CSS αρχεία του project.
- FR14: Assets (εικόνες) πρέπει να φορτώνονται σωστά στο preview.

Παραγωγή τελικών αρχείων

- FR15: Build EPUB και download.

- FR16: Export PDF και download.

3.5 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις καθορίζουν την ποιότητα, την ασφάλεια και την πρακτικότητα του συστήματος.

Ασφάλεια

- NFR1: Authentication με ασφαλή αποθήκευση κωδικών (hash).
- NFR2: Authorization σε όλα τα endpoints (όχι μόνο σε views).
- NFR3: Ασφάλεια path (αποφυγή directory traversal σε uploads και file serving).

Απόδοση και κλιμάκωση

- NFR4: Γρήγορη φόρτωση λίστας αρχείων.
- NFR5: Αποδοτικό preview χωρίς πλήρη rebuild.
- NFR6: Αποθήκευση binary στο disk για να μην βαραίνει η DB.

Χρηστικότητα

- NFR7: UI τριών panels με σαφή λειτουργία.
- NFR8: Rich editor για χρήστες που δεν θέλουν να γράφουν HTML.
- NFR9: Context menu και drag reorder (ώστε να μοιάζει με file manager).

Αξιοπιστία / Ορθότητα

- NFR10: Το build EPUB δεν πρέπει να αποτυγχάνει λόγω "κενών" εγγράφων ή μικρών λαθών.
- NFR11: Η σειρά κεφαλαίων στο τελικό EPUB να ακολουθεί ακριβώς τη σειρά του χρήστη.
- NFR12: Το PDF export να εμφανίζει σωστά ελληνικά (font embedding).

3.6 Περιορισμοί και παραδοχές

Γίνονται ορισμένες παραδοχές που απλοποιούν την υλοποίηση χωρίς να ακυρώνουν τη χρησιμότητα του συστήματος:

- Το σύστημα στοχεύει σε EPUB3 "γενικά", χωρίς να επιβάλλει αυστηρή παραγωγή nav/toc, αν ο χρήστης δεν το θέλει.

- Η εφαρμογή επιτρέπει στον χρήστη να οργανώσει τα XHTML όπως επιθυμεί· το σύστημα εγγυάται τη σωστή σειρά και τη σωστή ενσωμάτωση στο τελικό πακέτο.
- Οι εικόνες αποθηκεύονται στο disk και αναφέρονται με relative paths.
- Το PDF export υλοποιείται με βιβλιοθήκη που μπορεί να έχει περιορισμένη CSS υποστήριξη, αλλά πρέπει να καλύπτει βασική εμφάνιση και ελληνικούς χαρακτήρες.

Στο κεφάλαιο αυτό ορίστηκαν οι χρήστες, οι βασικές λειτουργίες και τα κριτήρια ποιότητας του ενιαίου συστήματος διαχείρισης EPUB. Η καταγραφή απαιτήσεων αναδεικνύει ότι το σύστημα δεν είναι απλώς ένας editor, αλλά μια ολοκληρωμένη πλατφόρμα που καλύπτει: δημιουργία/εισαγωγή έργων, διαχείριση αρχείων και πόρων, προεπισκόπηση και τελική εξαγωγή.

Κεφάλαιο 4ο: Παρόμοια συστήματα δημιουργίας/επεξεργασίας EPUB

Είναι σημαντικό να εξεταστούν υπάρχουσες λύσεις που χρησιμοποιούνται ήδη για δημιουργία, επιμέλεια και "διόρθωση" EPUB βιβλίων. Στην πράξη, τα πιο συνηθισμένα εργαλεία χωρίζονται σε δύο μεγάλες κατηγορίες: desktop editors (εγκαθίστανται τοπικά στον υπολογιστή) και web-based πλατφόρμες (λειτουργούν μέσα από browser). Στο πλαίσιο της παρούσας εργασίας επιλέγονται δύο αντιπροσωπευτικά συστήματα, τα οποία είναι ιδιαίτερα γνωστά και χρησιμοποιούνται ευρέως: Sigil και calibre (Edit book). Η επιλογή τους δεν είναι τυχαία: και τα δύο προσφέρουν άμεση επεξεργασία "μέσα" στο EPUB (XHTML/CSS/δομή), δηλαδή πλησιάζουν λειτουργικά το αντικείμενο της εργασίας.

4.1 Sigil – Desktop EPUB Editor

Το Sigil είναι ένα δωρεάν, ανοικτού κώδικα, multi-platform εργαλείο (Windows/macOS/Linux) που έχει σχεδιαστεί ειδικά για δημιουργία και επεξεργασία EPUB. Βασική του φιλοσοφία είναι ότι ο χρήστης μπορεί να χειριστεί το EPUB ως "πακέτο" και να επεξεργαστεί τα επιμέρους στοιχεία του σε κεφάλαια σε XHTML, CSS αρχεία, καθώς και metadata/δομή. Στην επίσημη περιγραφή του τονίζεται ότι δημιουργήθηκε για να διευκολύνει την παραγωγή ποιοτικών eBooks σε μορφή EPUB [7].

4.1.1 Δυνατότητες και τρόπος χρήσης

Στο Sigil ο χρήστης βλέπει μια δομή αρχείων (σαν project) και μπορεί να επεξεργάζεται το περιεχόμενο είτε σε "κώδικα" είτε με πιο οπτικές λειτουργίες. Ειδικά για WYSIWYG επεξεργασία XHTML, το Sigil υποστηρίζει και το PageEdit, ένα εργαλείο που λειτουργεί σαν οπτικός XHTML editor. Παράλληλα, επιτρέπει εργασίες όπως οργάνωση αρχείων και πόρων, editing των HTML/CSS "innards", και γενικότερα διαχείριση του EPUB bundle.

4.1.2 Πλεονεκτήματα

- **Πλήρης έλεγχος πάνω στη δομή EPUB**: ο χρήστης μπορεί να πειράξει άμεσα XHTML/CSS και πόρους.
- **Εστίαση αποκλειστικά στο EPUB**: είναι editor "για EPUB", όχι γενικό εργαλείο.

 Κατάλληλο για επαγγελματική επιμέλεια: υποστηρίζει workflow όπου ο χρήστης γνωρίζει (ή μαθαίνει) τη δομή του προτύπου.

4.1.3 Μειονεκτήματα / περιορισμοί σε σχέση με το σύστημα της εργασίας

Παρότι το Sigil είναι ισχυρό, ως desktop εργαλείο έχει συγκεκριμένους περιορισμούς:

- Η πρόσβαση είναι **δεμένη στον υπολογιστή** και στην εγκατάσταση του εργαλείου.
- Δεν υπάρχει "λογική χρήστη/λογαριασμού" με projects στο cloud και απομόνωση ανά χρήστη (multi-user web περιβάλλον).
- Δεν προσφέρει με τον ίδιο τρόπο ένα **ενιαίο web workflow** (login → projects → edit → export) που να είναι διαθέσιμο από παντού.
- Λειτουργίες όπως **ενσωματωμένο upload εικόνας από rich editor** (paste/insert → upload → εισαγωγή στο XHTML) απαιτούν συνήθως χειροκίνητη διαχείριση αρχείων, ανάλογα με τη ροή εργασίας.

Συνολικά, το Sigil αποτελεί ένα εξαιρετικό "εργαλείο επιμέλειας", αλλά προϋποθέτει ότι ο χρήστης είναι άνετος με την ιδέα του EPUB ως αρχειακού πακέτου και με desktop περιβάλλον.

4.2 calibre – Integrated "Edit book" editor

Το **calibre** είναι γνωστό κυρίως ως ολοκληρωμένη λύση διαχείρισης e-book βιβλιοθήκης, μετατροπών format και οργάνωσης συλλογών. Ωστόσο, διαθέτει και ενσωματωμένο editor (Edit book), ο οποίος μπορεί να χρησιμοποιηθεί για επεξεργασία βιβλίων σε EPUB (και άλλες μορφές όπως AZW3/KEPUB) [8].

4.2.1 Δυνατότητες του calibre editor

Σύμφωνα με την επίσημη τεκμηρίωση, ο calibre editor εμφανίζει το HTML/CSS του βιβλίου και παρέχει live preview που ενημερώνεται καθώς ο χρήστης κάνει αλλαγές. Επιπλέον, περιλαμβάνει εργαλεία αυτοματοποίησης για "cleanup/fixing tasks", δηλαδή για συχνές

διορθώσεις/βελτιστοποιήσεις. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη όταν ο στόχος είναι να διορθωθεί ένα ήδη υπάρχον EPUB, να καθαριστεί η δομή ή να γίνουν μαζικές αλλαγές.

4.2.2 Πλεονεκτήματα

- Εύκολη πρόσβαση σε editor μέσα από τη βιβλιοθήκη (right click → Edit book).
- Live preview κατά την επεξεργασία HTML/CSS, κάτι που είναι κρίσιμο για τεχνικές μορφοποιήσεις.
- Εργαλεία αυτόματης διόρθωσης για κοινά ζητήματα (καθαρισμός, έλεγχοι, κ.ά.).

4.2.3 Μειονεκτήματα

Το calibre είναι εξαιρετικό ως "εργαλείο βιβλιοθήκης", όμως ο editor του δεν έχει σχεδιαστεί ως web CMS:

- Δεν παρέχει web-based διαχείριση έργων ανά χρήστη (login/projects) — λειτουργεί τοπικά.
- Η διαχείριση αρχείων/πόρων γίνεται στο πλαίσιο της βιβλιοθήκης calibre και όχι ως "εφαρμογή έργου" που μπορεί να χρησιμοποιηθεί από πολλούς χρήστες σε έναν server.
- Δεν προσφέρει εγγενώς ένα περιβάλλον τριών panels (file manager → editor → preview) προσαρμοσμένο σε workflow δημιουργίας EPUB με "projects", upload assets, drag ordering και export μέσα από browser.
- Δεν ενσωματώνει "rich editor" τύπου Summernote για χρήστες που θέλουν οπτική συγγραφή, καθώς είναι κατά βάση code-oriented editor.

4.3 Συγκριτική θεώρηση με το προτεινόμενο σύστημα

4.3.1 Μοντέλο λειτουργίας

- **Sigil / calibre**: desktop εφαρμογές → ο χρήστης δουλεύει τοπικά, με αρχεία στον υπολογιστή του.
- **Προτεινόμενο σύστημα**: web εφαρμογή → ο χρήστης μπαίνει με login, βλέπει projects, επεξεργάζεται μέσω browser και εξάγει (EPUB/PDF).

4.3.2 Διαχείριση έργων (projects)

 Sigil / calibre: δεν υπάρχει "ενιαίο dashboard" ανά χρήστη σε server, ούτε πολυχρηστική πρόσβαση.

 Προτεινόμενο σύστημα: κάθε χρήστης έχει τα δικά του projects, με απομόνωση δεδομένων και οργανωμένη αποθήκευση.

4.3.3 Διαχείριση αρχείων με εργονομία CMS

Τα desktop tools έχουν file views, αλλά η εργασία στοχεύει σε UI που μοιάζει με file manager:

- δεξί κλικ (rename/duplicate/delete),
- drag-and-drop reorder που καθορίζει spine order,
- άμεσο preview στο πλάι, προσαρμοσμένο σε "κεφάλαιο-κεντρικό" workflow.

4.3.4 Εστίαση στη σειρά ανάγνωσης (spine)

Το Sigil και ο calibre editor παρέχουν τρόπους να ρυθμιστεί η δομή, αλλά η εργασία επιλέγει να "καθρεφτίζει" την επιθυμητή σειρά του χρήστη με απλό τρόπο: η σειρά που βλέπει ο χρήστης στη λίστα = η σειρά build του EPUB, χωρίς να απαιτείται χειροκίνητο editing OPF.

4.3.5 Rich editing και αυτόματο upload εικόνων

Τα desktop εργαλεία μπορούν να διαχειριστούν εικόνες, όμως η εργασία δίνει έμφαση σε ένα workflow τύπου "γράψε περιεχόμενο → βάλε εικόνα → ανέβασε αυτόματα → μπήκε στο XHTML", κάτι που ταιριάζει πολύ σε εκπαιδευτικά σενάρια και γρήγορη συγγραφή μέσω web.

Τα συστήματα Sigil και calibre editor αποτελούν ισχυρές και δοκιμασμένες λύσεις για δημιουργία/επεξεργασία EPUB, με ιδιαίτερη αξία σε χρήστες που εργάζονται τοπικά και θέλουν άμεσο έλεγχο XHTML/CSS και δομής. Παρόλα αυτά, το κοινό τους χαρακτηριστικό είναι ότι δεν λειτουργούν ως ενιαία web πλατφόρμα διαχείρισης έργων με login, multi-project workflow, αποθήκευση σε server, και UI προσανατολισμένο σε CMS λειτουργίες (context actions, drag ordering, rich editing, export από browser). Αυτή ακριβώς η "κενή περιοχή" αποτελεί τον βασικό χώρο στον οποίο τοποθετείται η

παρούσα πτυχιακή όπου είναι η ανάπτυξη ενός συστήματος που γεφυρώνει το κενό ανάμεσα στην τεχνική επεξεργασία EPUB και στην εργονομία/ευκολία μιας σύγχρονης web πλατφόρμας.

Εκτός από τα desktop εργαλεία (Sigil, calibre), στην αγορά υπάρχουν και web-based πλατφόρμες που υποστηρίζουν συγγραφή, μορφοποίηση και εξαγωγή βιβλίων σε EPUB. Αυτές οι λύσεις είναι ιδιαίτερα χρήσιμες όταν ο στόχος είναι η εργασία "από παντού" μέσω browser, ή όταν χρειάζεται πιο "εκδοτική" ροή εργασίας (θέματα, templates, online publishing). Παρακάτω γίνεται σύντομη αναφορά σε τρία χαρακτηριστικά παραδείγματα, ώστε να τοποθετηθεί καλύτερα η παρούσα εργασία στο οικοσύστημα εργαλείων.

4.3.6 Pressbooks (web-based book CMS)

Το Pressbooks περιγράφεται ως "book content management system" βασισμένο στο WordPress, το οποίο επιτρέπει δημοσίευση στο web και παραγωγή εξαγωγών σε πολλαπλές μορφές, όπως EPUB και PDF [9].

Η πλατφόρμα παρέχει workflow τύπου "γράψε περιεχόμενο → εξαγωγή", με δυνατότητα δημιουργίας και κατεβάσματος αρχείου EPUB μέσα από το dashboard της εφαρμογής.

Πού βοηθά: Είναι ισχυρή επιλογή όταν ζητείται εκδοτική διαδικασία και παραγωγή σε πολλές μορφές, ειδικά για εκπαιδευτικές εκδόσεις/OER.
Πού διαφέρει από το δικό μας σύστημα: Η παρούσα εργασία στοχεύει περισσότερο σε *file-level* έλεγχο και "εργαστήριο EPUB" (διαχείριση αρχείων, reorder κεφαλαίων, rich editor + code editor, import οποιουδήποτε EPUB για επιμέλεια), ενώ το Pressbooks είναι περισσότερο μια ολοκληρωμένη πλατφόρμα συγγραφής/έκδοσης με δικό της μοντέλο περιεχομένου.

4.3.7 Reedsy Studio (online book editor/formatter)

Το Reedsy Studio είναι online εργαλείο μορφοποίησης/συγγραφής που δίνει εξαγωγή σε EPUB, με στόχο ένα "reflowable EPUB 3" συμβατό με διανομείς/retailers [10]. Παρέχει επίσης διαδικασία export μέσα από την πλατφόρμα, όπου ο χρήστης επιλέγει ρυθμίσεις και παράγει τα τελικά αρχεία.

Πού βοηθά: Όταν ο στόχος είναι "γρήγορη παραγωγή" ebook με όμορφα themes χωρίς να αγγίξει κανείς XHTML/CSS.

Πού διαφέρει από το δικό μας σύστημα: Εδώ, το ζητούμενο της εργασίας είναι να δίνεται στον χρήστη άμεση πρόσβαση στα αρχεία του EPUB (XHTML/CSS/images), δυνατότητα import "όπως είναι" και επεξεργασία σε επίπεδο αρχείων, κάτι που δεν είναι το βασικό σενάριο των online formatters.

4.3.8 Book Creator (online authoring, εκπαιδευτική χρήση)

Το Book Creator χρησιμοποιείται πολύ σε εκπαιδευτικά σενάρια για δημιουργία ψηφιακών βιβλίων και υποστηρίζει εξαγωγή σε ePub (και PDF) [11].

Η λογική του είναι πιο "οπτική" και φιλική, με έμφαση στη δημιουργία περιεχομένου από μη τεχνικούς χρήστες. Πολύ καλό για σχολεία/εκπαίδευση και για δημιουργία βιβλίων με εύκολη διεπαφή. Πού διαφέρει από το δικό μας σύστημα: Το δικό μας σύστημα έχει στόχο να λειτουργεί ως "ενιαίο περιβάλλον διαχείρισης EPUB αρχείων" με code editing, file operations (rename/duplicate/delete), drag reorder και import/επιμέλεια ενός υπάρχοντος EPUB — δηλαδή πιο τεχνικό, αλλά και πιο ελεγχόμενο, κατάλληλο για workflow τύπου "developer/editor".

Οι web πλατφόρμες όπως Pressbooks, Reedsy Studio και Book Creator δείχνουν ότι υπάρχει έντονη ανάγκη για online συγγραφή και εξαγωγή EPUB, όμως συχνά στοχεύουν είτε σε εκδοτική παραγωγή μέσω templates είτε σε οπτική δημιουργία περιεχομένου. Η προσέγγιση της παρούσας πτυχιακής διαφοροποιείται γιατί δίνει προτεραιότητα στο μοντέλο "έργο = EPUB project" με διαχείριση αρχείων, άμεση επεξεργασία XHTML/CSS, import οποιουδήποτε EPUB για επιμέλεια, και σειρά ανάγνωσης (spine) μέσω drag-and-drop—χαρακτηριστικά που στοχεύουν σε ένα πιο "εργαστηριακό" και ελεγχόμενο περιβάλλον παραγωγής EPUB.

Κεφάλαιο 5ο: Σχεδίαση Συστήματος

Στο κεφάλαιο αυτό παρουσιάζεται η σχεδίαση του "Ενιαίου συστήματος διαχείρισης περιεχομένου EPUB" με τρόπο που να συνδέει άμεσα τις απαιτήσεις του προηγούμενου κεφαλαίου με συγκεκριμένες τεχνικές επιλογές. Η σχεδίαση δεν αφορά μόνο την επιλογή τεχνολογιών, αλλά κυρίως την οργάνωση των δεδομένων, τη ροή εργασίας χρήστη, την ασφάλεια πρόσβασης, τη διαχείριση αρχείων (κειμένου και binary), και τον τρόπο με τον οποίο το σύστημα "χτίζει" τελικά αρχεία EPUB/PDF. Η βασική φιλοσοφία του συστήματος είναι ότι κάθε βιβλίο αντιμετωπίζεται ως έργο (project), και το έργο αποτελείται από αρχεία (XHTML, CSS, εικόνες, κ.λπ.). Ο χρήστης δεν αναγκάζεται να γνωρίζει τη θεωρία του EPUB (manifest/spine/toc), αλλά το σύστημα παρέχει τα εργαλεία ώστε, αν χρειάζεται, να έχει και λεπτομερή έλεγχο.

5.1 Γενική αρχιτεκτονική (High-level Architecture)

Το σύστημα υλοποιείται ως web εφαρμογή τύπου client–server:

- **Frontend (Browser UI)**

  Παρουσιάζει το περιβάλλον εργασίας τριών panels:

  1. αριστερό panel: λίστα αρχείων και διαχείριση (rename/duplicate/delete + drag reorder)
  2. κεντρικό panel: editor (CodeMirror + Summernote με tabs)
  3. δεξί panel: live preview για XHTML και preview εικόνων για binary
     Το frontend επικοινωνεί με το backend μέσω HTTP requests (REST-like endpoints).

- **Backend (Python/Flask)**

  Διαχειρίζεται authentication, projects, files, uploads, import, export/build. Παρέχει endpoints για:

  - ανάκτηση λίστας αρχείων
  - ανάγνωση/αποθήκευση αρχείων
  - actions (rename/duplicate/delete)
  - reorder (sort_order)
  - upload εικόνων
  - build/export EPUB
  - export PDF

- **Database (MySQL)**

  Αποθηκεύει χρήστες, projects και text αρχεία (XHTML/CSS/metadata). Επιπλέον αποθηκεύει "σειρά" (sort_order) ώστε η σειρά που βλέπει ο χρήστης να είναι η σειρά build στο EPUB [12].

- **Disk Storage (uploads folder)**

  Αποθηκεύει binary αρχεία (εικόνες, fonts, κ.λπ.) σε δομή: uploads/<project_id>/<path μέσα στο epub>
  Έτσι το σύστημα μπορεί να σερβίρει εικόνες στο preview και να τις ενσωματώνει στο export.

Η επιλογή υβριδικής αποθήκευσης (DB για text, disk για binary) προκύπτει από ανάγκες απόδοσης και πρακτικότητας: οι εικόνες είναι μεγάλες, ενώ τα XHTML/CSS είναι εύκολα να γίνουν query, edit και versioned.

5.2 Μοντέλο δεδομένων και σχεδίαση βάσης (MySQL)

5.2.1 Οντότητες και σχέσεις

User

- Αντιπροσωπεύει τον χρήστη της εφαρμογής (login).
- Σχέση 1–N με projects.

EpubProject

- Αντιπροσωπεύει ένα βιβλίο/έργο.
- Περιέχει metadata (title, language, identifier).
- Σχέση 1–N με files.

EpubFile

- Αντιπροσωπεύει ένα αρχείο του βιβλίου (XHTML, CSS, εικόνα, κ.λπ.).
- Κύρια πεδία:
  - path: πλήρης εσωτερική διαδρομή μέσα στο epub (π.χ. OEBPS/text/ch1.xhtml)
  - content: κείμενο (μόνο για text αρχεία)
  - mime: mime type
  - is_binary: δηλώνει αν είναι binary (εικόνα κ.λπ.)

- is_editable: αν επιτρέπεται edit
- sort_order: αριθμητική σειρά εμφάνισης/παραγωγής

Γιατί χρειάζεται sort_order

Το sort_order είναι ο κρίσιμος μηχανισμός για να υλοποιηθεί η απαίτηση:

"Ο χρήστης κάνει drag τα xhtml στη σειρά που θέλει και το build ακολουθεί αυτή τη σειρά."

Το σύστημα δεν βασίζεται σε "toc.xhtml" ή "index.xhtml" ως υποχρεωτικά. Ο χρήστης μπορεί να τα έχει ή να μην τα έχει. Το μόνο που ενδιαφέρει για το spine είναι:

- ποια XHTML θεωρούνται κεφάλαια
- με ποια σειρά θα μπουν

Η sort_order αποθηκεύεται για **όλα** τα αρχεία ώστε να εμφανίζονται πάντα με συνέπεια στη λίστα, όχι αλφαβητικά.

5.3 Σχεδίαση διεπαφής χρήστη (UI Design)

Πίνακας "Projects"

Μετά το login ο χρήστης βλέπει:

- λίστα έργων του (με τίτλο, id, γλώσσα)
- κουμπί "Νέο project"
- φόρμα "Import EPUB" (ανεβάζει .epub → δημιουργείται νέο project)
- επιλογή "Άνοιγμα" (editor) και "Download" (export)

Η ύπαρξη dashboard είναι βασική CMS λογική: ο χρήστης δεν δουλεύει σε "ένα βιβλίο", αλλά σε "πολλά έργα".

Editor: τρία panels

Αριστερό panel (File Manager)

- εμφανίζει τα αρχεία **με filename μόνο** (χωρίς OEBPS/...)
- δεξί κλικ πάνω σε αρχείο:
  - rename
  - duplicate
  - delete

- drag & drop:
    - αλλάζει τη σειρά και στέλνει στο backend νέο ordering

Κεντρικό panel (Editor)

- CodeMirror για κώδικα (XHTML/CSS/XML)
- Summernote ως rich editor για XHTML, μέσω tabs:
    - Tab “Code”: πλήρης έλεγχος
    - Tab “Rich”: οπτική συγγραφή, insert εικόνας με αυτόματο upload

Δεξί panel (Preview)

- iframe με srcdoc για live preview XHTML
- injection CSS στο preview
- rewrite σχετικών paths (εικόνες) σε /u/<project>/<path> ώστε να φορτώνουν από τον server
- για binary εικόνες: άμεσο image preview

5.4 Σχεδίαση API και endpoints (Backend Interface)

Παρακάτω περιγράφονται οι βασικές κατηγορίες endpoints και ο ρόλος τους.

Authentication endpoints

- GET /login (φόρμα)
- POST /login (έλεγχος credentials)
- POST /logout (τερματισμός session)

Projects endpoints

- GET /projects (λίστα έργων)
- POST /projects/new (δημιουργία)
- POST /projects/import-epub (import από uploaded epub)
- GET /project/<id> (editor view)

Files endpoints

- GET /api/project/<id>/files
  επιστρέφει λίστα αρχείων ταξινομημένη με sort_order
- GET /api/file/<id>
  επιστρέφει το περιεχόμενο (ή metadata αν binary)
- POST /api/file/<id>
  αποθήκευση περιεχομένου (μόνο για text)

- POST /api/project/<id>/files
  δημιουργία νέου αρχείου

Actions endpoints (δεξί κλικ)

- POST /api/file/<id>/rename
- POST /api/file/<id>/duplicate
- POST /api/file/<id>/delete

Η σχεδίαση χωρίζει το “edit content” από τις “ενέργειες διαχείρισης”, για καθαρή λογική και έλεγχο.

Reorder endpoint

- POST /api/project/<id>/reorder
  στέλνει νέο ordering (λίστα file_id με σειρά) και ενημερώνει sort_order.

Η λογική reorder είναι κρίσιμη γιατί επηρεάζει:

- UI σειρά
- build spine σειρά

Upload & serve assets

- POST /api/project/<id>/upload
  ανεβάζει αρχείο σε target folder π.χ. OEBPS/images
- GET /u/<project_id>/<path>
  σερβίρει binary αρχείο από disk (μόνο αν το project ανήκει στον χρήστη)

Export endpoints

- GET /api/project/<id>/download → build EPUB
- GET /api/project/<id>/download.pdf → export PDF

5.5 Ασφάλεια και έλεγχος πρόσβασης (Security by design)

Ασφάλεια και έλεγχος πρόσβασης (Security by design)

Η εφαρμογή πρέπει να εφαρμόζει ασφάλεια σε κάθε endpoint, όχι μόνο στη διεπαφή.

Authorization

Κάθε κλήση που αφορά project/file πρέπει να ελέγχει:

- ότι ο χρήστης είναι logged in
- ότι το project ανήκει στον χρήστη
- ότι το file ανήκει σε project του χρήστη

Αυτό αποτρέπει πρόσβαση με “μαντεμένα ids”.

Safe paths (αποφυγή directory traversal)

Το path ενός αρχείου μέσα στο epub μπορεί να χρησιμοποιηθεί για αποθήκευση σε disk. Άρα πρέπει να αποτρέπεται:

- ../ sequences
- absolute paths
- windows backslashes
- μη επιτρεπτά πολύ μεγάλα paths

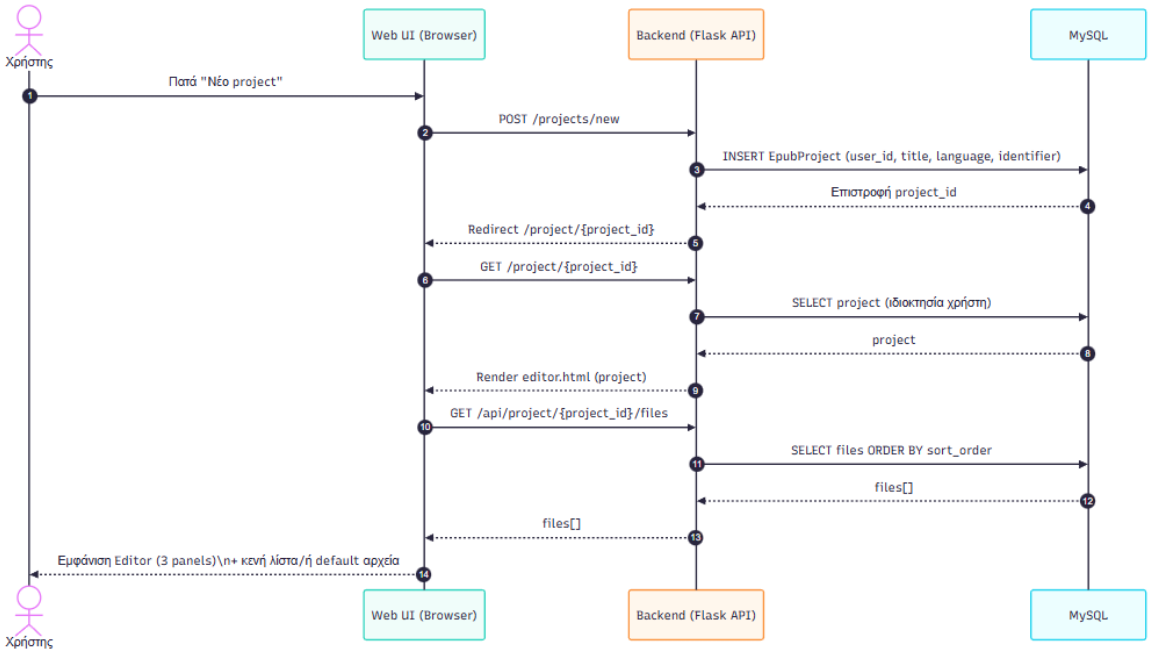
Η σχεδίαση περιλαμβάνει helper safe_epub_path() που κανονικοποιεί σε POSIX μορφή και απορρίπτει traversal.

Περιορισμοί επεξεργασίας

- binary files δεν επιτρέπεται να σώζονται ως text
- is_editable = false για αρχεία που θεωρούνται “system” ή προστατευμένα (αν το θέλουμε)

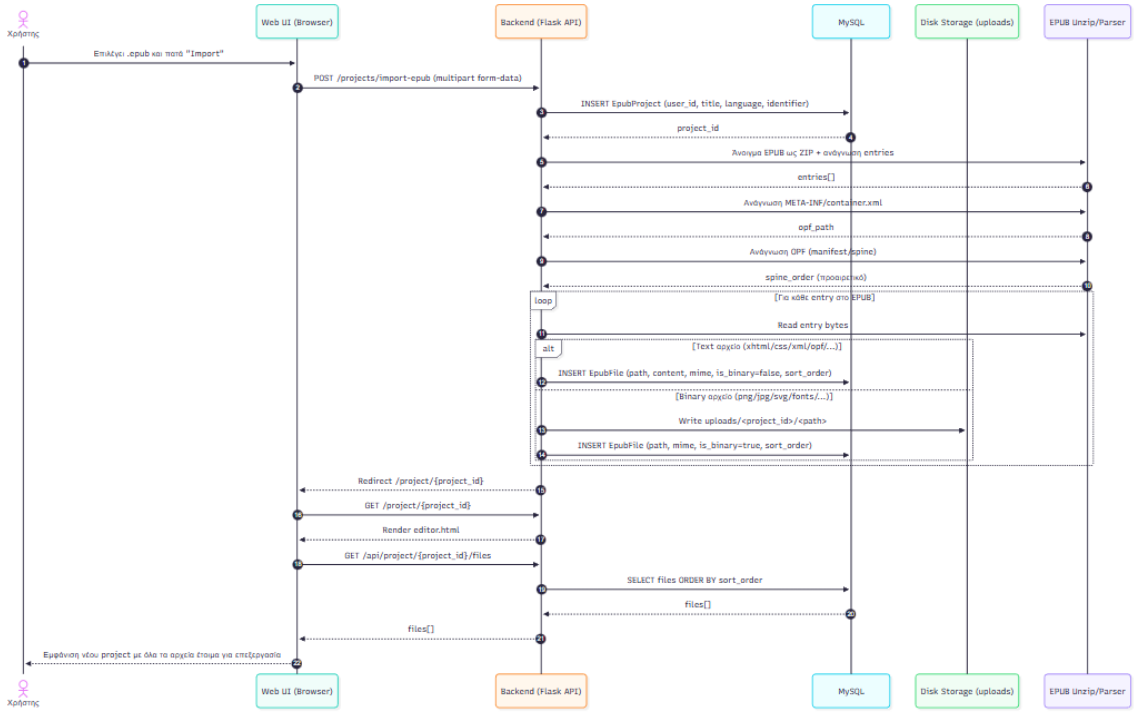
5.6 Σχεδίαση API και endpoints (Backend Interface)

Η ροή ξεκινά όταν ο χρήστης πατά «Νέο project» στο Web UI. Το backend δημιουργεί εγγραφή EpubProject στη MySQL και επιστρέφει ανακατεύθυνση στο /project/{project_id}. Στη συνέχεια το UI φορτώνει τη σελίδα του editor και ζητά από το API τη λίστα αρχείων του project, ώστε να εμφανιστεί το περιβάλλον εργασίας των τριών panels, όπως φαίνεται στην Εικόνα 5.1.

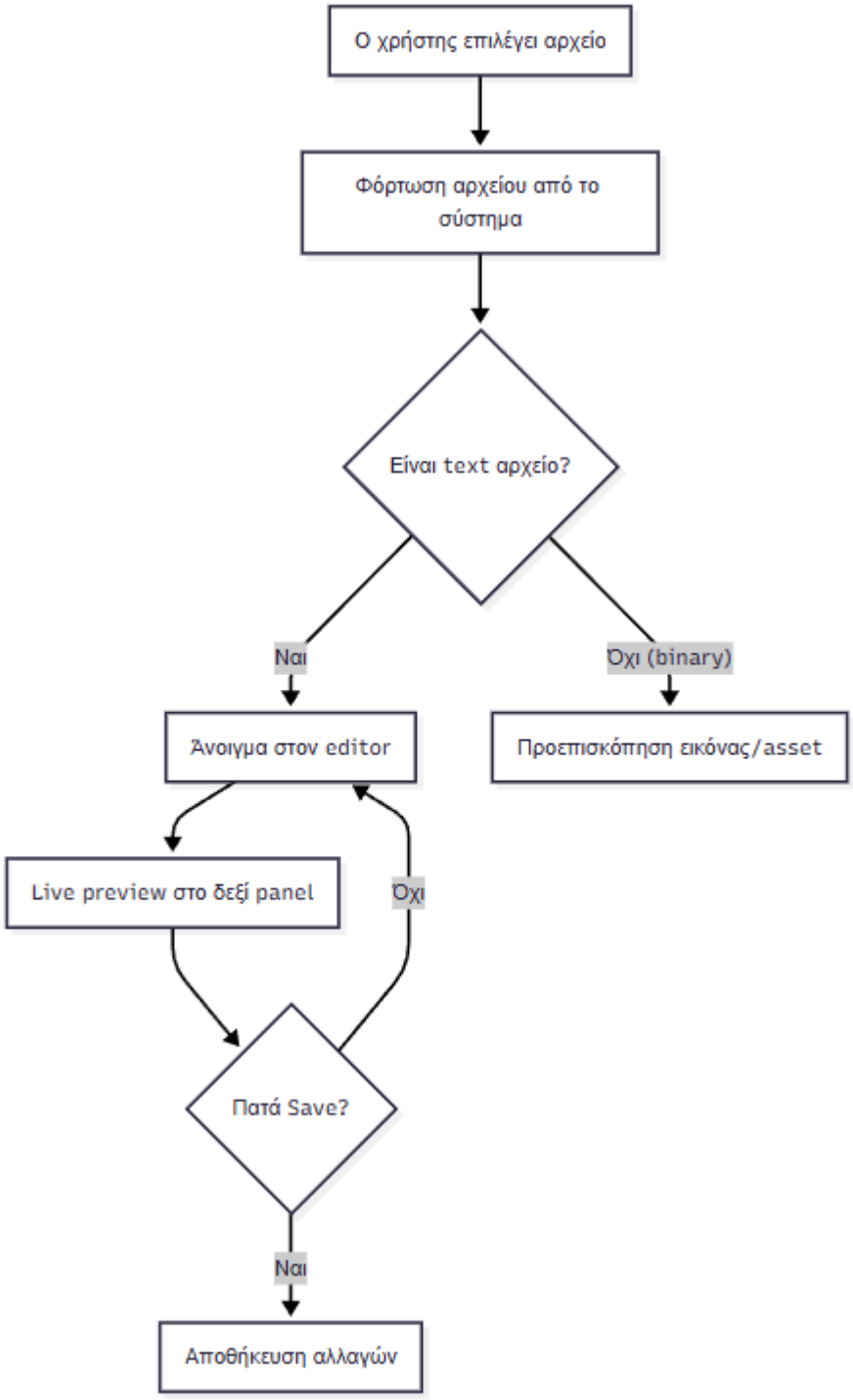


Εικόνα 5.1: Διάγραμμα ακολουθίας που περιγράφει τα βήματα δημιουργίας νέου έργου EPUB και μετάβασης στο περιβάλλον επεξεργασίας.

Στην Εικόνα 5.2 παρουσιάζεται η διαδικασία εισαγωγής ενός αρχείου .epub από τον χρήστη. Το Web UI στέλνει το αρχείο στο backend, το οποίο δημιουργεί νέο EpubProject στη MySQL και αποσυμπιέζει το EPUB σε προσωρινό χώρο. Στη συνέχεια, για κάθε αρχείο του πακέτου αποφασίζεται αν είναι text (XHTML/CSS/XML) ή binary (εικόνες). Τα text αρχεία αποθηκεύονται στη βάση, ενώ τα binary γράφονται στον δίσκο στον φάκελο uploads/<project_id>/.... Προαιρετικά, γίνεται ανάγνωση του OPF ώστε να εξαχθεί αρχική σειρά κεφαλαίων (spine) και να ενημερωθεί το sort_order. Τέλος, το σύστημα ανακατευθύνει τον χρήστη στο editor του νέου project και φορτώνεται η λίστα αρχείων.

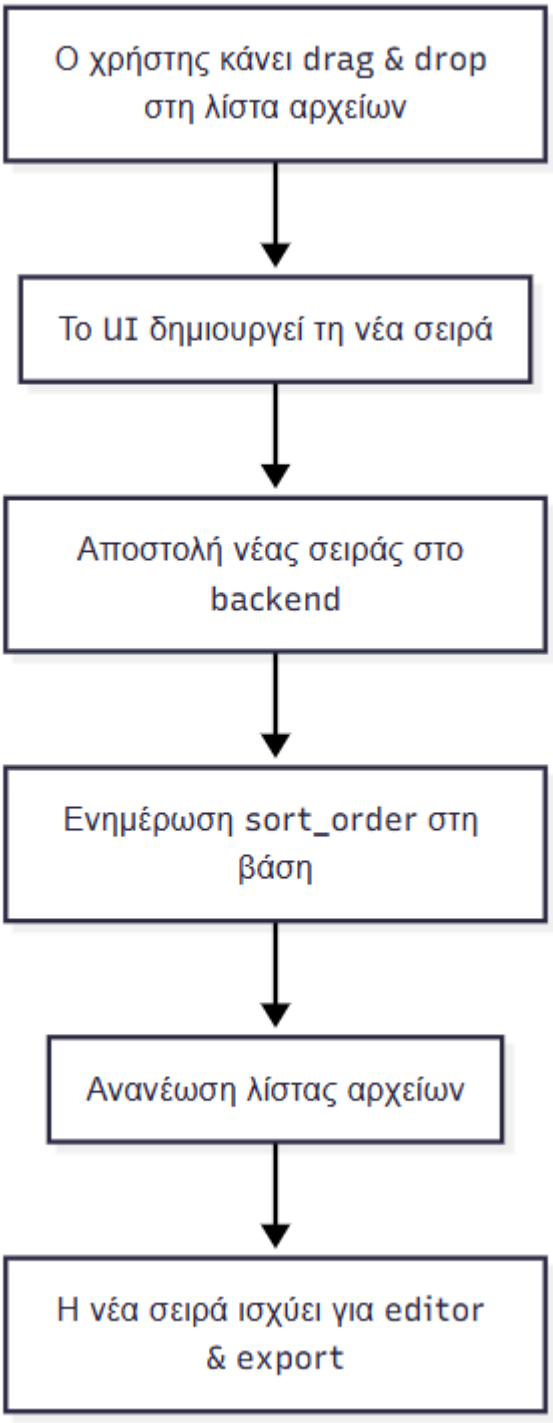


Εικόνα 5.2: Διάγραμμα ακολουθίας εισαγωγής υπάρχοντος EPUB και δημιουργίας νέου project



Εικόνα 5.3: Διάγραμμα ροής επεξεργασίας αρχείου και live preview

Στην Εικόνα 5.3 παρουσιάζεται συνοπτικά η ροή από την επιλογή ενός αρχείου μέχρι την προεπισκόπηση και την αποθήκευση των αλλαγών.



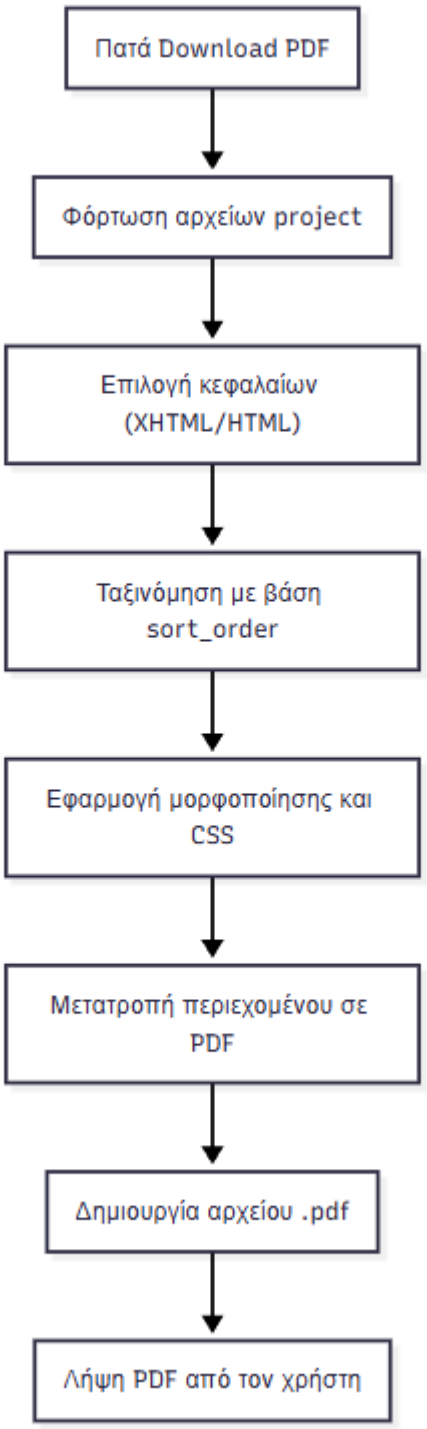
Εικόνα 5.4: Διάγραμμα ροής αλλαγής σειράς αρχείων με drag & drop

Στην Εικόνα 5.4 παρουσιάζεται η διαδικασία αλλαγής της σειράς των αρχείων μέσα σε ένα project μέσω drag & drop. Αρχικά, ο χρήστης μετακινεί ένα ή περισσότερα αρχεία στη λίστα του αριστερού panel, ώστε να ορίσει τη σειρά που επιθυμεί να εμφανίζονται και να "διαβάζονται" τα κεφάλαια. Το Web UI καταγράφει τη νέα σειρά (συνήθως ως λίστα από IDs ή paths) και τη στέλνει στο backend με ένα αίτημα reorder. Το backend ενημερώνει τη βάση δεδομένων, τροποποιώντας το πεδίο sort_order για κάθε αρχείο με βάση τη νέα θέση του στη λίστα. Στη συνέχεια, το UI ανανεώνει τη λίστα αρχείων

ώστε να εμφανίζεται σταθερά η νέα ταξινόμηση. Από εκεί και πέρα, η αποθηκευμένη σειρά θεωρείται “πηγή αλήθειας” και χρησιμοποιείται τόσο για την προβολή των αρχείων στο editor όσο και για τη δημιουργία του τελικού EPUB/PDF, εξασφαλίζοντας ότι το export ακολουθεί ακριβώς την σειρά που επέλεξε ο χρήστης.

Εικόνα 5.5: Διάγραμμα ροής δημιουργίας και λήψης EPUB

Στην Εικόνα 5.5 παρουσιάζεται συνοπτικά η διαδικασία παραγωγής (build) και λήψης ενός αρχείου EPUB από το σύστημα. Ο χρήστης, ενώ βρίσκεται στο περιβάλλον επεξεργασίας του project, επιλέγει την ενέργεια “Download EPUB”. Το σύστημα ξεκινά τη δημιουργία του τελικού πακέτου EPUB συλλέγοντας όλα τα αρχεία του έργου. Αρχικά, γίνεται φόρτωση των αρχείων από τη βάση δεδομένων και/ή τον δίσκο, με ταξινόμηση βάσει του sort_order, ώστε να διατηρηθεί η σειρά κεφαλαίων όπως την έχει ορίσει ο χρήστης. Στη συνέχεια δημιουργείται η δομή του EPUB (συμπεριλαμβανομένων των απαραίτητων αρχείων container και package), και ενσωματώνονται τα περιεχόμενα XHTML, τα CSS αρχεία και τα binary assets (όπως εικόνες). Το “spine” του βιβλίου διαμορφώνεται σύμφωνα με τη σειρά των XHTML αρχείων (όπως προκύπτει από το sort_order), χωρίς να απαιτείται να υπάρχουν υποχρεωτικά συγκεκριμένες σελίδες (π.χ. toc ή index). Τέλος, το σύστημα συμπιέζει το πακέτο σε μορφή .epub και το επιστρέφει στον browser ως αρχείο προς λήψη, ολοκληρώνοντας την εξαγωγή του βιβλίου.

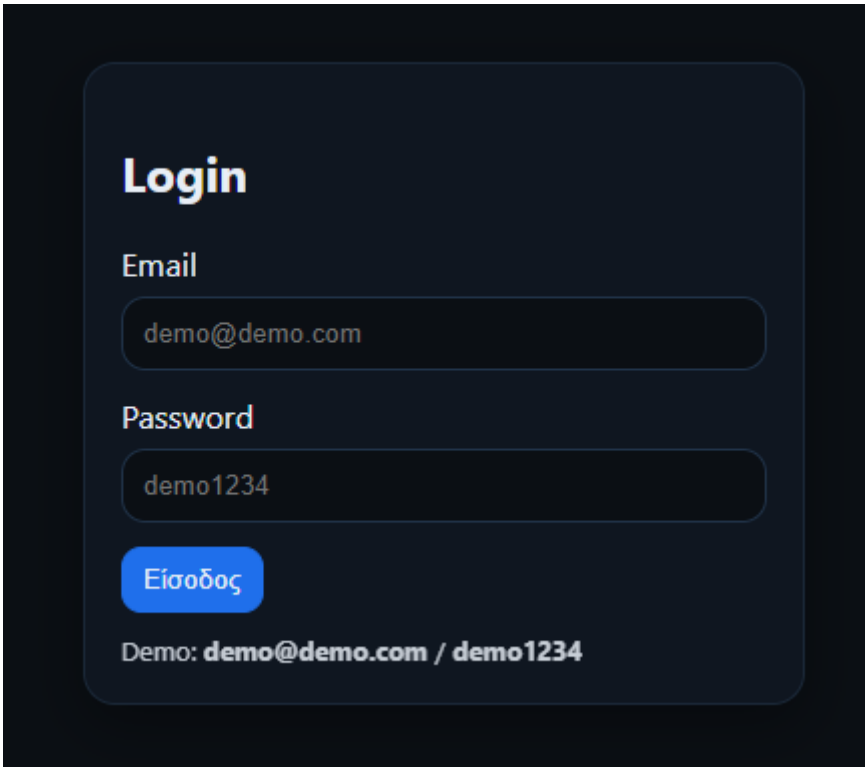


Εικόνα 5.6: Διάγραμμα ροής δημιουργίας και λήψης EPUB

Στην Εικόνα 5.6 παρουσιάζεται η διαδικασία με την οποία ο χρήστης εξάγει το βιβλίο σε μορφή PDF. Αρχικά, ο χρήστης επιλέγει την ενέργεια “Download PDF” από το περιβάλλον του project. Το σύστημα φορτώνει τα αρχεία του έργου και επιλέγει τα XHTML/HTML κεφάλαια που θα συμπεριληφθούν, ακολουθώντας τη σειρά που έχει οριστεί από το sort_order. Στη συνέχεια, το περιεχόμενο οργανώνεται

σε ενιαία ροή (π.χ. συγχώνευση κεφαλαίων) και εφαρμόζονται βασικοί κανόνες μορφοποίησης μαζί με τα CSS του project, ώστε το τελικό αποτέλεσμα να είναι αναγνώσιμο και συνεπές. Έπειτα, γίνεται μετατροπή σε PDF μέσω της αντίστοιχης βιβλιοθήκης παραγωγής PDF, με μέριμνα για σωστή απόδοση χαρακτήρων (π.χ. ελληνικών) μέσω κατάλληλων γραμματοσειρών. Τέλος, το παραγόμενο αρχείο PDF επιστρέφεται στον browser και ο χρήστης το κατεβάζει στον υπολογιστή του.

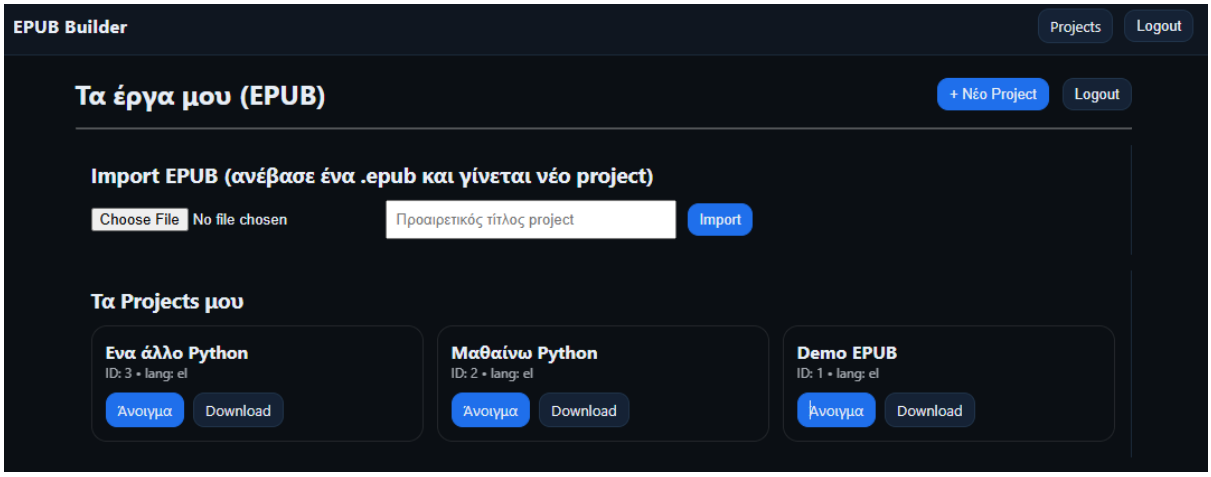
5.7 Πλοήγηση στην πλατφόρμα



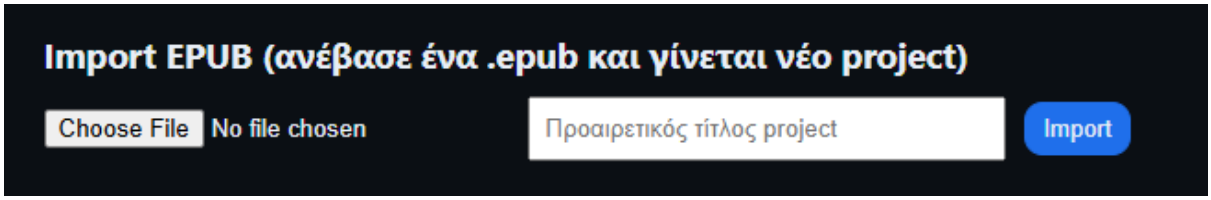
Εικόνα 5.7: Σελίδα για σύνδεση χρήστη

Στην **Εικόνα 5.7** παρουσιάζεται η σελίδα σύνδεσης χρήστη, όπου ο χρήστης εισάγει τα στοιχεία του (π.χ. email και κωδικό) για να αποκτήσει πρόσβαση στην πλατφόρμα και στις λειτουργίες διαχείρισης EPUB έργων.

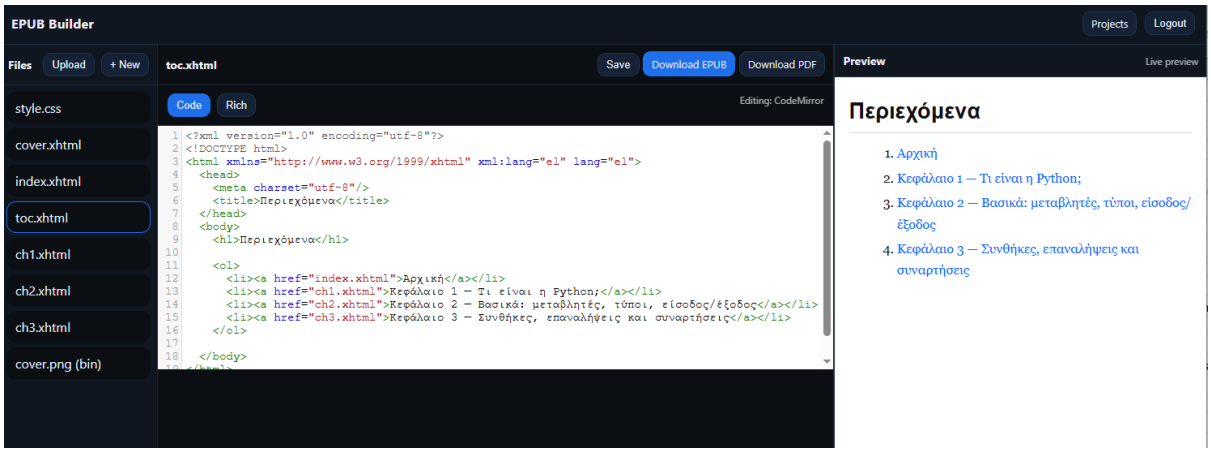
Στην **Εικόνα 5.8** παρουσιάζεται η κεντρική σελίδα “Τα έργα μου”, στην οποία εμφανίζονται συγκεντρωμένα όλα τα projects του χρήστη, καθώς και η δυνατότητα δημιουργίας νέου έργου ή εισαγωγής (upload) ενός έτοιμου EPUB για μετατροπή του σε επεξεργάσιμο project.



Εικόνα 5.8: Κεντρική Σελίδα με τα έργα μου και επιλογή ανεβάσματος epub

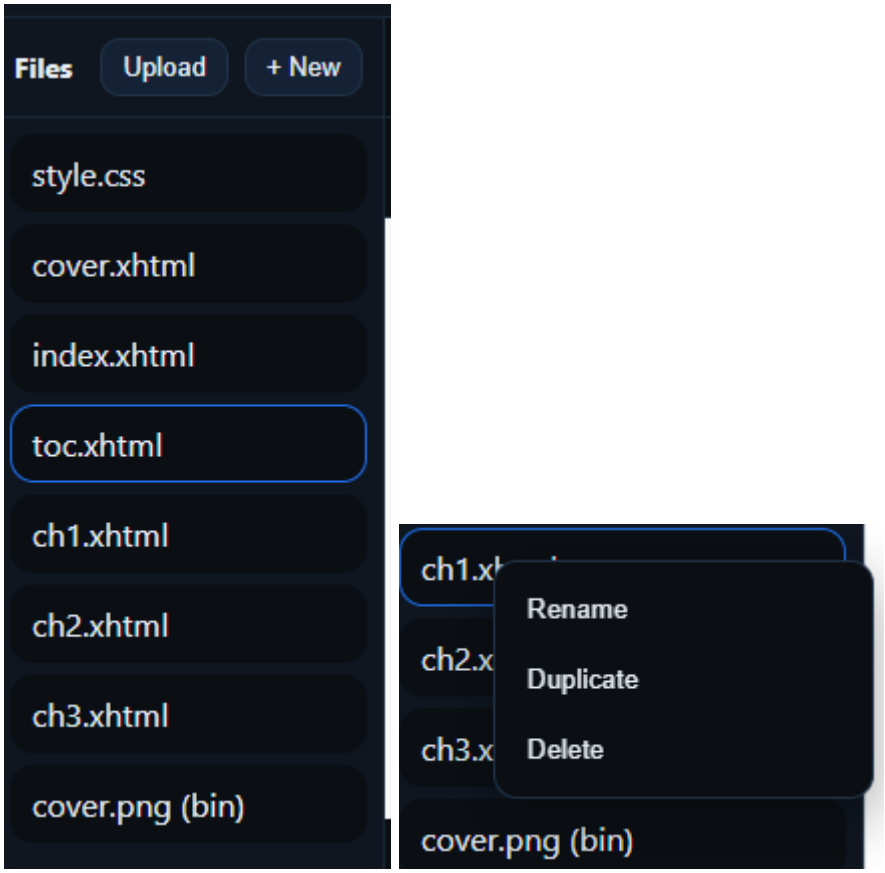


Εικόνα 5.9: Σελίδα για ανέβασμα epub



Εικόνα 5.10: Σελίδα επεξεργασίας epub με όλα τα διαθέσιμα υλικά

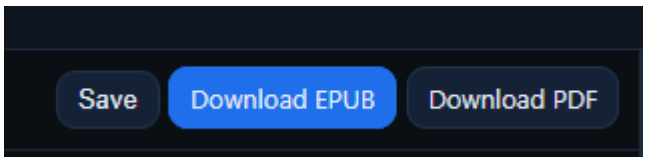
Στην **Εικόνα 5.9** παρουσιάζεται η σελίδα ανεβάσματος EPUB, όπου ο χρήστης επιλέγει το αρχείο .epub από τον υπολογιστή του και ξεκινά τη διαδικασία εισαγωγής, ώστε να δημιουργηθεί αυτόματα νέο project με όλα τα περιεχόμενα του βιβλίου.



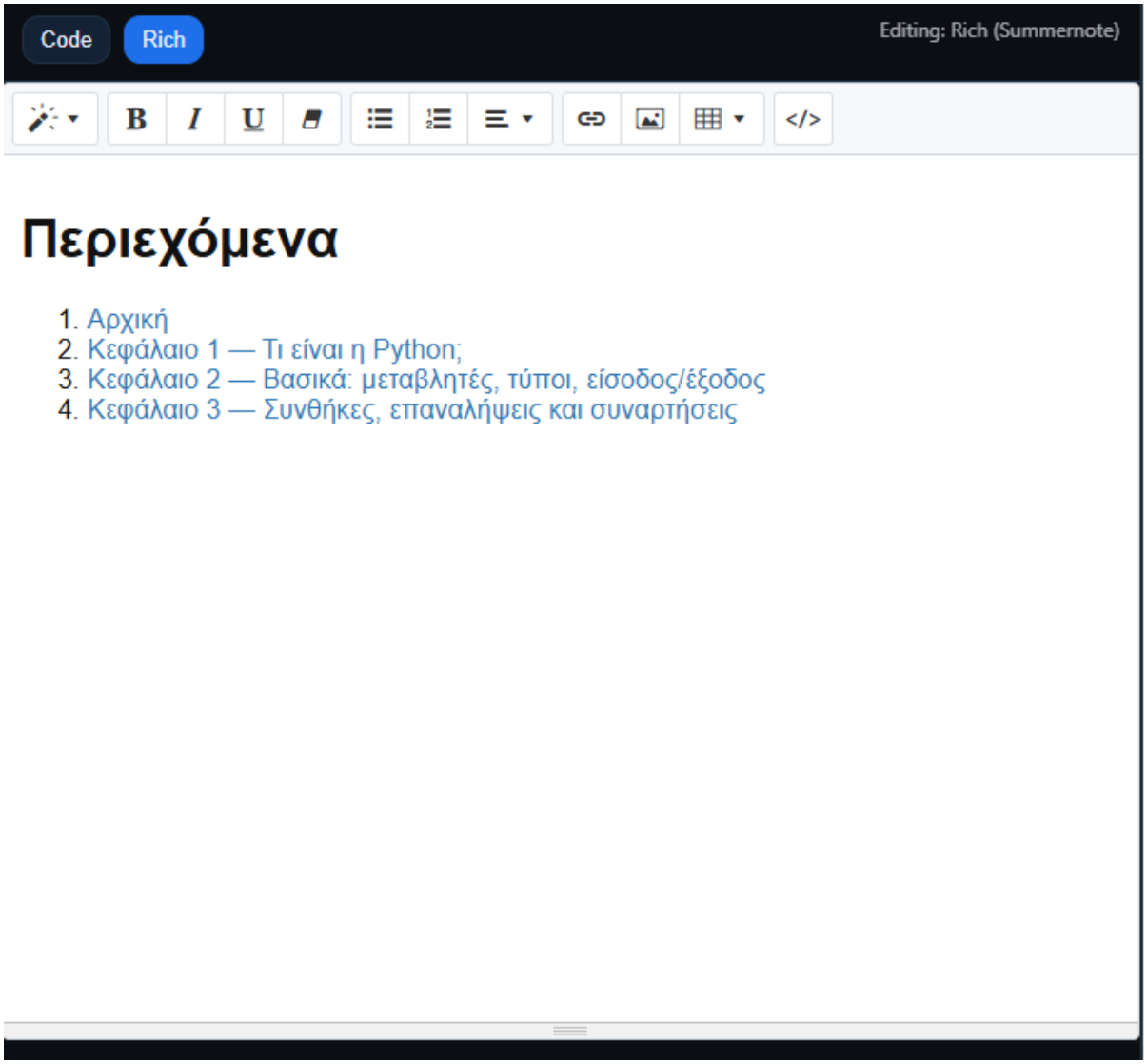
Εικόνα 5.11: Αριστερή μπάρα για εισαγωγή νέων αρχείων, ανέβασμα αρχείων και μενού με δεξί κλικ για μετονομασία, διπλασιασμός ή διαγραφή αρχείου.

Στην **Εικόνα 5.10** παρουσιάζεται η σελίδα επεξεργασίας EPUB (workspace), στην οποία προβάλλονται όλα τα διαθέσιμα υλικά του έργου (αρχεία XHTML/CSS και assets) και παρέχεται το περιβάλλον των τριών panels για επιλογή αρχείου, επεξεργασία και προεπισκόπηση.

Στην **Εικόνα 5.11** παρουσιάζεται η αριστερή μπάρα διαχείρισης αρχείων, όπου ο χρήστης μπορεί να δημιουργήσει νέο αρχείο, να ανεβάσει αρχεία (π.χ. εικόνες), καθώς και να εκτελέσει ενέργειες μέσω δεξιού κλικ όπως μετονομασία, διπλασιασμό ή διαγραφή ενός αρχείου.



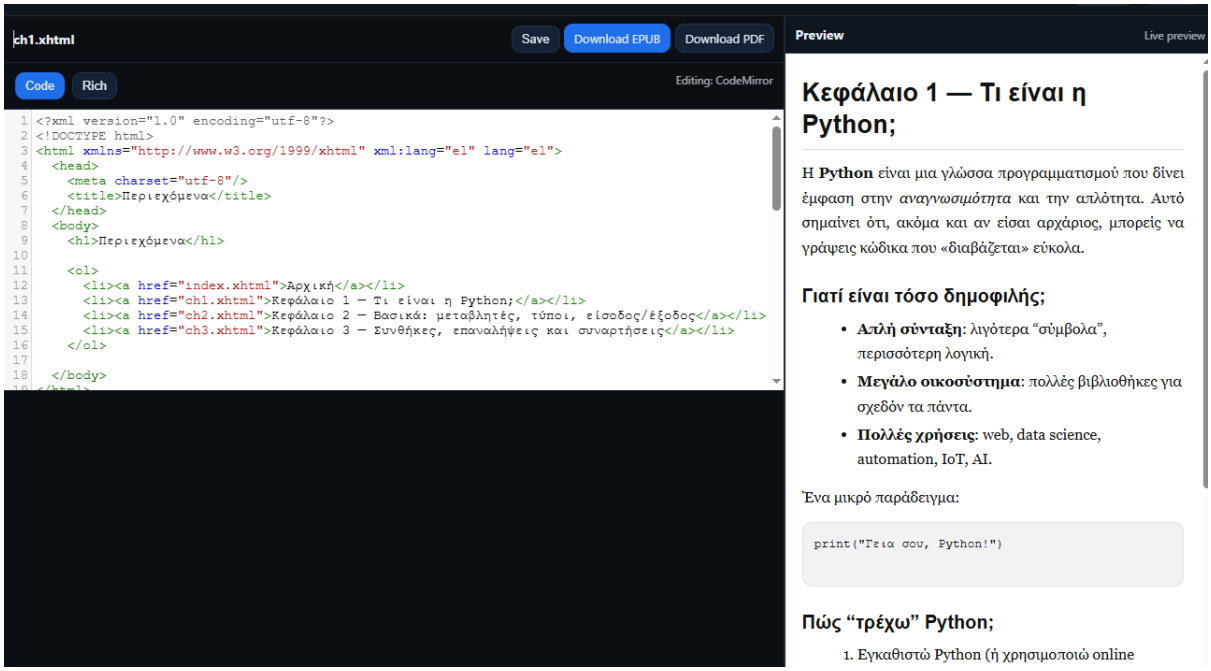
Εικόνα 5.12: Κουμπιά για παραγωγή και κατέβασμα EPUB και PDF



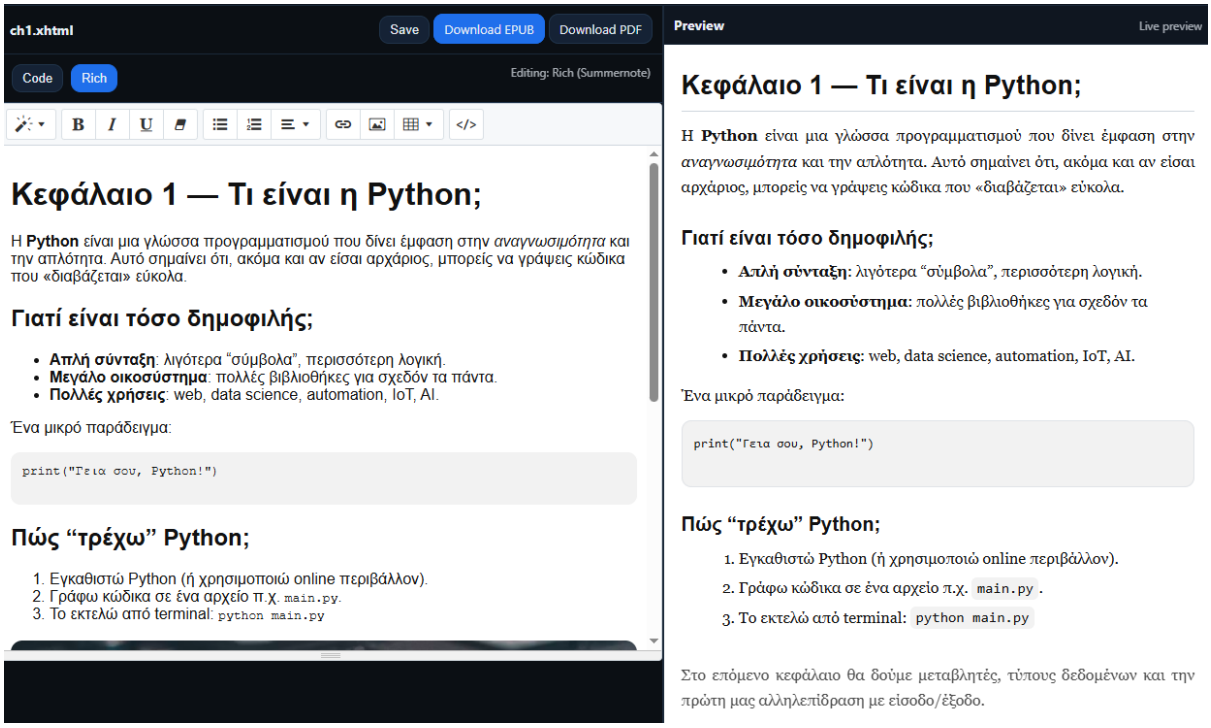
Εικόνα 5.13: Πεδίο για επεξεργασία ενός αρχείου πίνακα περιεχομένων σε rich mode

Στην **Εικόνα 5.12** παρουσιάζονται τα κουμπιά εξαγωγής του έργου, μέσω των οποίων ο χρήστης μπορεί να ξεκινήσει την παραγωγή (build) και να κατεβάσει το τελικό βιβλίο είτε σε μορφή EPUB είτε σε μορφή PDF.

Στην **Εικόνα 5.13** παρουσιάζεται το πεδίο επεξεργασίας ενός αρχείου πίνακα περιεχομένων σε λειτουργία rich mode, όπου ο χρήστης μπορεί να μορφοποιήσει το περιεχόμενο οπτικά χωρίς να απαιτείται άμεση επεξεργασία κώδικα.



Εικόνα 5.14: Πεδίο επεξεργασίας ενός αρχείου σε μορφοποίηση κώδικα – δεξιά φαίνεται η προεπισκόπηση του αποτελέσματος



Εικόνα 5.15: Πεδίο επεξεργασίας ενός αρχείου σε μορφοποίηση rich edit – δεξιά φαίνεται η προεπισκόπηση του αποτελέσματος

Στην **Εικόνα 5.14** παρουσιάζεται το πεδίο επεξεργασίας ενός αρχείου σε μορφοποίηση κώδικα (code mode), ενώ στη δεξιά πλευρά εμφανίζεται η προεπισκόπηση του αποτελέσματος, ώστε ο χρήστης να επιβεβαιώνει άμεσα την εμφάνιση των αλλαγών του.

Στην **Εικόνα 5.15** παρουσιάζεται το πεδίο επεξεργασίας ενός αρχείου σε μορφοποίηση rich edit, ενώ στη δεξιά πλευρά εμφανίζεται η αντίστοιχη προεπισκόπηση, δίνοντας τη δυνατότητα στον χρήστη να εργάζεται με οπτική μορφοποίηση και ταυτόχρονα να βλέπει σε πραγματικό χρόνο το τελικό αποτέλεσμα.

Κεφάλαιο 6ο: Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται η υλοποίηση του συστήματος, δηλαδή πώς οι απαιτήσεις και η σχεδίαση μετατράπηκαν σε λειτουργικό λογισμικό. Η παρουσίαση ακολουθεί μια πρακτική προσέγγιση: ξεκινά από τις τεχνολογίες και την οργάνωση του έργου, συνεχίζει με τη βάση δεδομένων και τα βασικά modules του backend, και έπειτα εξηγεί τη λειτουργία του frontend (τρία panels, editors, preview, διαχείριση αρχείων). Στο τέλος αναλύονται οι δύο πιο "βαριές" λειτουργίες: εισαγωγή υπάρχοντος EPUB (import) και παραγωγή τελικών αρχείων (export σε EPUB και PDF).

6.1 Τεχνολογίες και εργαλεία ανάπτυξης

6.1.1 Backend

Ο backend υλοποιήθηκε σε Python με χρήση του Flask, ένα ελαφρύ web framework που επιτρέπει γρήγορη ανάπτυξη REST endpoints και server-rendered σελίδων. Η επιλογή Flask έγινε επειδή:

- είναι απλό στη δομή και εύκολο για project πτυχιακής,
- επιτρέπει καθαρό διαχωρισμό σε routes, services και helpers,
- συνεργάζεται άμεσα με ORM (SQLAlchemy) και authentication libraries.

Για τη σύνδεση με τη βάση χρησιμοποιήθηκε MySQL, καθώς είναι ευρέως διαδεδομένη, αξιόπιστη και κατάλληλη για multi-user περιβάλλον. Η επικοινωνία γίνεται μέσω ORM (SQLAlchemy) για πιο ασφαλή και καθαρό query layer, ενώ τα migrations/alter statements μπορούν να εφαρμοστούν χειροκίνητα ή με εργαλείο migration (π.χ. Flask-Migrate) σε μελλοντική εξέλιξη.

6.1.2 Frontend

Το frontend βασίζεται σε HTML templates και JavaScript. Για τον editor επιλέχθηκαν δύο συμπληρωματικά εργαλεία:

- **CodeMirror** για code editing (XHTML/CSS/XML), επειδή προσφέρει syntax highlighting, indentation, και καλύτερη εμπειρία συγγραφής κώδικα.
- **Summernote** ως rich editor για XHTML, ώστε χρήστες που δεν θέλουν να γράφουν HTML να μπορούν να διαμορφώνουν κείμενο οπτικά. Μέσα από tabs ο χρήστης μπορεί να εναλλάσσει "Code" και "Rich".

Η προεπισκόπηση υλοποιείται με **iframe**, όπου το UI γράφει srcdoc για να αποδίδει άμεσα το XHTML περιεχόμενο. Επιπλέον, εφαρμόζεται μηχανισμός injection CSS ώστε το preview να έχει όμοια εμφάνιση με το τελικό EPUB.

6.1.3 Διαχείριση EPUB

Για την παραγωγή EPUB χρησιμοποιείται βιβλιοθήκη που υποστηρίζει δημιουργία container, προσθήκη items, και εξαγωγή σε .epub. Το build pipeline παίρνει αρχεία από DB/disk, τα τοποθετεί στο πακέτο και καθορίζει τη σειρά ανάγνωσης (spine) με βάση sort_order.

6.1.4 Παραγωγή PDF

Για PDF προτιμήθηκε μια "ελαφριά" λύση χωρίς εξωτερικά dependencies που δυσκολεύουν σε Windows. Η τεχνική είναι: συγχώνευση XHTML σε ενιαίο HTML και μετατροπή σε PDF μέσω βιβλιοθήκης (π.χ. xhtml2pdf) με ενσωμάτωση ελληνικών γραμματοσειρών (TTF). Το κομμάτι των fonts είναι κρίσιμο για την απόδοση ελληνικών χαρακτήρων και αντιμετωπίζεται με register στο ReportLab.

6.2 Οργάνωση έργου και δομή φακέλων

Η δομή του project σχεδιάστηκε ώστε να χωρίζει καθαρά το backend, τα templates, τα static assets και τον χώρο αποθήκευσης uploads:

- app.py
  Κεντρικό αρχείο Flask application, routes και configuration.
- models.py
  Περιέχει τα SQLAlchemy models: User, EpubProject, EpubFile.
- build_epub.py
  Περιέχει τη λογική παραγωγής EPUB (build pipeline).
- pdf_builder_simple.py
  Περιέχει τη λογική παραγωγής PDF από τα XHTML.
- templates/
  HTML templates (login, projects dashboard, editor).
- static/
  CSS και JS αρχεία.

    - static/editor.js → βασικό UI logic (φόρτωση αρχείων, save, preview, upload)
    - static/styles.css → layout, panels, buttons
    - static/fonts/ → fonts για PDF (DejaVu / Noto / Arial)
- uploads/
  Φάκελος στο disk για binary assets:
    - uploads/<project_id>/OEBPS/images/...
    - uploads/<project_id>/OEBPS/fonts/...

Με αυτή τη δομή:

- text αρχεία (XHTML/CSS) βρίσκονται σε DB για εύκολο edit,
- binary assets βρίσκονται σε disk ώστε να σερβίρονται γρήγορα και να μη βαραίνουν τη MySQL.

6.3 Βάση δεδομένων (MySQL) και βασικές οντότητες

6.3.1 Πίνακας χρηστών

Ο πίνακας users περιλαμβάνει τα βασικά στοιχεία ταυτοποίησης, με αποθήκευση κωδικού ως hash. Η υλοποίηση ακολουθεί την κλασική προσέγγιση: σε login γίνεται verify του hash και δημιουργείται session.

6.3.2 Πίνακας projects

Ο πίνακας epub_projects αποθηκεύει:

- τίτλο βιβλίου
- γλώσσα
- identifier (προαιρετικό)
- user_id (ιδιοκτησία)
- timestamps (created_at)

6.3.3 Πίνακας αρχείων

Ο πίνακας epub_files είναι ο "πυρήνας" της εφαρμογής. Κάθε εγγραφή αντιστοιχεί σε ένα file entry μέσα στο EPUB. Το σημαντικότερο πεδίο για τη λειτουργία "drag reorder" και export είναι το:

- sort_order (αριθμητική σειρά)

Η ταξινόμηση στη λίστα και η σειρά build γίνεται πάντα με: ORDER BY sort_order, id

Έτσι εξασφαλίζεται σταθερό ordering ακόμα και αν δύο αρχεία έχουν ίδιο sort_order.

6.4 Backend υλοποίηση: endpoints και έλεγχος πρόσβασης

6.4.1 Authorization helper

Για κάθε route που αφορά project/file, χρησιμοποιείται κοινός helper (π.χ. _project_owned_or_404) που:

- διαβάζει project από DB,
- ελέγχει ότι project.user_id == current_user.id,
- διαφορετικά επιστρέφει 404/403.

Το ίδιο ισχύει και σε file επίπεδο, ώστε κανένας χρήστης να μην έχει πρόσβαση σε αρχεία άλλου project.

6.4.2 Projects dashboard

Υλοποιείται endpoint που επιστρέφει HTML με λίστα projects. Εκεί προστίθεται επίσης φόρμα "Import EPUB" και κουμπί "New project".

6.4.3 API endpoints αρχείων

Τα endpoints δίνουν JSON:

- λίστα αρχείων
- επιστροφή περιεχομένου
- save αλλαγών

Εδώ είναι σημαντικό το backend να αποτρέπει:

- edit σε binary

- edit σε non-editable αρχεία (αν οριστεί)
- invalid paths / directory traversal

6.5 Frontend υλοποίηση: τρία panels, editors και preview

6.5.1 File list και απλοποίηση εμφάνισης

Η λίστα αρχείων εμφανίζει μόνο το filename (χωρίς OEBPS/...) για ευκολία χρήστη. Παρόλα αυτά, το πλήρες path παραμένει στο backend για σωστή διαχείριση στο EPUB package.

6.5.2 Context menu (rename/duplicate/delete)

Στο αριστερό panel υλοποιούνται ενέργειες δεξιού κλικ:

- rename → POST endpoint
- duplicate → δημιουργία νέου file entry με νέο path
- delete → διαγραφή DB record και (αν binary) διαγραφή από disk

6.5.3 Drag & drop reorder

Στο UI χρησιμοποιείται drag-and-drop (π.χ. SortableJS). Μετά από reorder:

- το UI στέλνει λίστα IDs με τη νέα σειρά,
- backend ενημερώνει sort_order,
- refresh της λίστας.

6.5.4 Live preview

Το preview υλοποιείται ως iframe με srcdoc. Για να λειτουργούν σωστά:

- CSS injection: συγχώνευση όλων CSS αρχείων και εισαγωγή σε <style>
- rewrite assets: src/href μετατρέπονται σε /u/<project_id>/<path> ώστε να φορτώνονται εικόνες από server.

6.6 Import EPUB: parsing και δημιουργία project

Η λειτουργία import είναι σύνθετη, επειδή πρέπει να μετατρέψει ένα "έτοιμο" EPUB σε εσωτερική μορφή project:

- unzip και ανάγνωση περιεχομένων
- εντοπισμός OPF
- διάκριση αρχείων σε text και binary
- αποθήκευση αντίστοιχα σε DB ή disk
- δημιουργία αρχικής σειράς (sort_order)

Αν υπάρχει spine στο OPF, μπορεί να χρησιμοποιηθεί ως αρχική σειρά. Διαφορετικά, ορίζεται σειρά με βάση λογικούς κανόνες (π.χ. insertion order).

6.7 Export: Build EPUB και Export PDF

6.7.1 Build EPUB

Το build EPUB ακολουθεί τα παρακάτω:

1. φόρτωση όλων των files του project
2. ταξινόμηση με sort_order
3. δημιουργία πακέτου EPUB και προσθήκη items
4. για binary assets: ανάγνωση από disk και ενσωμάτωση στο πακέτο
5. δημιουργία spine με βάση τη σειρά των XHTML
6. εξαγωγή .epub ως download

Η υλοποίηση πρέπει να χειρίζεται σφάλματα όπως:

- κενά XHTML (Document is empty)
- ελλιπή metadata
- λάθος paths

6.7.2 Export PDF

Το export PDF ακολουθεί αντίστοιχη λογική:

1. επιλογή XHTML/HTML κεφαλαίων με βάση sort_order
2. συγχώνευση σε ενιαίο HTML
3. injection CSS
4. μετατροπή σε PDF
5. ενσωμάτωση ελληνικών fonts για σωστή απόδοση χαρακτήρων

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] https://www.adobe.com/uk/acrobat/resources/document-files/ebook-files/epub.html

[2] https://en.wikipedia.org/wiki/EPUB

[3] https://www.continualengine.com/blog/epub-files/

[4] https://www.oreilly.com/library/view/cascading-style-sheets/0596005253/ch01s04.html

[5] https://en.wikipedia.org/wiki/Content_management_system

[6] https://www.reviversoft.com/en/file-extensions/opf

[7] https://sigil-ebook.com/

[8] http://calibre-ebook.com/

[9] https://pressbooks.com/

[10] https://reedsy.com/studio

[11] https://bookcreator.com/

[12] https://www.mysql.com/

ΠΑΡΑΡΤΗΜΑ Α

```
import os
import zipfile
import tempfile
import mimetypes
import xml.etree.ElementTree as ET
from io import BytesIO

from flask import Flask, render_template, request, redirect, url_for, jsonify, send_file, abort
from flask_login import LoginManager, login_user, logout_user, login_required, current_user

from models import db, User, EpubProject, EpubFile
from build_epub import build_epub_bytes

from pdf_builder import build_pdf_bytes_simple
from io import BytesIO



def create_app() -> Flask:
    app = Flask(__name__)

    app.config["SECRET_KEY"] = os.environ.get("SECRET_KEY", "dev-secret-change-me")

    db_url = os.environ.get("DATABASE_URL")
    if not db_url:
        mysql_user = os.environ.get("MYSQL_USER", "root")
        mysql_password = os.environ.get("MYSQL_PASSWORD", "")
        mysql_host = os.environ.get("MYSQL_HOST", "127.0.0.1")
        mysql_port = os.environ.get("MYSQL_PORT", "3306")
        mysql_db = os.environ.get("MYSQL_DB", "epub ")
        db_url = (
            f"mysql+pymysql://{mysql_user}:{mysql_password}@{mysql_host}:{mysql_port}/{mysql_db}"
            "?charset=utf8mb4"
        )

    app.config["SQLALCHEMY_DATABASE_URI"] = db_url
    app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

```
db.init_app(app)

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

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

# ---- uploads folder on disk ----
UPLOADS_DIR = os.path.join(app.root_path, "uploads")
os.makedirs(UPLOADS_DIR, exist_ok=True)

# --------------------
# Helpers
# --------------------
def safe_epub_path(p: str) -> str:
    p = (p or "").replace("\\", "/").strip()
    # remove leading ./ and leading /
    while p.startswith("./"):
        p = p[2:]
    while p.startswith("/"):
        p = p[1:]
    # block traversal
    if not p or ".." in p.split("/"):
        raise ValueError("Invalid path")
    if len(p) > 255:
        raise ValueError("Path too long")
    return p

def guess_mime(path: str) -> str:
    p = (path or "").lower()
    if p.endswith(".xhtml"):
        return "application/xhtml+xml"
    if p.endswith(".html"):
        return "text/html"
```

```
    if p.endswith(".css"):
        return "text/css"
    if p.endswith(".svg"):
        return "image/svg+xml"
    mt, _ = mimetypes.guess_type(path)
    return mt or "application/octet-stream"


def is_text_file(path: str) -> bool:
    p = (path or "").lower()
    return p.endswith((".xhtml", ".html", ".css", ".xml", ".opf", ".ncx", ".txt", ".svg", ".js"))


def parse_spine_order(zipf: zipfile.ZipFile):
    """
    Προσπαθεί να βρει spine order από OPF:
    META-INF/container.xml -> full-path opf -> manifest/spine -> href list.
    Επιστρέφει λίστα paths (όπως μέσα στο zip) με σειρά ανάγνωσης.
    Αν αποτύχει, επιστρέφει [].
    """
    try:
        container = zipf.read("META-INF/container.xml")
        root = ET.fromstring(container)
        # namespaces not always declared; handle by searching tags that endwith
        opf_path = None
        for elem in root.iter():
            if elem.tag.lower().endswith("rootfile") and "full-path" in elem.attrib:
                opf_path = elem.attrib.get("full-path")
                break
        if not opf_path:
            return []

        opf_path = safe_epub_path(opf_path)
        opf_bytes = zipf.read(opf_path)
        opf_root = ET.fromstring(opf_bytes)

        # build id->href from manifest
        id_to_href = {}
        for item in opf_root.iter():
            if item.tag.lower().endswith("item"):
```

```
            iid = item.attrib.get("id")
            href = item.attrib.get("href")
            if iid and href:
                id_to_href[iid] = href

        # base dir for resolving href
        base_dir = os.path.dirname(opf_path).replace("\\", "/")

        spine_paths = []
        for itemref in opf_root.iter():
            if itemref.tag.lower().endswith("itemref"):
                iid = itemref.attrib.get("idref")
                href = id_to_href.get(iid)
                if not href:
                    continue
                # resolve relative to opf dir
                joined = (base_dir + "/" + href).replace("\\", "/") if base_dir else href.replace("\\", "/")
                # normalize .. and .
                norm = os.path.normpath(joined).replace("\\", "/")
                norm = safe_epub_path(norm)
                spine_paths.append(norm)

        return spine_paths
    except Exception:
        return []

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

@app.get("/login")
def login():
    return render_template("login.html")
```

```
@app.post("/login")
def login_post():
    email = (request.form.get("email") or "").strip().lower()
    password = request.form.get("password") or ""
    user = User.query.filter_by(email=email).first()
    if not user:# or not user.check_password(password):
        return render_template("login.html", error="Λάθος email ή κωδικός.")
    login_user(user)
    return redirect(url_for("projects"))

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

@app.get("/projects")
@login_required
def projects():
    projs = EpubProject.query.filter_by(user_id=current_user.id).order_by(EpubProject.id.desc()).all()
    return render_template("projects.html", projects=projs)

@app.post("/projects/new")
@login_required
def project_new():
    p = EpubProject(user_id=current_user.id, title="Νέο βιβλίο", language="el", identifier="urn:uuid:demo-epub")
    db.session.add(p)
    db.session.commit()
    return redirect(url_for("editor", project_id=p.id))

@app.post("/projects/import-epub")
@login_required
def project_import_epub():
    f = request.files.get("epub_file")
    if not f or not f.filename.lower().endswith(".epub"):
        projs = EpubProject.query.filter_by(user_id=current_user.id).order_by(EpubProject.id.desc()).all()
        return render_template("projects.html", projects=projs, error="Διάλεξε ένα αρχείο .epub")
```

```
title_override = (request.form.get("title") or "").strip()

# Create project first
proj = EpubProject(
    user_id=current_user.id,
    title=title_override or os.path.splitext(os.path.basename(f.filename))[0] or "Imported EPUB",
    language="el",
    identifier="urn:uuid:demo-epub",
)
db.session.add(proj)
db.session.commit()

proj_upload_dir = os.path.join(UPLOADS_DIR, str(proj.id))
os.makedirs(proj_upload_dir, exist_ok=True)

# Save epub temp
tmp = tempfile.NamedTemporaryFile(suffix=".epub", delete=False)
tmp_path = tmp.name
tmp.close()
f.save(tmp_path)

try:
    with zipfile.ZipFile(tmp_path, "r") as z:
        spine_order = parse_spine_order(z)  # list of zip paths in reading order

        # Build sort_order map for spine docs
        spine_pos = {}
        for idx, pth in enumerate(spine_order, start=1):
            spine_pos[pth] = idx * 10

        # Iterate zip entries
        sort_fallback = 100000  # non-spine items after
        for zi in z.infolist():
            if zi.is_dir():
                continue
            name = zi.filename.replace("\\", "/")
            try:
```

```
            name = safe_epub_path(name)
        except ValueError:
            continue

        data = z.read(zi.filename)

        mime = guess_mime(name)
        editable = True
        binary = not is_text_file(name)

        so = spine_pos.get(name)
        if so is None:
            sort_fallback += 10
            so = sort_fallback

        # store
        ef = EpubFile(
            project_id=proj.id,
            path=name,
            mime=mime,
            is_editable=editable,
        )

        # Αν έχεις ήδη πεδία is_binary/sort_order στη βάση σου, ξεσχολίασε τα 2:
        if hasattr(EpubFile, "is_binary"):
            ef.is_binary = binary
        if hasattr(EpubFile, "sort_order"):
            ef.sort_order = so

        if binary:
            # write to uploads/<proj.id>/<path>
            abs_path = os.path.join(proj_upload_dir, name)
            os.makedirs(os.path.dirname(abs_path), exist_ok=True)
            with open(abs_path, "wb") as wf:
                wf.write(data)
            ef.content = ""
        else:
            # decode as utf-8 best effort
```

```
                try:
                    ef.content = data.decode("utf-8")
                except UnicodeDecodeError:
                    ef.content = data.decode("utf-8", errors="replace")

            db.session.add(ef)

        db.session.commit()

    except zipfile.BadZipFile:
        db.session.delete(proj)
        db.session.commit()
        projs = EpubProject.query.filter_by(user_id=current_user.id).order_by(EpubProject.id.desc()).all()
        return render_template("projects.html", projects=projs, error="Το αρχείο δεν είναι έγκυρο EPUB/ZIP.")
    finally:
        try:
            os.remove(tmp_path)
        except OSError:
            pass

    return redirect(url_for("editor", project_id=proj.id))

@app.get("/project/<int:project_id>")
@login_required
def editor(project_id: int):
    proj = EpubProject.query.filter_by(id=project_id, user_id=current_user.id).first_or_404()
    return render_template("editor.html", project=proj)

# ---------------------
# API helpers
# ---------------------
def _project_owned_or_404(project_id: int) -> EpubProject:
    proj = EpubProject.query.filter_by(id=project_id, user_id=current_user.id).first()
    if not proj:
        abort(404)
    return proj

def _file_owned_or_404(file_id: int) -> EpubFile:
```

```
    f = EpubFile.query.get(file_id)
    if not f:
        abort(404)
    proj = EpubProject.query.filter_by(id=f.project_id, user_id=current_user.id).first()
    if not proj:
        abort(404)
    return f

@app.get("/api/project/<int:project_id>/files")
@login_required
def api_list_files(project_id: int):
    _project_owned_or_404(project_id)

    q = EpubFile.query.filter_by(project_id=project_id)

    # αν υπάρχει sort_order, ταξινόμησε έτσι
    if hasattr(EpubFile, "sort_order"):
        q = q.order_by(EpubFile.sort_order.asc(), EpubFile.id.asc())
    else:
        q = q.order_by(EpubFile.path.asc())

    files = q.all()
    out = []
    for f in files:
        item = {
            "id": f.id,
            "path": f.path,
            "mime": f.mime,
            "is_editable": f.is_editable,
        }
        if hasattr(f, "is_binary"):
            item["is_binary"] = bool(getattr(f, "is_binary"))
        if hasattr(f, "sort_order"):
            item["sort_order"] = int(getattr(f, "sort_order") or 0)
        out.append(item)
    return jsonify(out)

@app.post("/api/project/<int:project_id>/files")
```

```
@login_required
def api_create_file(project_id: int):
    _project_owned_or_404(project_id)
    data = request.get_json(force=True)

    path = (data.get("path") or "").strip()
    content = data.get("content") or ""
    mime = data.get("mime") or "application/xhtml+xml"
    is_editable = bool(data.get("is_editable", True))

    try:
        path = safe_epub_path(path)
    except ValueError:
        return jsonify({"ok": False, "error": "Μη έγκυρο path"}), 400

    f = EpubFile(project_id=project_id, path=path, content=content, mime=mime, is_editable=is_editable)

    # default ordering at end (if supported)
    if hasattr(EpubFile, "sort_order"):
        max_so = db.session.query(db.func.max(EpubFile.sort_order)).filter_by(project_id=project_id).scalar()
        f.sort_order = int(max_so or 0) + 10

    if hasattr(EpubFile, "is_binary"):
        f.is_binary = False

    db.session.add(f)
    db.session.commit()
    return jsonify({"ok": True, "id": f.id})

@app.get("/api/file/<int:file_id>")
@login_required
def api_get_file(file_id: int):
    f = _file_owned_or_404(file_id)
    payload = {
        "id": f.id,
        "path": f.path,
        "content": f.content,
        "mime": f.mime,
```

```
        "is_editable": f.is_editable,
    }
    if hasattr(f, "is_binary"):
        payload["is_binary"] = bool(getattr(f, "is_binary"))
    if hasattr(f, "sort_order"):
        payload["sort_order"] = int(getattr(f, "sort_order") or 0)
    return jsonify(payload)


@app.post("/api/file/<int:file_id>")
@login_required
def api_save_file(file_id: int):
    f = _file_owned_or_404(file_id)
    if not f.is_editable:
        return jsonify({"ok": False, "error": "Το αρχείο δεν είναι editable"}), 400

    if hasattr(f, "is_binary") and getattr(f, "is_binary"):
        return jsonify({"ok": False, "error": "Binary αρχείο δεν σώζεται σαν text"}), 400

    data = request.get_json(force=True)
    f.content = data.get("content") or ""
    db.session.commit()
    return jsonify({"ok": True})


@app.get("/api/project/<int:project_id>/download")
@login_required
def api_download_epub(project_id: int):
    proj = _project_owned_or_404(project_id)
    files = EpubFile.query.filter_by(project_id=project_id).all()
    epub_bytes = build_epub_bytes(proj, files)

    bio = BytesIO(epub_bytes)
    bio.seek(0)
    filename = (proj.title or "book").replace(" ", "_") + ".epub"
    return send_file(
        bio,
        as_attachment=True,
        download_name=filename,
        mimetype="application/epub+zip",
```

```
        )


    @app.get("/api/project/<int:project_id>/download.pdf")
    @login_required
    def api_download_pdf(project_id: int):
        proj = _project_owned_or_404(project_id)
        files = EpubFile.query.filter_by(project_id=project_id).all()

        pdf_bytes = build_pdf_bytes_simple(proj, files)

        bio = BytesIO(pdf_bytes)
        bio.seek(0)
        filename = (proj.title or "book").replace(" ", "_") + ".pdf"
        return send_file(
            bio,
            as_attachment=True,
            download_name=filename,
            mimetype="application/pdf",
        )




    return app


app = create_app()

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