National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Development of Web-Based Content Management Software Using Open Technologies

Diploma Thesis of

Oikonomou Eleftherios

Supervisor: Vassilios Vescoukis, Professor NTUA

Athens, February 2025

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Development of Web-Based Content Management Software Using Open Technologies

Diploma Thesis of

Oikonomou Eleftherios

Supervisor: Vassilios Vescoukis, Professor NTUA

Approved by the examination committee on February 21st.

(Signature)	(Signature)	(Signature)
.....	.....	.....
Vassilios Vescoukis	Nikolaos Papaspurou	Kostis Sagonas
Professor	Professor	Professor

Athens, February 2025

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

(Signature)

.....

Oikonomou Eleftherios

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Acknowledgments

After completing my thesis, my journey to the School of Electrical and Computer Engineering at the National Technical University of Athens (NTUA) ends. This would not have been possible without my family and friends, who supported me throughout my studies, as they always do.

I want to express my gratitude to my professor and supervisor, Mr. Vassilis Vescoukis, for the opportunity to work on such an important subject and for his valuable guidance, feedback, and excellent cooperation during my thesis.

In addition, I wish to acknowledge the PhD Candidate and Researcher Hadjichristofi Christos for his help and guidance.

Athens, February 2025

Oikonomou Eleftherios

Περίληψη

Κάθε ίδρυμα χρειάζεται μια διαδικτυακή παρουσία στην ψηφιακή εποχή. Αυτή η διπλωματική εργασία παρουσιάζει το σχεδιασμό και την υλοποίηση ενός σύγχρονου, κλιμακούμενου, υψηλής απόδοσης ιστότοπου για το SoftLab ΕΜΠ, χρησιμοποιώντας μια αρχιτεκτονική headless CMS για διαχείριση περιεχομένου με βάση API, αυτοματισμούς με άλλες πλατφόρμες και στρατηγικές hosting της εφαρμογής χρησιμοποιώντας ανοιχτές τεχνολογίες.

Η παρούσα διατριβή εξετάζει τις αρχιτεκτονικές επιλογές, τις μεθοδολογίες και τις στρατηγικές ανάπτυξης που απαιτούνται για την υλοποίηση μιας ακαδημαϊκής διαδικτυακής πλατφόρμας που είναι ασφαλής και έχει εύκολη διαχείριση και συντήρηση ενώ γίνεται hosted σε ιδιόκτητους servers. Αναλύει επίσης σε βάθος τις τεχνικές βελτιστοποίησης των επιδόσεων της και τα ζητήματα ασφάλειας.

Έπειτα, εξετάζει την υλοποίηση και καταδεικνύει σημαντικές βελτιώσεις στην εμπειρία του χρήστη, τη συντηρησιμότητα και την αυτοματοποίηση σε σύγκριση με τις συμβατικές στατικές ιστοσελίδες. Τέλος, αναφέρονται μελλοντικές βελτιώσεις για την πλατφόρμα.

Keywords:

Next.js, Strapi, Headless CMS, Αρχιτεκτονική Βασισμένη σε API, Βελτιστοποίηση Απόδοσης Ιστού, Nginx, SQLite, Βελτιστοποίηση για τις μηχανές αναζήτησης, SSG, Διαδίκτυο, Διαχείριση Περιεχομένου

Abstract

Every institution requires an online presence in the digital era. This diploma thesis presents the design and implementation of a modern, scalable, high-performance website for SoftLab NTUA, utilizing a headless CMS architecture for API-based content management, automation with other platforms, and hosting strategies using open technologies.

This thesis examines the architectural choices, methodologies, and development strategies required to implement an academic web platform that is secure, easy to manage, and maintainable while being hosted on self-owned servers. Additionally, it provides an in-depth analysis of performance optimization techniques and security concerns.

Subsequently, it explores the implementation phase and highlights significant improvements in user experience, maintainability, and automation compared to conventional static websites. Finally, it outlines potential future enhancements for the platform.

Keywords:

Next.js, Strapi, Headless CMS, API-driven architecture, Web Performance Optimization, Nginx, SQLite, SEO, SSG, ISR, Web, Content Management

Εκτεταμένη Περίληψη

Στην ψηφιακή εποχή, η ανάγκη για μια σύγχρονη, υψηλής απόδοσης και ευέλικτη διαδικτυακή παρουσία καθίσταται απαραίτητη για κάθε ακαδημαϊκό ίδρυμα και ερευνητικό εργαστήριο. Η παρούσα διατριβή εξετάζει τον σχεδιασμό και την υλοποίηση ενός δικτυακού συστήματος διαχείρισης περιεχομένου (CMS) για το SoftLab NTUA, χρησιμοποιώντας ανοικτές τεχνολογίες και υιοθετώντας μια αρχιτεκτονική βασισμένη σε headless CMS. Η λύση αυτή στοχεύει όχι μόνο στη βελτιστοποίηση της εμπειρίας του χρήστη και της ορατότητας στα αποτελέσματα αναζήτησης, αλλά και στην απλοποίηση της διαδικασίας ενημέρωσης και διαχείρισης του περιεχομένου μέσω αυτοματοποιημένων διεργασιών.

Εισαγωγή και Δήλωση Προβλήματος

Υπάρχει ανάγκη για ένα σύγχρονο και ευέλικτο σύστημα διαχείρισης περιεχομένου που να ανταποκρίνεται στις απαιτήσεις ενός ερευνητικού περιβάλλοντος. Τα παραδοσιακά, μονολιθικά συστήματα διαχείρισης περιεχομένου συχνά παρουσιάζουν περιορισμούς όσον αφορά την κλιμακωσιμότητα, την απόδοση και την ευκολία στη συντήρηση. Με την αύξηση των απαιτήσεων για ταχύτητα φόρτωσης, βελτιστοποίηση SEO και ενσωμάτωση με εξωτερικές υπηρεσίες (όπως το API του LinkedIn για αυτόματη δημοσίευση ενημερώσεων), καθίσταται επιτακτική η ανάγκη για μια προσέγγιση που βασίζεται σε ένα headless CMS. Η δομή της διατριβής έχει οργανωθεί με τρόπο που ξεκινά από την τεχνολογική αναδρομή και καταδεικνύει τα πλεονεκτήματα της μοντέρνας αρχιτεκτονικής, προχωρώντας στη συνέχεια στην υλοποίηση, την ανάπτυξη και την παρουσίαση της εφαρμογής, ώστε να προσφέρει μια ολοκληρωμένη εικόνα για το σχεδιασμό και την πρακτική εφαρμογή της λύσης.

Επισκόπηση Απαιτούμενων Τεχνολογιών για την Υλοποίηση του Προτεινόμενου Συστήματος

1. CMS – Μονολιθικά και Headless:
   Αρχικά, γίνεται αναφορά στη σημασία των Content Management Systems για την οργάνωση και διαχείριση του ψηφιακού περιεχομένου. Παρουσιάζονται οι παραδοσιακές λύσεις, οι οποίες βασίζονται σε μονολιθική αρχιτεκτονική, όπως οι WordPress, Joomla και Drupal. Ενώ αυτές παρέχουν ένα ολοκληρωμένο περιβάλλον για τη διαχείριση περιεχομένου, συχνά παρουσιάζουν προβλήματα απόδοσης και περιορισμένη ευελιξία για μελλοντικές επεκτάσεις. Αντίθετα, η προσέγγιση του headless CMS αποσπά το back-end από το front-end, επιτρέποντας τη δημιουργία περιεχομένου μέσω API και την ανεξάρτητη ανάπτυξη των δύο. Στο πλαίσιο αυτό, το Strapi επιλέγεται ως ο κύριος πάροχος, λόγω της ευελιξίας, του ανοικτού κώδικα και της δυνατότητας ενσωμάτωσης με μοντέρνα εργαλεία ανάπτυξης.

2. Front-end Τεχνολογίες:
   Στην ανάπτυξη του front-end, το Next.js επιλέγεται ως κύριο framework, καθώς προσφέρει δυνατότητες όπως το Static Site Generation (SSG) και το Server-Side Rendering (SSR), οι οποίες συμβάλλουν στην ταχύτητα φόρτωσης και στη βελτιστοποίηση SEO. Επιπλέον, οι βασικές τεχνολογίες – HTML, CSS και JavaScript – συνδυάζονται με σύγχρονες βιβλιοθήκες και frameworks για τη δημιουργία μιας δυναμικής και ανταποκρινόμενης εφαρμογής. Η υποστήριξη για πολλαπλές γλώσσες (i18n) είναι κρίσιμη για το ερευνητικό περιβάλλον, καθώς επιτρέπει την προσβασιμότητα σε χρήστες από διαφορετικές γλωσσικές ομάδες..

3. SEO, SSG, SSR και CSR:
   Η βελτιστοποίηση για τις μηχανές αναζήτησης (SEO) αποτελεί κρίσιμο παράγοντα για την επιτυχία του ιστότοπου. Η χρήση στρατηγικών όπως το SSG και το SSR εξασφαλίζει ότι οι σελίδες παράγονται είτε εκ των προτέρων είτε δυναμικά στον server, δίνοντας πλήρη HTML στον χρήστη και στους crawlers των μηχανών αναζήτησης. Παράλληλα, παρουσιάζονται και τα μειονεκτήματα της Client-Side Rendering (CSR), η οποία μπορεί να επηρεάσει αρνητικά την ορατότητα του περιεχομένου.

4. API Integration και Αυτοματοποίηση:
   Ένα επιπρόσθετο σημαντικό στοιχείο αποτελεί η ενσωμάτωση APIs, που επιτρέπει την ανταλλαγή δεδομένων μεταξύ της πλατφόρμας και

εξωτερικών υπηρεσιών. Μέσω της χρήσης Webhooks από το Strapi CMS, η εφαρμογή επιτυγχάνει αυτόματη ενημέρωση του περιεχομένου, και ενσωμάτωση με το LinkedIn API, δημοσιεύοντος αυτόματα ανακοινώσεις στο προφίλ του Softlab NTUA στο Linkeidn.

5. Deployment και Βάσεις Δεδομένων:
   Η επιλογή του SQLite ως απλή, βάση δεδομένων που λειτουργεί σε ένα αρχείο και η χρήση ενός αντίστροφου διακομιστή (Nginx) για εξισορρόπηση φορτίου, προσωρινή αποθήκευση και κρυπτογράφηση ενισχύει την απόδοση και την ασφάλεια της πλατφόρμας, καθιστώντας την ιδανική για deployment σε ιδιόκτητα μηχανήματα με αυστηρές απαιτήσεις ασφαλείας και ελέγχου.

Υλοποίηση της Πλατφόρμας

Στο τρίτο κεφάλαιο της διατριβής παρουσιάζεται η υλοποίηση της προτεινόμενης διαδικτυακής πλατφόρμας, η οποία αποτελεί το τελικό αποτέλεσμα της συνδυαστικής επιλογής των τεχνολογιών που περιγράφηκαν στο προηγούμενο κεφάλαιο. Η υλοποίηση αυτή επιδιώκει όχι μόνο να αξιοποιήσει πλήρως τις δυνατότητες ενός headless CMS, αλλά και να εξασφαλίσει υψηλή απόδοση, ευκολία στη διαχείριση του περιεχομένου και ασφαλή λειτουργία του συστήματος.

1. Ενσωμάτωση του Strapi CMS και Διαχείριση Περιεχομένου
   Το Strapi, επιλέχθηκε ως open source και αυτο-φιλοξενούμενου headless CMS. Με την ευελιξία που παρέχει στη δημιουργία και διαχείριση περιεχομένου μέσω τόσο RESTful και GraphQL API, το Strapi διευκολύνει την υλοποίηση ενός δυναμικού περιβάλλοντος, όπου οι μη τεχνικοί χρήστες μπορούν να επεξεργάζονται, να οργανώνουν και να δημοσιεύουν περιεχόμενο με ευκολία. Η οπτική αναπαράσταση του σχήματος της βάσης (Schema Builder), η δυνατότητα διαφορετικών δικαιωμάτων μεταξύ των χρηστών και η ευκολία χρήσης της διαχειριστικής επιφάνειας προσφέρουν μια ολοκληρωμένη λύση για τις ανάγκες της εφαρμογής.

2. Βάση Δεδομένων
   Για την αποθήκευση των δεδομένων επιλέχθηκε το SQLite, λόγω της απλότητας, της φορητότητας και της άμεσης ενσωμάτωσής του στο Strapi. Η του καθιστά τη διαδικασία ανάπτυξης και συντήρησης του συστήματος

πιο αποτελεσματική, χωρίς την ανάγκη για περίπλοκες ρυθμίσεις διακομιστών βάσεων δεδομένων.

3. Υλοποίηση του Front-end με Next.js
   Η ανάπτυξη του front-end πραγματοποιείται με το Next.js, το οποίο προσφέρει δυνατές δυνατότητες όπως το Static Site Generation (SSG) και το Server-Side Rendering (SSR). Μέσω αυτών των τεχνικών, επιτυγχάνεται βέλτιστη απόδοση, γρήγορη φόρτωση σελίδων και εξαιρετική βελτιστοποίηση για τις μηχανές αναζήτησης (SEO). Επιπλέον, η ενσωμάτωση πολυγλωσσικών δυνατοτήτων (i18n) εξασφαλίζει ότι η πλατφόρμα εξυπηρετεί ένα διεθνές κοινό.

4. Back-end Λογική και Ενσωμάτωση LinkedIn API
   Το Next.js API routes χρησιμοποιείται για την υλοποίηση του back-end, επιτρέποντας τη δημιουργία εξειδικευμένων λειτουργιών, όπως η αυτοματοποίηση της δημοσίευσης περιεχομένου στο LinkedIn. Η ενσωμάτωση με το LinkedIn API εξασφαλίζει ότι οι νέες ανακοινώσεις δημοσιεύονται αυτόματα στην επίσημη σελίδα του SoftLab NTUA, συμβάλλοντας στην προβολή της έρευνας και των δραστηριοτήτων του ιδρύματος.

5. Deployment
   Η πλατφόρμα αναπτύσσεται με σκοπό να είναι hosted σε ιδιόκτητους servers. Για αυτό έγινε επιλογή του Nginx ως αντίστροφου διακομιστή, που επίσης ενισχύει την ασφάλεια και την απόδοση του συστήματος. Ο Nginx διαχειρίζεται τη δρομολόγηση των αιτήσεων, παρέχει caching, διευκολύνει την κρυπτογράφηση και λειτουργεί ως επιπλέον επίπεδο προστασίας ενάντια σε επιθέσεις, εξασφαλίζοντας την ομαλή και αξιόπιστη λειτουργία της εφαρμογής.

6. Διαχείριση Περιεχομένου και Ενημερώσεις
   Δεδομένου ότι ο ιστότοπος παράγεται ως στατικά αρχεία, απαιτείται μια μηχανή για την ενημέρωση του περιεχομένου. Η διαδικασία αυτή επιτυγχάνεται μέσω δύο βασικών μεθόδων: των πλήρων επανακατασκευών (rebuilds) του front-end και της χρήσης της τεχνικής Incremental Static Regeneration (ISR) του Next.js. Με τον ISR, οι σελίδες ανανεώνονται στο παρασκήνιο χωρίς διακοπή της εφαρμογής, εξασφαλίζοντας έτσι συνεχείς ενημερώσεις με ελάχιστο κόστος χρόνου και πόρων.

7. Μέτρα Ασφαλείας και Απομόνωσης
   Η ασφάλεια αποτελεί ακρογωνιαίο λίθο της υλοποίησης. Η αρχιτεκτονική, με τον διαχωρισμό μεταξύ front-end και back-end, μειώνει την επιφάνεια επίθεσης, ενώ τα ασφαλισμένα API endpoints προστατεύουν τα δεδομένα από μη εξουσιοδοτημένη πρόσβαση. Επιπλέον, ο Nginx λειτουργεί ως Security Gateway, εφαρμόζοντας πολιτικές περιορισμού πρόσβασης, rate limiting και προστασίας κατά των επιθέσεων DDoS.

8. Συνολική Αρχιτεκτονική και Διαγράμματα
   Το κεφάλαιο ολοκληρώνεται με μια συνολική επισκόπηση της αρχιτεκτονικής της εφαρμογής, μέσω UML. Αυτά τα διαγράμματα απεικονίζουν με σαφήνεια την αλληλεπίδραση μεταξύ των διαφόρων στοιχείων του συστήματος, τον τρόπο που επικοινωνούν μεταξύ τους και το συνολικό περιβάλλον ανάπτυξης, προσφέροντας μια ολοκληρωμένη εικόνα του έργου.

Εκκίνηση ή Επαναφορά της Εφαρμογής

Στο τέταρτο κεφάλαιο περιγράφονται οι διαδικασίες που απαιτούνται για την εκκίνηση ή την επαναφορά της εφαρμογής σε ένα παραγωγικό περιβάλλον. Η διαδικασία ξεκινά με την απόκτηση του πηγαίου κώδικα από το αποθετήριο, διασφαλίζοντας ότι όλα τα απαραίτητα στοιχεία – το front-end, το back-end, οι ρυθμίσεις της βάσης δεδομένων και οι εντολές ανάπτυξης – είναι διαθέσιμα και ενημερωμένα. Στη συνέχεια, παρουσιάζεται η διαδικασία επαναφοράς της βάσης δεδομένων SQLite, μαζί με τα σχετικά αρχεία πολυμέσων, ώστε να ανακτηθεί πλήρως το αποθηκευμένο περιεχόμενο.

Η σωστή διαμόρφωση των ευαίσθητων παραμέτρων (API keys, μυστικά) μέσω αρχείων περιβάλλοντος (.env) είναι κρίσιμη και εξασφαλίζει την ασφαλή επικοινωνία μεταξύ των συστημικών στοιχείων. Ακολουθεί η εκκίνηση των εφαρμογών, που περιλαμβάνει την εγκατάσταση των απαραίτητων βιβλιοθηκών, τη μεταγλώττιση του κώδικα και την εκκίνηση των επιμέρους υπηρεσιών. Επιπλέον, εξηγείται η ρύθμιση του αντίστροφου διακομιστή (Nginx), ο οποίος δρομολογεί τις εισερχόμενες αιτήσεις μεταξύ του front-end και του back-end. Τέλος, ενημερώνονται οι ρυθμίσεις DNS ώστε το επίσημο domain να δείχνει στη νέα εικονική μηχανή, ολοκληρώνοντας τη μετάβαση στο νέο περιβάλλον.

Επίδειξη της Εφαρμογής

Στο πέμπτο κεφάλαιο παρουσιάζει μια πρακτική επίδειξη των βασικών λειτουργιών της εφαρμογής και της εμπειρίας του τελικού χρήστη. Ξεκινά με την παρουσίαση της διαχειριστικής επιφάνειας του Content Management System (CMS) που υλοποιείται με το Strapi. Οι διαχειριστές μπορούν εύκολα να δημιουργούν, να επεξεργάζονται και να δημοσιεύουν περιεχόμενο μέσω του εύχρηστου Content Manager. Αναδεικνύονται βασικές λειτουργίες όπως η δημιουργία νέων εγγραφών και η υποστήριξη πολλαπλών γλωσσών, που επιτρέπουν την αποτελεσματική διαχείριση περιεχομένου για ένα ευρύ κοινό.

Στη συνέχεια, παρουσιάζεται το front-end της εφαρμογής, ανεπτυγμένο με το Next.js, που προσφέρει ένα σύγχρονο, responsive design και γρήγορους χρόνους φόρτωσης. Τα διαδραστικά στοιχεία όπως το φιλτράρισμα του περιεχομένου με βάση την επιλεγμένε ομάδα εργαστηρίου, βελτιώνουν την πλοήγηση και την πρόσβαση στο περιεχόμενο. Τέλος, επιδεικνύεται η αυτόματη ενσωμάτωση με το LinkedIn, όπου, όταν δημοσιεύεται νέο περιεχόμενο στο CMS, δίνεται η επιλογή να δημοσιευτεί αυτόματα και στην επίσημη σελίδα του SoftLab NTUA στο LinkedIn. Αυτή η ολοκληρωμένη λειτουργία αποδεικνύει την αποτελεσματικότητα των επιλεγμένων τεχνολογιών, προσφέροντας ένα δυναμικό, αποδοτικό και εύχρηστο σύστημα.

Βελτιώσεις

Το τελευταίο κεφάλαιο εξετάζει προτάσεις για μελλοντικές βελτιώσεις που στοχεύουν σε καλύτερη απόδοση, επεκτασιμότητα και ασφάλεια της εφαρμογής. Αρχικά, προτείνεται η υλοποίηση ενός αυτοματοποιημένου μηχανισμού επανακατασκευής (build), ο οποίος θα επιτρέπει στους διαχειριστές να ενεργοποιούν ενημερώσεις του front-end απευθείας από το CMS, εξασφαλίζοντας ότι οι αλλαγές στο περιεχόμενο εμφανίζονται αμέσως χωρίς χειροκίνητη επανακατασκευή της εφαρμογής. Επιπλέον, τονίζεται η ανάγκη για μια αυτοματοποιημένη διαδικασία backup της βάσης δεδομένων SQLite και των σχετικών αρχείων πολυμέσων ώστε να μειώνεται ο κίνδυνος απώλειας δεδομένων σε περίπτωση βλάβης.

Επιπρόσθετα, προτείνεται η ανάπτυξη ενός αυτοματοποιημένου μηχανισμού αποκατάστασης που θα αποκαθιστά αυτόματα την εφαρμογή σε περίπτωση βλάβης. Η ενσωμάτωση ενός ειδικού mail server με το Strapi για την αποστολή ειδοποιήσεων προτείνεται επίσης για την περαιτέρω βελτίωση της διαχείρισης. Τέλος, εξετάζονται μελλοντικές API ενσωματώσεις, όπως η σύνδεση με ακαδημαϊκά repositories και η αξιοποίηση του συστήματος αυθεντικοποίησης του ιδρύματος, για την επέκταση των δυνατοτήτων της πλατφόρμας και την ενίσχυση της ασφάλειας και της διαχείρισης χρηστών.

Table of Contents

Chapter 1 - Introduction

In the rapidly evolving digital world, having a modern, performant, and scalable web presence is essential for any research institution. SoftLab NTUA, as a leading software engineering laboratory, requires a website that reflects its cutting-edge research, facilitates collaboration, and provides easy access to academic content for students, researchers, and external visitors.

The new website is designed with a modern technology stack to ensure:

- Improved User Experience (UX): A sleek, responsive, and accessible interface optimized for desktop and mobile users.
- Dynamic & Scalable Content Management: A CMS-driven architecture that allows administrators to update content effortlessly.
- Enhanced Performance & SEO: By leveraging Next.js with Static Site Generation (SSG) and Server-Side Rendering (SSR), the site achieves fast load times and optimal search engine visibility.
- Automation & API Integrations: Webhooks and structured APIs enable automatic content updates and integration with third-party services like LinkedIn and research repositories.
- Secure Deployment: Using Nginx and following best practices, the infrastructure is scalable, portable, and easy to maintain in a production environment.

This approach ensures that the NTUA website is future-proof, easy to maintain, and adaptable to evolving technological and academic needs.

1.1 Problem Statement

In an era where digital presence is crucial for academic and research institutions, a modern, high-performance, scalable website is more pressing than ever. Traditional web platforms, often built on static architectures, lack the required flexibility, maintainability, and interoperability.

As research output grows and digital engagement becomes more complex, institutions must adopt dynamic, API-driven architectures that enhance usability, automation, and integration with external services.

The redevelopment of the website aims to address several fundamental challenges associated with outdated web infrastructures:

- Limited Content Management Capabilities: Traditional websites rely on manual updates and lack a structured content management system (CMS), leading to inefficiencies in updating and maintaining information.

- Suboptimal Performance and SEO Deficiencies: Without Server-Side Rendering (SSR) or Static Site Generation (SSG), content loading times increase, affecting user experience and search engine rankings.

- Lack of API and Automation Support: Modern academic workflows demand integration with third-party platforms like LinkedIn and other scholarly repositories.

- Scalability and Maintainability Issues: Monolithic and static websites are difficult to extend, requiring a high development effort for new features and raising concerns about long-term sustainability. They are also difficult for non-technical people to operate.

- Security and Deployment Constraints: Traditional hosting solutions lack containerized deployment strategies, which increases the complexity of managing updates, security, and scalability.

1.2 Structure

This thesis is structured into six main chapters, each addressing different aspects of developing a modern website for SoftLab NTUA. The organization follows a logical sequence, starting with theoretical foundations and technological choices, progressing through implementation details, and concluding demonstrations, and improvements.

The first chapter introduces the motivation behind this thesis, emphasizing the need for a modern and efficient web platform. It defines the problem statement and discusses the challenges associated with website development for research laboratories. Additionally, this chapter provides an overview of the thesis structure, guiding the reader through its key sections.

The second chapter reviews the technologies required to develop the website. It explores content management systems (CMS), front-end frameworks, API integration techniques, and deployment strategies. This chapter is a foundation for understanding the technological landscape that influenced the project's design decisions.

The third chapter details the project's technical implementation. It describes the chosen CMS, its integration with the database, and the selected front-end technology. Furthermore, it covers back-end development, including LinkedIn API automation for content sharing. Deployment strategies are examined in depth. The chapter concludes with an architectural analysis supported by diagrams illustrating the system's structure.

The fourth chapter focuses on setting up and restoring the website. It provides a step-by-step guide for deploying the system, managing database backups, handling API secrets, and configuring the reverse proxy. This chapter ensures the website can be efficiently maintained and redeployed when necessary.

The fifth chapter showcases the functional aspects of the developed system through a series of demonstrations. It presents key features of the content management system, front-end application, and LinkedIn integration and highlights their usability and performance.

The final chapter outlines ideas for future improvements, including further automation, enhanced deployment strategies, and additional integrations to expand the system's capabilities.

Chapter 2 – Overview of Technologies Needed

Modern web development relies on various technologies that simplify content management, enhance performance, and ensure scalability. This chapter overviews the essential technologies used in developing the website, focusing on Content Management Systems (CMS), front-end technologies, API integrations, and deployment strategies. Each section discusses key concepts, available solutions, and their impact on a project's implementation.

2.1 CMS

A Content Management System (CMS) manages digital content, allowing users to create, edit, and publish information without requiring advanced programming skills. CMS platforms vary in structure and functionality, ranging from traditional monolithic solutions to modern headless and external services-based architectures. This section explores different CMS types, their advantages and disadvantages.

2.1.1 What is a CMS?

A Content Management System (CMS) 11] is a software application that facilitates the creation, management, and modification of digital content without requiring extensive technical knowledge. CMS platforms are widely used for developing and maintaining websites, enabling users to manage text, images, videos, and other forms of content through a user-friendly interface. They provide functionalities such as content organization, user management, and media handling while often supporting collaboration between multiple users.

Traditional web development requires direct HTML, CSS, and JavaScript coding, as well as back-end development for handling databases and server logic. A CMS abstracts much of this complexity, allowing users to build and manage dynamic websites with minimal coding effort. This makes CMS solutions particularly valuable for businesses, educational institutions, and research laboratories, where non-technical users must update and maintain content efficiently.

Content **M**anagement **S**ystems come in different forms, each catering to specific needs and use cases. The two primary types of CMS are Traditional (**Monolithic**) CMS and **Headless** CMS [2]. Those can be self-hosted or managed by External Services.

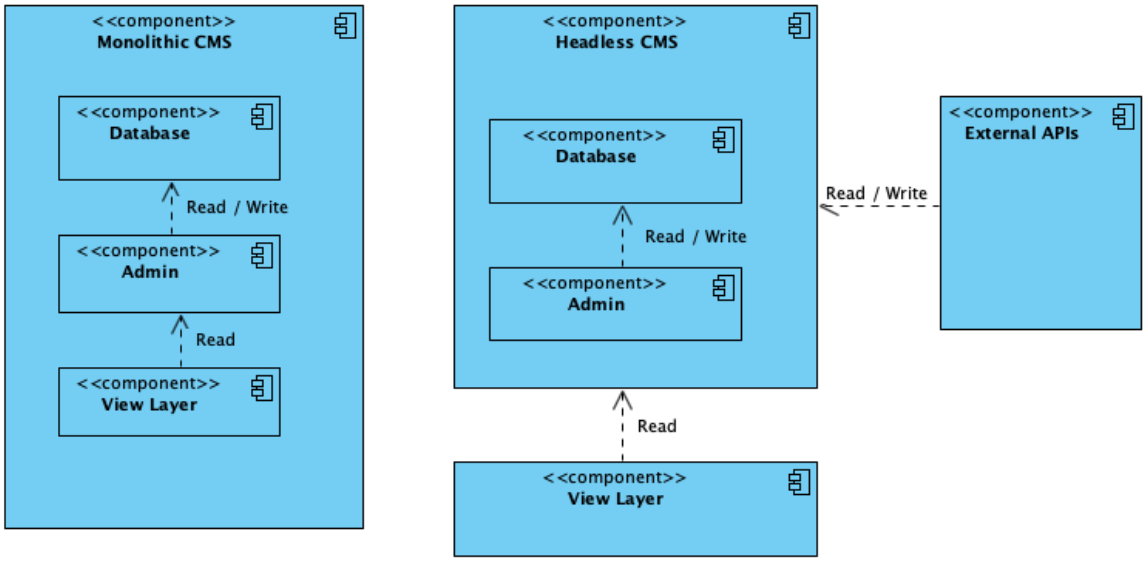

Figure 2.1: Monolithics vs Headless CMS [2]

2.1.2 Monolithic CMS

A traditional CMS, also known as a coupled CMS, combines the back-end (content storage, management) and front-end (content presentation) into a single system. Popular examples include WordPress, Joomla, and Drupal.

Advantages:

- All-in-one solution: Includes content management, design, and hosting in a single package.
- User-friendly: Often comes with a visual editor, making content updates easy for non-technical users.
- Large plugin ecosystem: Provides extensive customization options through plugins and themes.

Disadvantages:

- Limited flexibility: The predefined front-end may restrict how content is displayed.
- Performance issues: The tightly integrated front-end and back-end can lead to slower performance.
- Harder to scale: As content grows, performance optimization can become challenging.

2.1.2.1 WordPress

WordPress is the most widely used monolithic CMS, powering over 40% of all websites on the internet. It is an open-source platform based on PHP and MySQL, providing a user-friendly content management system that enables individuals and businesses to create and manage websites with minimal technical expertise.

Key Features:

- Large Community and Extensive Documentation – A vast ecosystem of developers and users provides ongoing support and resources.
- Rich Plugin and Theme Repository – Thousands of plugins and themes allow for extensive customization.
- User-Friendly Interface – A graphical administration panel (Dashboard) makes content management accessible to non-technical users.
- SEO Optimization – Popular plugins such as Yoast SEO enhance search engine visibility.
- Built-in User Management System – Supports multiple roles and permissions for different users.
- E-Commerce Support – Integration with WooCommerce enables the creation of online stores.

Disadvantages:

- Security Vulnerabilities – Due to its popularity, WordPress is frequently targeted by cyberattacks.
- Performance Limitations – Extensive use of plugins can lead to slower page loading times.
- Limited Flexibility in Architecture – While plugins extend functionality, WordPress remains a monolithic system, making it less suitable for integration into modern microservices-based architectures.

2.1.2.2 Joomla

Joomla is another open-source CMS that balances ease of use and flexibility. It is based on PHP and MySQL and is designed for users who require more advanced content management capabilities while still maintaining usability. Joomla is particularly well-suited for multilingual websites and complex permission structures.

Key Features:

- Advanced User Management – Offers a sophisticated system for managing user roles and permissions.
- Multilingual Content Support – Unlike WordPress, Joomla provides built-in support for multilingual websites without the need for additional plugins.
- Template and Extension System – Provides more customization flexibility than WordPress, allowing for advanced design modifications.
- Caching and Performance Optimization – Includes features to improve website speed and efficiency.

Disadvantages:

- Steeper Learning Curve – More complex than WordPress, requiring a higher level of technical proficiency.
- Smaller Community and Fewer Extensions – Compared to WordPress, Joomla has a more limited selection of plugins and themes.
- More Challenging Maintenance – Updates and modifications require more technical knowledge.

2.1.2.3 Drupal

Drupal is a highly flexible and secure CMS, making it the preferred choice for large-scale enterprises, government websites, and organizations with complex content structures. It is designed to handle advanced workflows, complex permissions, and structured content.

Key Features:

- Strong Security Measures – Used by government institutions due to its robust security architecture.
- Advanced Content and User Management – Supports highly customizable content types and taxonomy systems.
- Highly Scalable and Performance-Oriented – Suitable for large websites with high traffic demands.
- Multilingual Capabilities – Offers built-in support for multiple languages without the need for plugins.
- Headless CMS Capabilities – Can be used as a decoupled CMS, providing flexibility for frontend development.

Disadvantages:

- Complexity and Learning Curve – Requires advanced technical skills, making it less suitable for beginners.
- Smaller Community and Limited Themes – Compared to WordPress, Drupal has fewer ready-to-use themes and extensions.
- Resource-Intensive – Requires more server resources, leading to higher hosting costs.

2.1.3 Headless CMS

A Headless CMS separates the back-end (where content is stored and managed) from the front-end (how the content is displayed). It provides content via an API, allowing developers to use different front-end frameworks. Examples include Strapi, Contentful, and Sanity.

Advantages:

- Greater flexibility: Content can be delivered to multiple platforms (web, mobile apps, IoT).
- Better performance: The front-end is independent, allowing for optimized rendering strategies.
- Developer-friendly: The usage of modern frameworks provides complete control over how content is displayed.

Disadvantages:

- Requires technical knowledge: Unlike traditional CMS, Headless CMS solutions do not provide built-in themes or visual editors.
- More complex setup: Requires additional effort to integrate with a front-end framework.

2.1.3.1 Strapi

Strapi is an open-source headless CMS built with JavaScript and powered by Node.js. It enables developers to create, manage, and expose content through RESTful or GraphQL APIs. Strapi is designed to be self-hosted, providing complete control over data storage and content structure.

Key Features:

- Self-hosted & Open-source – Unlike many other headless CMS platforms, Strapi allows users to fully control their backend and database.
- GraphQL & REST API Support – Provides built-in GraphQL and REST API endpoints for flexible content delivery.
- Role-based Access Control (RBAC) – Allows for fine-grained permissions, making it suitable for enterprise applications.
- Customizable Admin Panel – Developers can extend and modify the admin panel to fit project needs.
- Supports Multiple Databases – Compatible with SQLite, PostgreSQL, MySQL, and MongoDB.

Disadvantages:

- Requires Self-Hosting – Users need to manage their own infrastructure, unlike SaaS solutions like Contentful.
- More Configuration Required – Initial setup can be more complex compared to cloud-based headless CMS options.

2.1.3.2 Contentful

Contentful is a cloud-based SaaS headless CMS, designed for enterprises and businesses that require a scalable, API-driven content platform. Unlike Strapi, Contentful is fully managed and provides a rich set of features for content modeling and API integrations.

Key Features:

- Cloud-based & Fully Managed – No need for self-hosting or infrastructure management.
- Rich API Support – Offers REST and GraphQL APIs for content retrieval.
- Flexible Content Modeling – Users can define custom content types and fields.
- Built-in Content Delivery Network (CDN) – Ensures fast content delivery worldwide.
- Integration with Third-Party Services – Supports integrations with services like AWS, Netlify, and Vercel.

Disadvantages:

- Pricing – Contentful follows a subscription-based model, which can be costly for large projects.
- Limited Customization – Unlike Strapi, users cannot modify the backend as it is a managed service.

2.1.3.3 Sanity

Sanity is a real-time, API-first headless CMS with a strong focus on structured content and collaboration. It provides a flexible content management approach that enables dynamic content delivery across multiple platforms.

Key Features:

- Real-Time Collaboration – Supports simultaneous editing and content updates, similar to Google Docs.
- Highly Customizable Studio (Sanity Studio) – Provides a flexible UI for content management that developers can extend.
- GraphQL & GROQ Support – Uses GROQ (Graph-Relational Object Queries) in addition to GraphQL for optimized content queries.
- Content Delivery API (Sanity CDN) – Ensures fast and efficient content distribution.
- Developer-Friendly – Provides custom schemas, making it easy to structure content efficiently.

Disadvantages:

- Learning Curve – GROQ is not as widely used as GraphQL or REST, requiring additional learning.
- Limited Self-Hosting Options – Primarily a cloud-based solution, with limited on-premise deployment capabilities.

2.2 Front-end Technologies

Front-end technologies are essential for creating modern, dynamic, and interactive web applications. They define how users interact with a website and influence performance, responsiveness, and overall user experience. This section explores the key front-end technologies used in this project, including the core web development languages, rendering strategies, and techniques for optimizing search engine visibility. By selecting the right front-end stack, the project ensures a fast, scalable, and SEO-friendly website.

2.2.1 What are front-end technologies?

The front-end, also known as the client-side, is the portion of a web application that users interact with directly through their web browsers. As web applications become more sophisticated, front-end development has evolved to accommodate growing demands for responsiveness, scalability, and user experience.

2.2.1.1 Core Front-end Technologies

HTML (HyperText Markup Language) provides the structural foundation of web pages. Organizes elements such as headings, paragraphs, images, and links, and Forms the backbone of all web applications and ensures accessibility.

CSS (Cascading Style Sheets) is responsible for styling and layout. Defines styles such as colors, typography, spacing, and responsive design. Also, Modern CSS tools like Flexbox, Grid, and CSS Variables enhance styling efficiency.

JavaScript, is a programming language that enables interactivity and dynamic content updates. It allows developers to manipulate the DOM (Document Object Model) and respond to user actions. Over time, JavaScript has become a dominant force in front-end development, with numerous frameworks and libraries designed to simplify and streamline development processes.

2.2.1.2 Modern Javascript Frameworks

These frameworks enable the development of SPAs, which dynamically update content within a single HTML page rather than requiring full-page reloads. This is achieved through client-side routing, state management, and virtual DOM manipulation, allowing for a smooth, app-like user experience.

React [3] is component-based JavaScript library developed by Facebook. Efficient Virtual DOM updates improve rendering performance. Suitable for building Single Page Applications (SPAs) and Progressive Web Apps (PWAs).

Vue.js [4] is A progressive framework with an easy learning curve and flexible integration. Supports reactive data binding and component-based architecture. Ideal for projects of all sizes, from small applications to enterprise-level platforms.

Angular [5] is A full-fledged MVC (Model-View-Controller) framework maintained by Google. Provides built-in dependency injection and TypeScript support. Best suited for large-scale, enterprise applications.

2.2.1.3 Supporting Technologies

Beyond core technologies and frameworks, front-end development involves various supporting tools that optimize performance and workflow:

- CSS Preprocessors (SASS, LESS): Extend CSS capabilities with variables, mixins, and nested syntax.
- Build Tools (Webpack, Vite, Parcel): Bundle and optimize code for better performance.
- State Management Libraries (Redux, Pinia, Vuex): Handle complex application state across components.
- Component Libraries (Bootstrap, Tailwind CSS, Material UI): Provide pre-styled UI components for faster development.

2.2.2 Search Engine Optimization (SEO)

Search Engine Optimization (SEO) [6] is the process of improving a website's visibility on search engines like Google. A well-optimized site ensures its content is easily discoverable and ranked higher in search engine results, leading to increased traffic.

SEO is crucial for modern web development, as search engines act as primary gateways for users seeking information. Proper SEO implementation enhances a website's usability, accessibility, and credibility. It encompasses various strategies, including keyword optimization, structured data usage, performance optimization, and user experience enhancements.

Front-end technologies directly impact SEO through how content is rendered, structured, and served to search engine crawlers. Choosing an appropriate rendering strategy, such as Static Site Generation (SSG) or Server-Side Rendering (SSR), can significantly enhance a website's search engine ranking.

2.2.2.1 SEO Factor for Rankings

Below are the key SEO factors that influence the rank of a website in search engine results:

- Technical SEO – Ensures that search engines can crawl and index the site efficiently. This includes optimizing robots.txt and XML sitemaps and ensuring proper canonicalization.
- On-Page SEO – Optimizes content, headers, meta tags, and images for relevance and readability.
- Off-Page SEO – Relates to external factors like backlinks, domain authority, and social media engagement.
- Performance & UX (User Experience) – Includes page load speed, mobile-friendliness, and secure connections (HTTPS).
- Rendering Strategy – The method of delivering content impacts how search engines index the site

2.2.2.2 Structured Data & XML Sitemaps

Structured data helps search engines understand the context of website content. Schema.org [7] markup enables rich search results like breadcrumbs, FAQs, and product ratings.

An XML sitemap provides a structured list of URLs, guiding search engines to crawl a website efficiently. Below is an example of an XML sitemap and a meta tag structure for SEO:

sitemap.xml > urlset

```xml
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>https://www.example.com/</loc>
    <lastmod>2024-02-19</lastmod>
    <changefreq>daily</changefreq>
    <priority>1.0</priority>
  </url>
  <url>
    <loc>https://www.example.com/blog</loc>
    <lastmod>2024-02-18</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>https://www.example.com/contact</loc>
    <lastmod>2024-02-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.5</priority>
  </url>
</urlset>
```

Figure 2.2: XML Sitemap Example

2.2.2.3 SEO Meta Tags

SEO meta tags are snippets of HTML code that provide metadata about a webpage to search engines and social media platforms. These tags help search engines understand the content of a page, influencing how it appears in search results.

Key meta tags include the title tag, which defines the page's title shown in search results. The meta description summarizes the page's content and meta keywords and lists relevant terms.

Open Graph (og) tags optimize how links are displayed on social media platforms. Properly using meta tags enhances a website's visibility, click-through rate, and overall SEO performance.

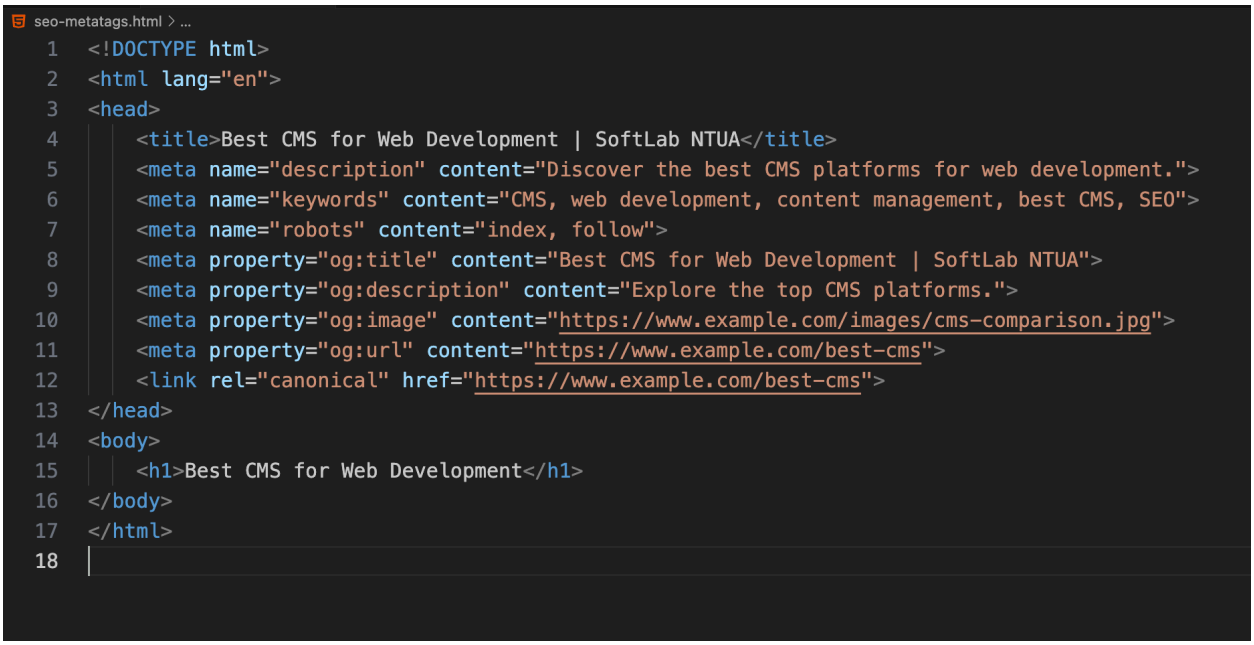

Figure 2.3: SEO Metatags Example

2.2.3 Static Site Generation

Static Site Generation (SSG) [8] is a rendering method in which web pages are pre-built at compile time and serve as static files to the user. This approach ensures fast loading times and better SEO performance, as search engine crawlers can easily index pre-rendered pages.

SSG generates HTML files during the build process, eliminating the need for frequent server-side computations. This makes it ideal for content-heavy websites like blogs, documentation sites, and e-commerce product pages. Since the pages are already compiled, they load almost instantly, improving Core Web Vitals, a key metric for SEO ranking.

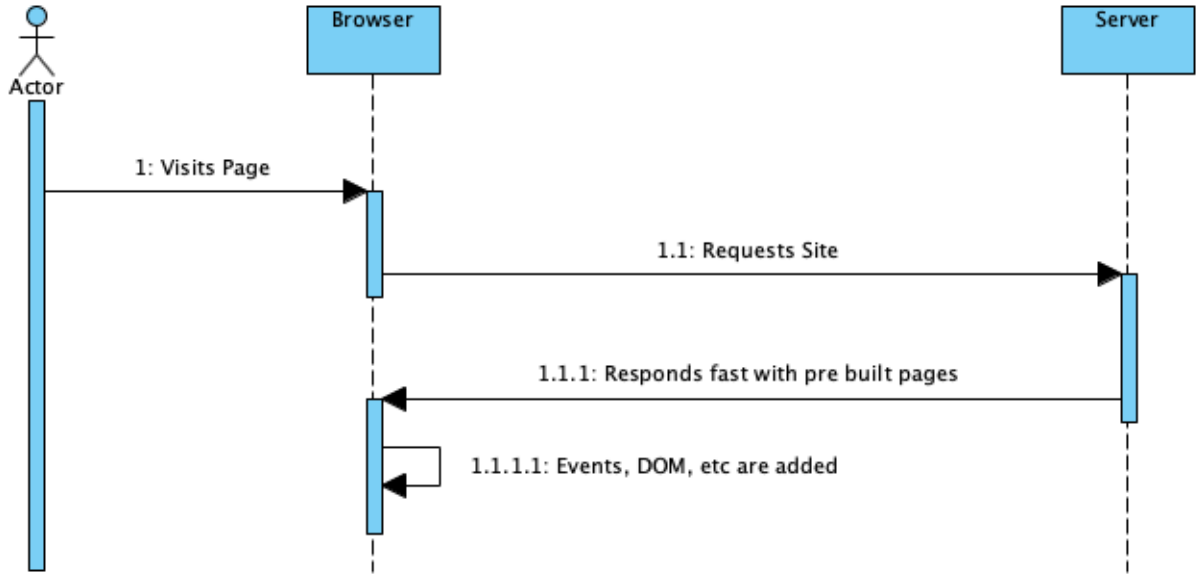

Figure 2.4: Static Site Generation

2.2.4 Server-side Rendering

Server-side rendering (SSR) [9] is a rendering strategy where pages are dynamically generated on the server before being sent to the client. Unlike SSG, which pre-builds pages, SSR processes each request separately, ensuring that users always receive up-to-date content.

SSR significantly enhances SEO because search engine crawlers receive fully rendered HTML content, making it easier to index than client-rendered pages. This is particularly useful for websites that rely on dynamic content.

However, SSR introduces additional server load, as each request requires processing before delivering the response.

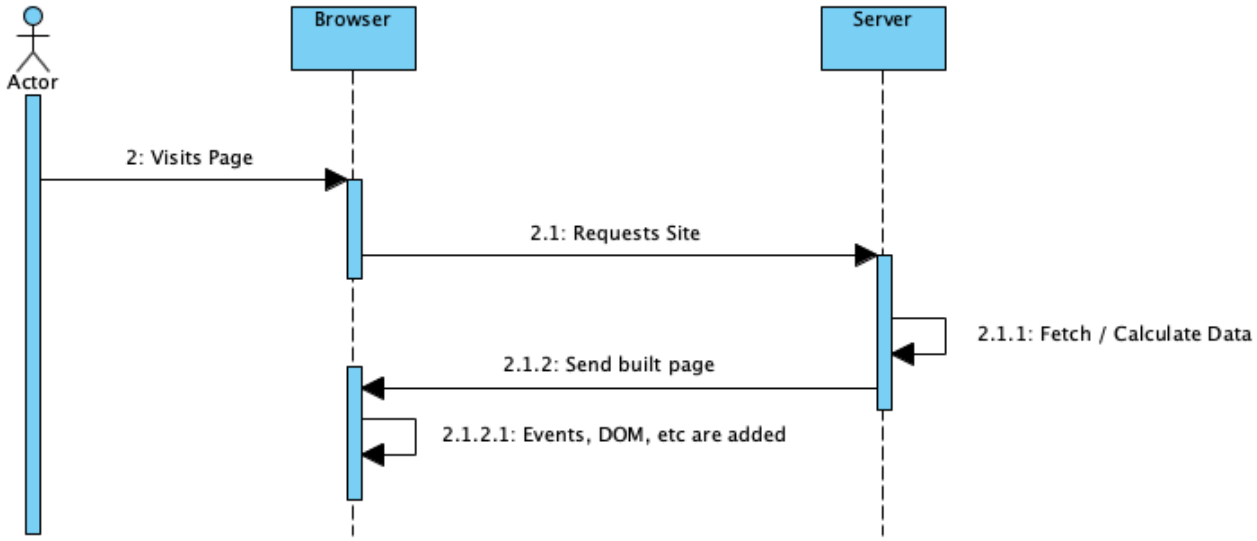

Figure 2.5: How Server Side Rendering Works

2.1.5 Client-side Rendering

Client-side rendering (CSR) [10] is a technique in which JavaScript renders content directly in the browser rather than on the server. In CSR-based applications, the browser initially loads a minimal HTML file along with JavaScript, which then fetches and renders the necessary content dynamically.

While CSR provides a smooth, app-like experience, it poses challenges for SEO. Since search engine crawlers primarily rely on pre-rendered HTML, CSR applications often result in poor indexing and ranking due to delayed content loading. To mitigate this, techniques like server-side hydration or prerendering can be used to improve search engine visibility.

CSR is well-suited for highly interactive applications, such as dashboards and web-based tools, where user experience takes priority over search visibility.

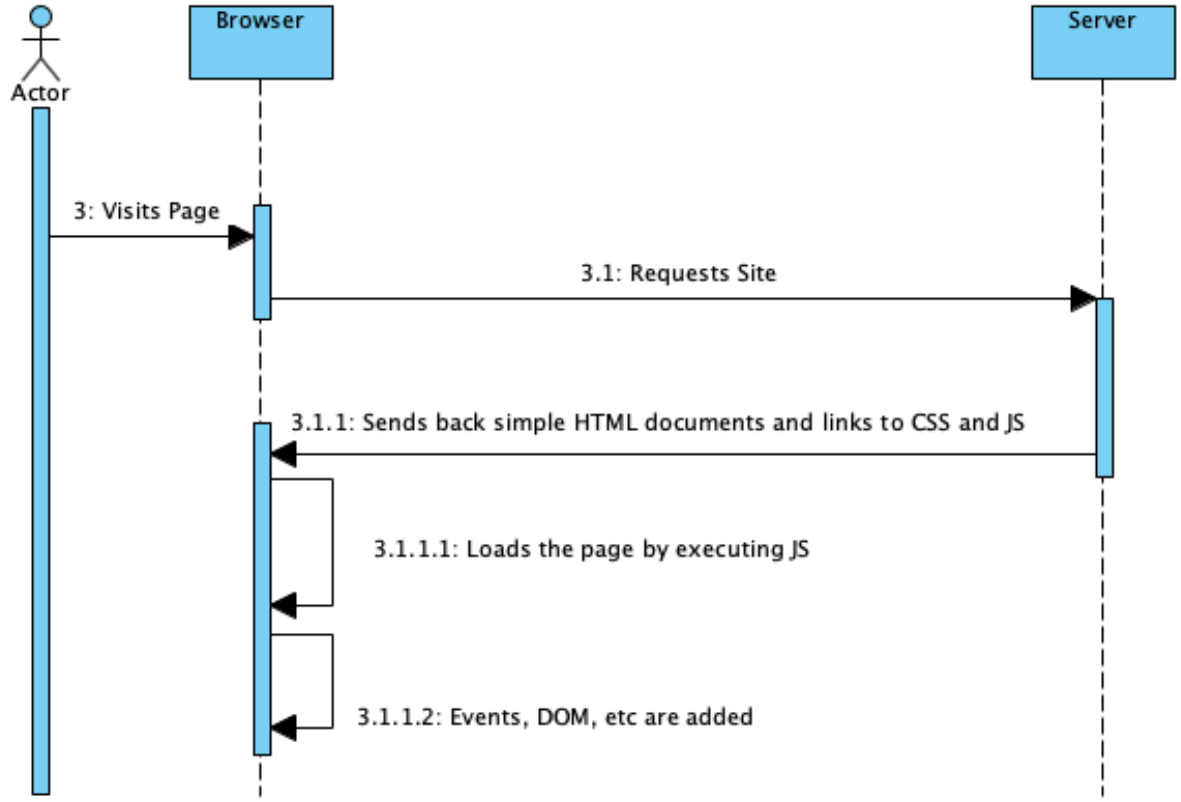

Figure 2.6: How Client Side Rendering Works

2.3 API Integration and Automation

By integrating APIs, applications can exchange data, extend functionalities, and automate workflows without manual intervention. This section explores the fundamentals of APIs, their integration processes, and automation mechanisms such as webhooks, which allow real-time event-driven interactions between services. These technologies enhance efficiency, scalability, and interoperability in web development.

2.3.1 What are APIs?

An Application Programming Interface (**API**) [11] is a set of rules and protocols that allows different software applications to communicate with each other. APIs define how requests and responses should be structured, enabling data exchange between systems. They act as intermediaries, allowing applications to access features or data from external services without needing direct access to their underlying code or database.

APIs are widely used in web and mobile development. They enable applications to fetch data, interact with third-party services, and enhance functionality. Common examples include retrieving weather data from an external service, integrating a payment gateway, or accessing social media platforms like Twitter or LinkedIn from within another application.

APIs come in different types, including **REST** (Representational State Transfer), which is most commonly used for web applications; **GraphQL**, which allows flexible data querying; and **SOAP** (Simple Object Access Protocol), which is often used in enterprise applications requiring strict security and structure.

2.3.2 What is an API Integration

API integration is the process of connecting two or more software applications through APIs, allowing them to share data and functionalities. This integration enables different systems to work together efficiently without manual intervention, improving automation, productivity, and user experience.

API integrations can be one-way (where data flows in a single direction) or two-way (where both applications exchange information).

2.3.3 Webhooks and Automation

Webhooks are an API mechanism that allows applications to automatically send real-time data to other systems when a specific event occurs. Unlike traditional API calls that require polling (repeatedly checking for updates), webhooks enable event-driven automation by pushing updates as soon as they happen.

For example, when a user submits a form on a website, a webhook can instantly send the data to a CRM system, triggering an automatic response or notification.

Webhooks play a crucial role in API integrations and automation, enabling the connection between different applications and workflow automation. They are commonly used in payment processing (e.g., notifying a system when a transaction is completed), CI/CD pipelines (e.g., triggering a deployment after a code push), and chatbot integrations (e.g., sending real-time alerts to messaging platforms like Slack or Discord)

2.4 Deployment

Efficient deployment is crucial in software development, ensuring applications run smoothly across different environments. This section explores key deployment strategies, including Docker containerization, which allows applications to run in isolated environments for consistency and scalability. Additionally, it covers the role of reverse proxies in improving security, performance, and load balancing, making deployment more efficient and reliable.

2.4.1 Deployment Basics

Deployment in software development refers to making an application available for use in a specific environment, such as production, testing, or staging. It involves transferring the developed application from a local or development environment to a live system where end users can access it. Deployment encompasses various tasks, including configuring servers, setting up databases, and ensuring the application runs efficiently and securely.

2.4.2 Docker

Docker plays a critical role in modern software deployment by providing containerization, which allows applications to run in isolated environments known as containers. A container packages an application along with all its dependencies, ensuring it runs consistently across different computing environments. This eliminates the common problem of an application that behaves differently in development, testing, and production due to discrepancies in system configurations.

By using Docker, developers can create lightweight, portable, and self-sufficient containers that can be deployed on any system supporting Docker. This greatly simplifies deployment, as applications do not need to be manually configured on each server. Docker also enhances scalability by enabling applications to run efficiently across multiple environments, making it easier to distribute workloads.

2.4.3 Reverse Proxy

A reverse proxy [12] is a server that sits between client devices and back-end servers, directing client requests to the appropriate back-end service. It is vital in deployment because it improves security, performance, and scalability.

A reverse proxy manages load balancing in deployment, distributing incoming requests across multiple back-end servers to prevent any single server from becoming overwhelmed. This ensures high availability and improved response times, especially for applications that experience large volumes of traffic. Reverse proxies are also instrumental in caching frequently requested content, reducing the load on back-end servers and improving application performance.

Additionally, reverse proxies enhance security by hiding the internal architecture of a system, preventing direct access to back-end services, and providing SSL termination to encrypt and decrypt traffic.

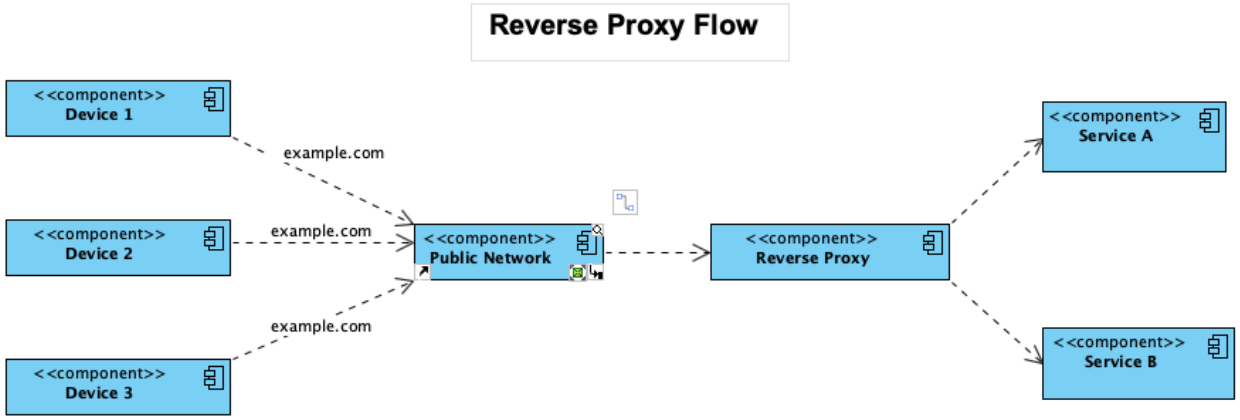

Figure 2.7: Reverse Proxy Visualization

Chapter 3 – Implementation

The implementation phase focuses on the practical development of the application, detailing how various technologies were integrated to achieve a modern, scalable, and efficient system. This chapter covers the CMS setup, database integration, front-end and back-end development, API automation, deployment strategies, and the system architecture. Each section explores the technical choices made and their role in achieving the project's objectives.

3.1 Strapi CMS

Strapi was chosen as the CMS for developing the application due to its flexibility, modern architecture, and ease of integration with the Next.js front-end. Several key factors make Strapi the most suitable option.

3.1.1 Open-Source and Self-Hosted

Unlike SaaS-based CMS platforms, Strapi is open-source, providing complete control over data and customization. It allows hosting on private servers, ensuring better security and compliance with university policies.

3.1.2 Headless Architecture

Strapi provides a headless CMS solution that offers RESTful and GraphQL API. This enables the integration of modern front-end frameworks, allowing them to optimize content rendering and SEO performance.

3.1.3 Ease of Use and Content Management

Strapi comes with an admin panel that allows non-technical users to manage content efficiently. The role-based access control feature ensures proper user management, making it ideal for research teams with different content responsibilities.

3.1.4 Schema Builder - Visualization

Strapi provides a visual interface for defining content structures. This feature allows administrators to view, create, modify, and organize content types using an intuitive drag-and-drop editor. Unlike traditional CMS platforms that require manual schema definitions in code, Strapi simplifies this process by offering a graphical representation of the database structure.

3.1.4 GraphQL Support

Unlike many traditional CMS solutions, Strapi provides built-in GraphQL support, improving API efficiency by allowing clients to request only the needed data.

3.1.5 Flexibility in Database Choices

Supports multiple databases, including SQLite, PostgreSQL, and MySQL, making it adaptable to different project needs.

3.1.6 Scalability and Performance

The decoupled architecture optimizes performance and scalability as the project grows. Content delivery is independent of the front-end, reducing load times and improving user experience.

3.1.7 Conclusion

Considering the research-oriented nature of the project, Strapi's flexibility, self-hosting capability, and integration with modern front-end frameworks make it the ideal CMS for this project. It provides complete control over content, ensures high performance, and enables long-term maintainability, aligning perfectly with the project's objectives.

3.2 Database

A database is the backbone of any modern web application. It provides structured and efficient data storage, retrieval, and management.

A database is necessary for the website to store and organize content dynamically, manage data, and support various functionalities such as API interactions. Content management would be static and inefficient without a database, requiring manual intervention for every update.

Given the project's scope, a well-structured database ensures data consistency, accessibility, and security while facilitating integration with the CMS and front end. Below, we will explore whether SQLite is the best option.

3.2.1 SQLite

SQLite is a lightweight, self-contained database engine that requires minimal setup and operates without a separate database server. It follows the relational database model and supports SQL-based queries for structured data management.

Unlike traditional database management systems such as MySQL or PostgreSQL, SQLite stores the entire database as a single file on disk, making it highly portable and easy to manage. Due to its serverless architecture, SQLite is particularly well-suited for applications that do not require extensive concurrent write operations or complex transaction handling.

3.2.2 Strapi Integration

One key advantage of SQLite is its integration with Strapi. Strapi supports multiple database options, including SQLite, PostgreSQL, and MySQL, allowing for flexibility in deployment. SQLite's native support in Strapi makes it an optimal choice, enabling rapid prototyping, testing, and fast deployment without requiring an external database server.

3.2.3 Advantages of SQLite

SQLite was selected for the website primarily due to its versatility, simplicity, and ease of maintenance. Its key advantages include:

- Flexibility: SQLite's single-file format makes it easy to back up, transfer, and restore without complex database configurations.
- Ease of Use: The absence of a separate database server eliminates the need for additional setup and administration, reducing system overhead.
- Portability: SQLite databases can be easily embedded into applications, making development, deployment, and debugging straightforward.
- Reliability: Despite its lightweight nature, SQLite ensures compliance with ACID (Atomicity, Consistency, Isolation, Durability), guaranteeing data integrity.

In summary, SQLite's adoption for the website is a strategic choice that maximizes efficiency and simplicity while ensuring smooth content management and integration with the CMS. Its low overhead and ease of backup and restoration make it an optimal solution for the project’s requirements.

3.3 Front-end

The front-end plays a pivotal role in defining the user experience and ensuring the overall performance of a web application. For the development of the website, Next.js [13] was chosen as the primary front-end framework due to its ability to deliver high-performance, SEO-optimized, and scalable applications. This section explores why Next.js is the best option for this project.

3.3.1 Next.js

Next.js is a Javascript-based framework that simplifies the development of modern web applications by providing built-in features for server-side rendering (SSR), static site generation (SSG), and API routes. Additionally, Next.js supports incremental static regeneration, which enables the website to update content efficiently without requiring a full rebuild. Thus, it is particularly suitable for dynamic content-driven applications.

3.3.2 Search Engine Optimization – Static Site Generation

Search engine optimization (SEO) is a fundamental aspect of any modern website, influencing visibility, traffic, and user engagement. Next.js significantly enhances SEO by offering server-side rendering and static site generation features, ensuring search engines can efficiently index content.

Static Site Generation (SSG) is particularly beneficial for SEO. It allows pages to be generated at build time and served as static HTML files, reducing server load and ensuring a fast, stable browsing experience. By pre-rendering content, SSG eliminates the delays associated with fetching data on the client side, improving time-to-first-byte (TTFB) and overall performance metrics.

The integration of SSG with Next.js ensures that the website remains highly performant while delivering fully optimized, indexable content for search engines.

3.3.3 Multilingual

Given that the SoftLab NTUA is a research laboratory, multilingual support is essential to provide accessibility in multiple languages. Next.js simplifies internationalization (i18n) through its built-in support for locale-based routing and compatibility with libraries such as next-i18next [14].

The i18n functionality in Next.js enables efficient language detection, locale switching, and dynamic content rendering based on user preferences. This allows the website to serve different audiences while maintaining optimal performance through static site generation.

Additionally, by structuring translations efficiently, Next.js ensures that multilingual support remains scalable and maintainable as new content is added over time.

3.3.4 Integration with Strapi

Next.js integrates with Strapi, the selected headless CMS, enabling efficient content management and dynamic data retrieval. Strapi serves as a back-end content repository, allowing administrators to manage text, images, and structured data, which the Next.js front end can fetch during the build process.

Next.js supports on-demand revalidation, which enables real-time content updates without requiring complete site redeployment. This integration ensures a proper content workflow, where updates made in Strapi are reflected in the front-end without unnecessary performance overhead.

3.4 Back-end

This section examines the rationale behind Next.js API routes to integrate the LinkedIn API within the application. Given the need for automated content distribution, the integration enables administrators to publish announcements and updates directly to the SoftLab NTUA LinkedIn page without manual intervention.

3.4.1 Next.js Back-end

Next.js API routes provide built-in back-end functionality within the framework, allowing the creation of server-side logic without requiring a separate back-end service. These API routes operate within the same Next.js project, facilitating a transition between front-end and back-end operations.

Each API route is defined within the **routes/api** directory and is accessible via an HTTP endpoint. These routes can handle server-side operations, such as fetching external data, processing form submissions, authenticating users, and interacting with third-party APIs.

Next.js API routes serve as intermediaries between Strapi and LinkedIn's API for this project. When an administrator publishes a new announcement in Strapi, a webhook triggers a Next.js API route, which processes the data and posts it to LinkedIn. This approach eliminates the need for additional back-end infrastructure while ensuring a streamlined, maintainable solution for LinkedIn integration.

3.4.2 Linkedin API integration

The LinkedIn API provides functionalities for automating content posting on behalf of a LinkedIn organization page. Through LinkedIn's OAuth-based authentication mechanism, an application can obtain permission to create and share content directly under an organization's name rather than an individual user's profile.

To post on a LinkedIn page, an application must authenticate and obtain an access token with the necessary permissions. Once authenticated, LinkedIn allows posts to be created in multiple formats using its community management Posts API [15].

In this project, the Next.js back-end is configured to send requests to LinkedIn's API using an **urn:li:organization** identifier, which ensures that all posts appear under the SoftLab NTUA LinkedIn organization page rather than an individual account. This setup guarantees that announcements, events, and updates are consistently published on the official LinkedIn page without requiring manual posting from administrators.

3.4.3 Automated Post on Linkedin

LinkedIn posts are automated using a structured workflow that eliminates manual intervention while ensuring content delivery. When an administrator creates an announcement or relevant content within Strapi, they have the option to publish it directly to LinkedIn.

Once the content is published, Strapi triggers a webhook [16] that sends a notification to the Next.js back-end. The webhook contains structured data related to the newly published content, including its title, description, and associated media. The Next.js API route then processes this data, formats it according to LinkedIn's API requirements, and makes a request to post it on the LinkedIn page.

This workflow ensures that updates from the website are consistently shared on LinkedIn without requiring additional manual steps. By automating the posting process, administrators can maintain an active LinkedIn presence while focusing on content creation rather than social media management.

Section 5 presents a demonstration of this automated workflow, showcasing how an announcement is created in Strapi, processed by the Next.js back-end, and published on LinkedIn.

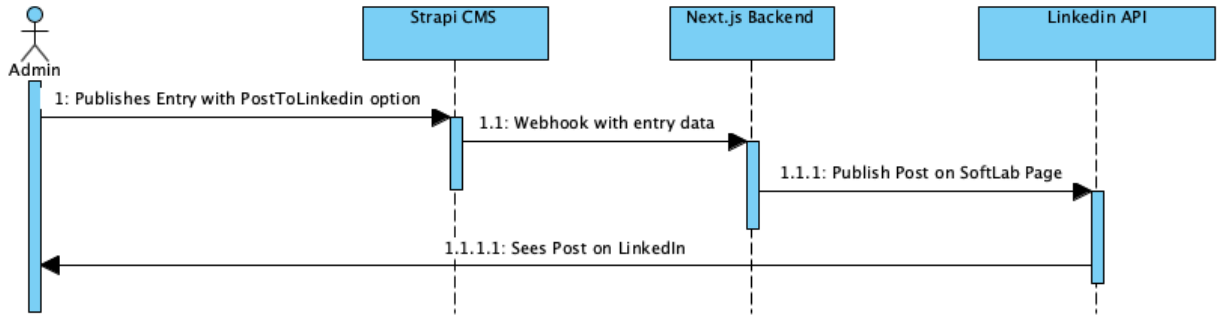

Figure 3.1: Sequence Diagram of LinkedIn Integration

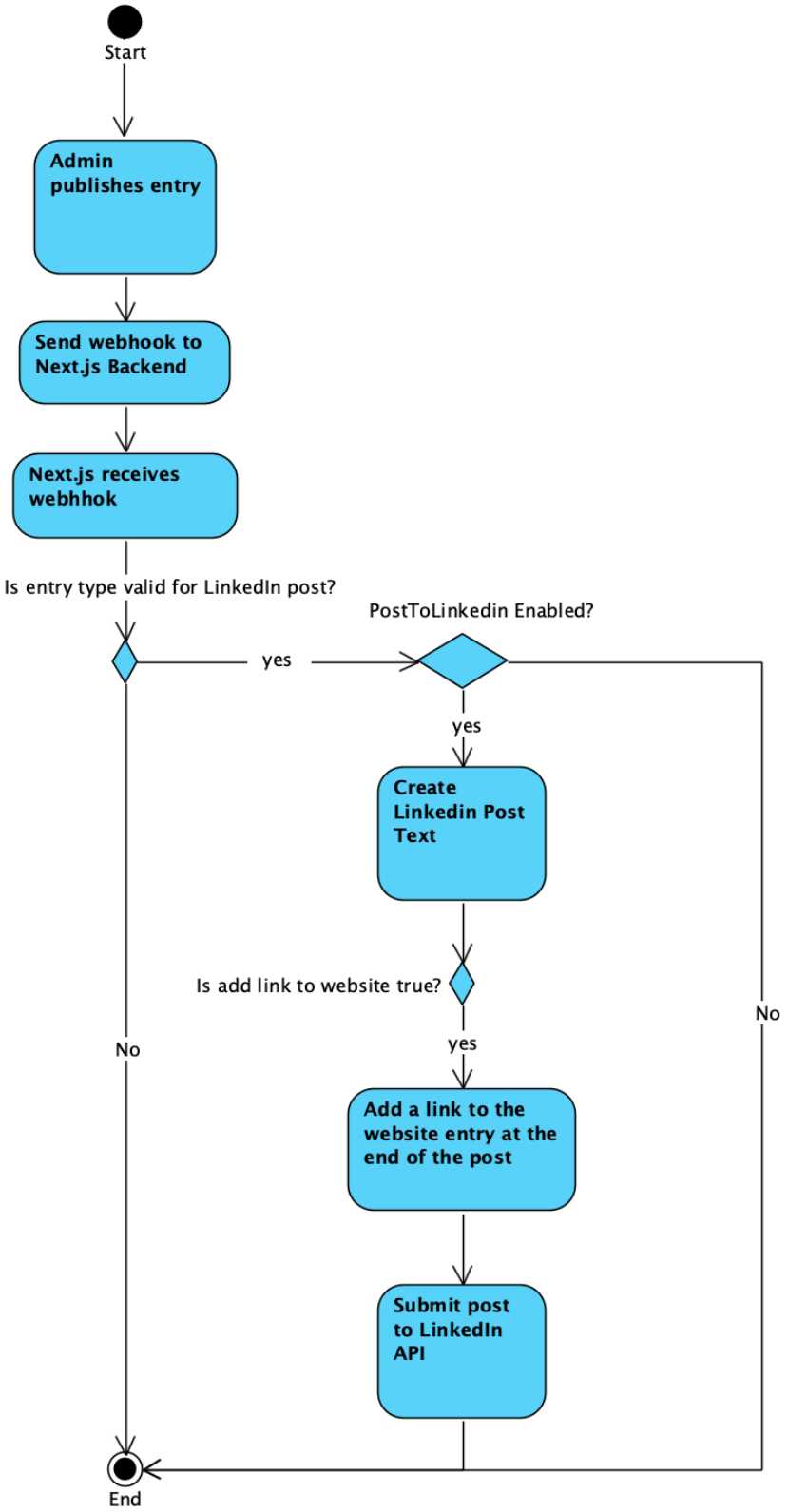

Figure 2.1: LinkedIn Integration Activity Diagram

3.5 Deployment

In this section, we examine the deployment strategy for the application and the reasoning behind the selection of specific technologies. A well-structured deployment approach is essential for ensuring stability, security, and scalability while maintaining ease of maintenance.

3.5.1 Lightweight, Reliable, and Flexible Deployment

The deployment approach is designed to be lightweight, ensuring minimal resource overhead while maintaining high efficiency.

Reliability is a key consideration, as the system must remain operational with minimal downtime, even under varying workloads. Flexibility is also a fundamental requirement, allowing for easy updates, scalability, and adaptability to future infrastructure or application architecture changes.

By selecting technologies that align with these principles, the website benefits from a robust deployment strategy that ensures long-term maintainability and efficiency.

3.5.2 Deployment in Physical Servers

A critical decision in the deployment process was to opt for physical servers owned by NTUA instead of cloud-based infrastructure. Security concerns and institutional policies primarily drove this decision. While offering scalability and convenience, cloud-based solutions often introduce vendor lock-in, where services become dependent on proprietary ecosystems. This limits future flexibility and increases long-term costs.

Additionally, hosting on NTUA's infrastructure aligns with the university's established practices. Maintaining control over the deployment environment ensures better data security. Furthermore, self-hosting eliminates reliance on external service providers, granting total control over server configurations, updates, and optimizations. This approach ensures deployment remains cost-effective, secure, and aligned with institutional standards.

3.5.3 Reverse Proxy Server

To further optimize performance and security, Nginx [17] was implemented as a reverse proxy. A reverse proxy manages incoming requests and directs them to the appropriate services.

It acts as an additional security layer, preventing direct access to back-end services and mitigating potential security threats such as DDoS attacks and unauthorized access attempts. Nginx provides caching, which allows frequently requested content to be stored and served quickly, reducing server load and improving response times.

Also, Nginx simplifies SSL/TLS certificate management, ensuring secure connections without requiring direct SSL handling within the application.

3.6 Content Updates

This section examines how content updates are handled. Since the website is built and served as static files, updating the CMS does not automatically reflect changes on the front-end. Unlike dynamic websites, where content updates are immediately visible, static sites require a mechanism to regenerate or refresh content. Two primary approaches are used to ensure that updates from Strapi are correctly reflected in the front-end: manual rebuilds of the front-end and Next.js Incremental Static Regeneration (ISR).

3.6.1 Rebuilds

One approach to updating content is to rebuild the front-end and redeploy the application. Since the website is generated as static files at build time, a full rebuild ensures that the latest data from Strapi is included in the front-end. This process involves the following steps:

- Generating a new frontend build that incorporates the latest content updates from Strapi.
- Stopping the old instance running the previous front-end version.
- Deploying and restarting the updated instance with the new front-end build.

This method guarantees that all content updates are fully reflected but has some limitations. Since rebuilding and redeploying the entire front-end requires server resources and time, it is not an instant update process. To address this, the rebuild process can be scheduled periodically or manually triggered when significant updates are made in the CMS.

3.6.2 Next.js Incremental Static Regeneration Strategy

To improve efficiency and reduce the need for frequent full rebuilds, the Next.js Incremental Static Regeneration (ISR) strategy is utilized. ISR allows specific pages to be automatically re-generated in the background without requiring an entire site rebuild. This ensures that updated content can be served dynamically while maintaining the performance benefits of static generation.

ISR sets a revalidation interval, which defines how often a page should check for updates. When a request is made for a specific page:

- If the page is still valid within the revalidation period, the cached version is served instantly.
- If the page is stale (past its revalidation period), Next.js triggers a background regeneration process while still serving the existing cached version to users.
- Once the new page version is generated, it automatically replaces the old version, ensuring updated content without downtime.

This method balances static site performance and content freshness, making it particularly effective for news, announcements, and frequently updated website sections. Unlike full rebuilds, ISR enables content updates to be reflected without stopping and redeploying the entire front-end.

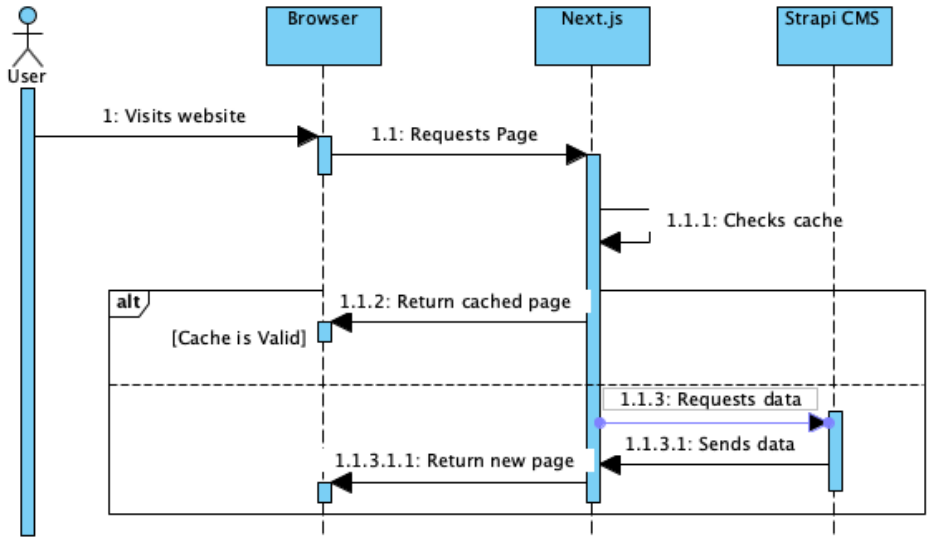

Figure 3.3: Sequence Diagram of ISR

3.7 Security

Security is fundamental to any web application, but it becomes even more critical for a research-oriented website. Research websites often contain sensitive data, unpublished work, and institutionally managed content that must be safeguarded against unauthorized access, data breaches, and cyber threats. Ensuring a secure architecture protects both the website's integrity and its content's credibility.

A multi-layered approach is followed to achieve a robust security framework.

3.7.1 Server – Client Decoupling

One key security principle adopted in this project is decoupling the front-end and back-end. Unlike monolithic architectures, where all application logic is handled within a single system, the application employs a headless CMS (Strapi) for content management and a Next.js front-end for content delivery. These components communicate via secure API endpoints, ensuring the front-end remains independent of back-end logic and database interactions.

Decoupling enhances security by minimizing the attack surface. Since the front-end does not have direct access to the database, potential threats such as SQL injection attacks and direct database exploits are significantly reduced. All back-end services are protected behind authentication layers, allowing only authorized API requests.

Another advantage of this architecture is secure content delivery. The front-end serves pre-rendered static pages, meaning no direct database queries or back-end logic are exposed to users. This makes it more resistant to DDoS [18] (Distributed Denial of Service) attacks and other automated attacks targeting dynamic server-side processin

3.7.2 Security Gateway – Reverse Proxy

The deployment incorporates a reverse proxy (Nginx) as a security gateway, adding another layer of protection between users and services. A reverse proxy serves as an intermediary that processes and forwards incoming requests, shielding the application components from direct exposure to the internet.

Using Nginx as a reverse proxy enhances security in multiple ways:
Traffic Filtering & Access Control:

- The proxy can enforce strict access controls, allowing only requests from trusted sources and blocking malicious traffic.
- SSL/TLS Termination: It handles secure HTTPS connections, ensuring all communication is encrypted and protected.
- Rate Limiting & DDoS Mitigation: It can limit the number of requests from a single IP, preventing automated attacks and brute-force attempts.

Placing Nginx as the first point of contact mitigates potential threats before they reach the services, ensuring secure API access and controlled resource allocation.

3.8 Architecture of Application

This section presents the system's architectural design, outlining its key components and their interactions. The following UML Component and Deployment Diagrams illustrate the application's structure, deployment environment, and communication flow.

3.8.1 Component Diagram

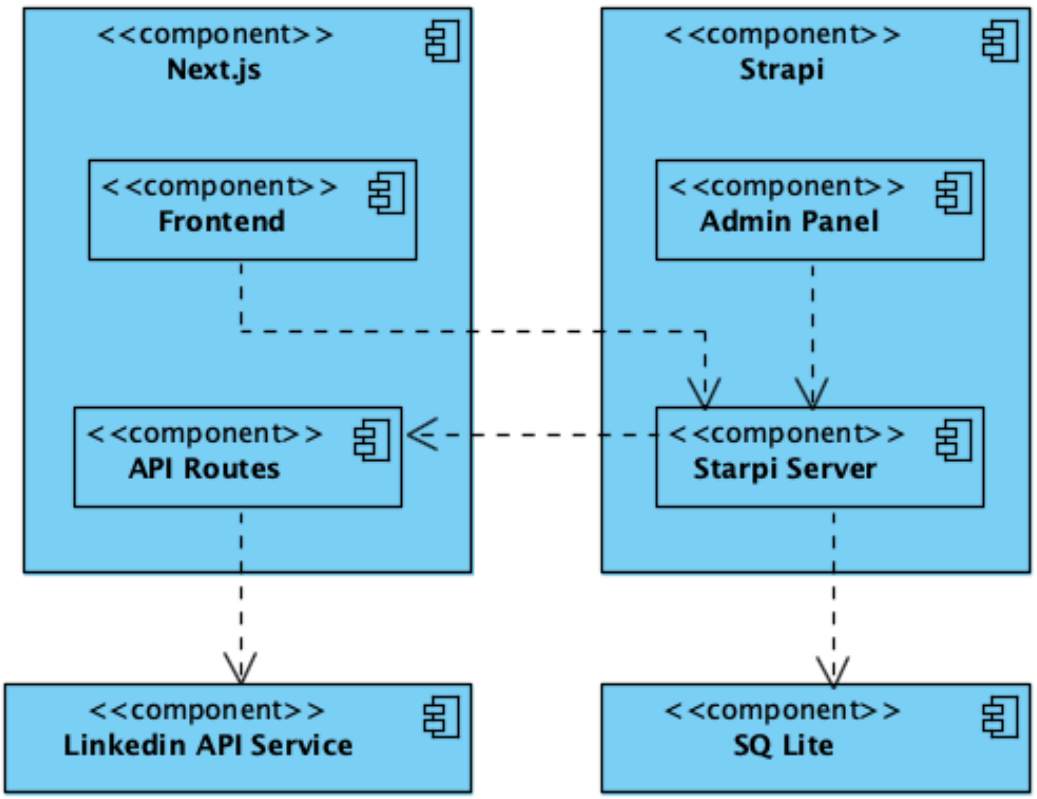

Figure 3.4: Component Diagram

3.8.2 Deployment Diagram

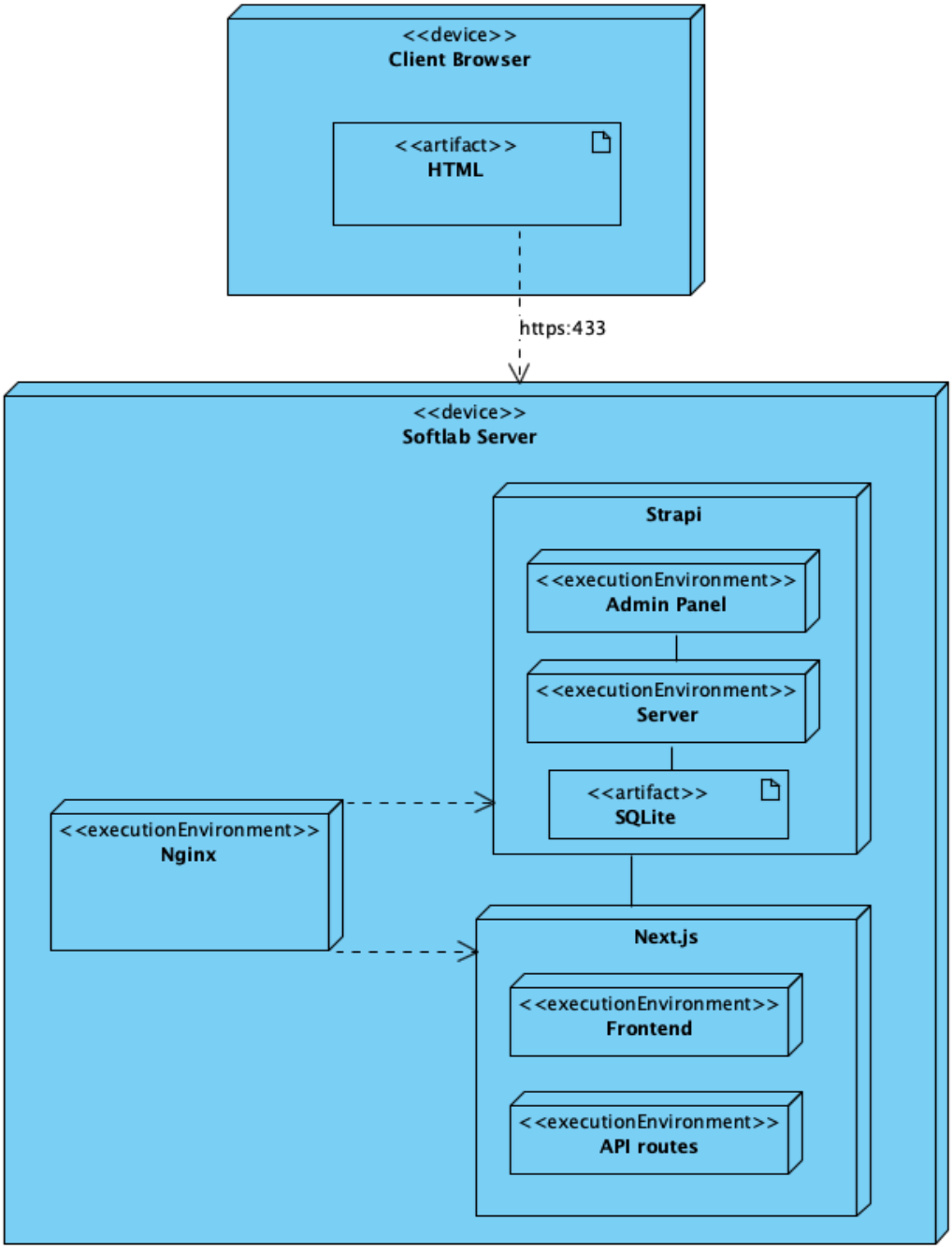

Figure 3.5: Deployment Diagram

Chapter 4 Starting or Restoring the Application

This chapter outlines the necessary steps for setting up or restoring the application. Ensuring a well-documented and structured restoration process is essential for maintaining system availability, minimizing downtime, and ensuring the integrity of the deployment environment.

4.1 Source Code

The first step in setting up or restoring the website is to fetch the source code from the repository and place it on the target machine. The source code contains the entire application, including the front-end, back-end, database configuration, and deployment scripts. Cloning the repository ensures that all required files are available and that the latest version of the codebase is deployed.

4.2 Data Backups

Restoring the database is a critical step in recovering the website. Since the website utilizes SQLite as its database management system, the restoration process involves retrieving the latest database backup file as well as all images located in the Strapi folder. These files should be placed in the correct paths, as specified in the source code documentation, to ensure that Strapi can properly access and serve the data.

SQLite is a file-based database, which simplifies the backup and restoration process since it only requires replacing the existing database file with the most recent backup. Additionally, restoring the Strapi images folder is essential to maintain the integrity of the website's media content. Once the database file and images are restored, Strapi automatically reads the database and accesses the associated media, ensuring that all content, configurations, and user data are available as they were prior to the restoration.

4.3 Secrets and Keys

After restoring the database, the secrets and API keys required for the application to function correctly must be configured. Environment variables must also be set for the front-end and back-end services to ensure they can securely interact with external services, databases, and authentication mechanisms.

These secrets are stored in **.env** files specific to each service, including sensitive information such as API keys, database connection strings, and authentication tokens. Proper handling of these credentials is essential to maintaining security and preventing unauthorized access to critical services.

4.4 Starting the Processes

The application processes must be initialized by launching both the front-end and back-end services. This includes installing dependencies, compiling the necessary components, and executing the services to ensure proper functionality. Once running, the services become accessible and ready to be managed by the reverse proxy.

4.5 Setting up the Reverse Proxy

The reverse proxy configuration files are in the source code and must be applied to the Nginx instance running on the target machine. This setup ensures that website requests are correctly routed to the front-end while API requests are forwarded to the back-end.

4.6 Pointing the Domain to the New Virtual Machine

The final step in the restoration process involves updating the domain name system (DNS) settings to point the website domain to the new VM hosting the application. This step ensures that website users are directed to the correct server.

Updating the DNS records usually involves modifying the A record to reflect the new VM's IP address. Once the changes propagate across the DNS system, the website becomes accessible under its domain name.

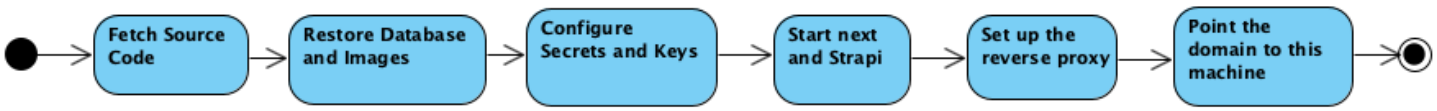

Figure 4.1: Configuring the Application Activity Diagram

Chapter 5 Demonstration of the App

This chapter presents a comprehensive walkthrough of the developed application, highlighting its core functionalities and user interface.

We will explore different aspects of the system, starting with the Content Management System (CMS), where administrators can manage content efficiently. After this, the frontend application will be showcased, highlighting key features such as homepage navigation, filtering mechanisms, and multilingual support.

Additionally, we will demonstrate the LinkedIn integration, showcasing how events are created and published directly from the admin panel. This demonstration will illustrate the system's effectiveness and usability in real-world scenarios.

5.1 CMS

In this section, we will explore the applications's administration panel. It can be accessed after logging in to the admin panel.

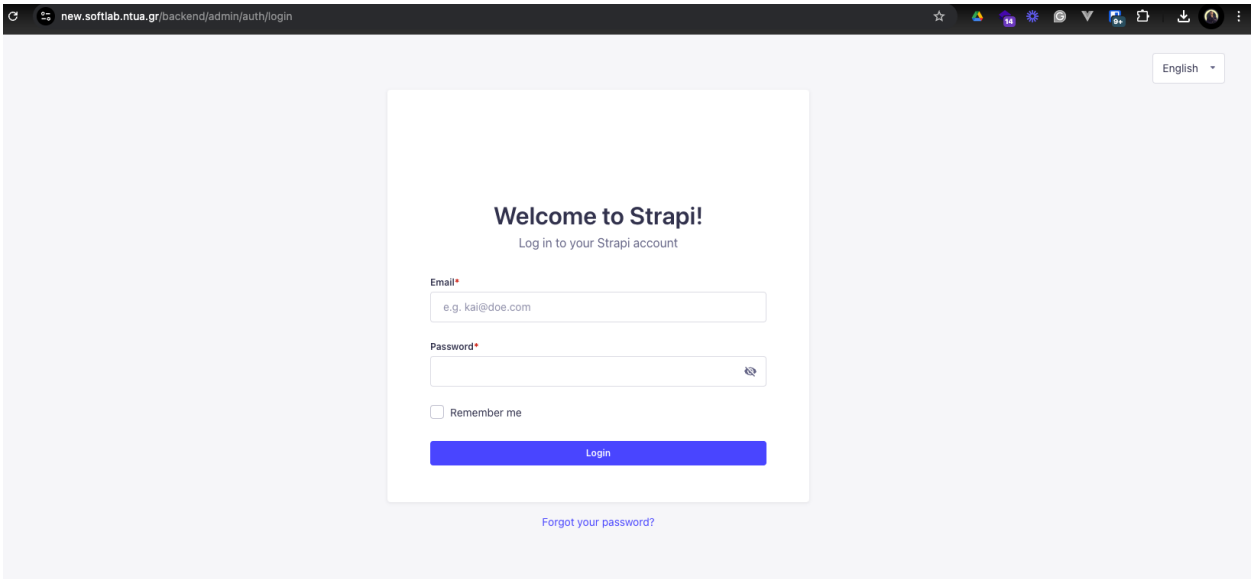

Figure 5.1: Login Page

5.1.1 Content Manager

The Content Manager [19] is accessible from the main navigation. Clicking on it opens a sub-navigation displaying two categories: collection types and Single types.

Each category contains the available collection and single content types created using the Content Type Builder beforehand. Administrators can create, manage, and publish content from these two categories.

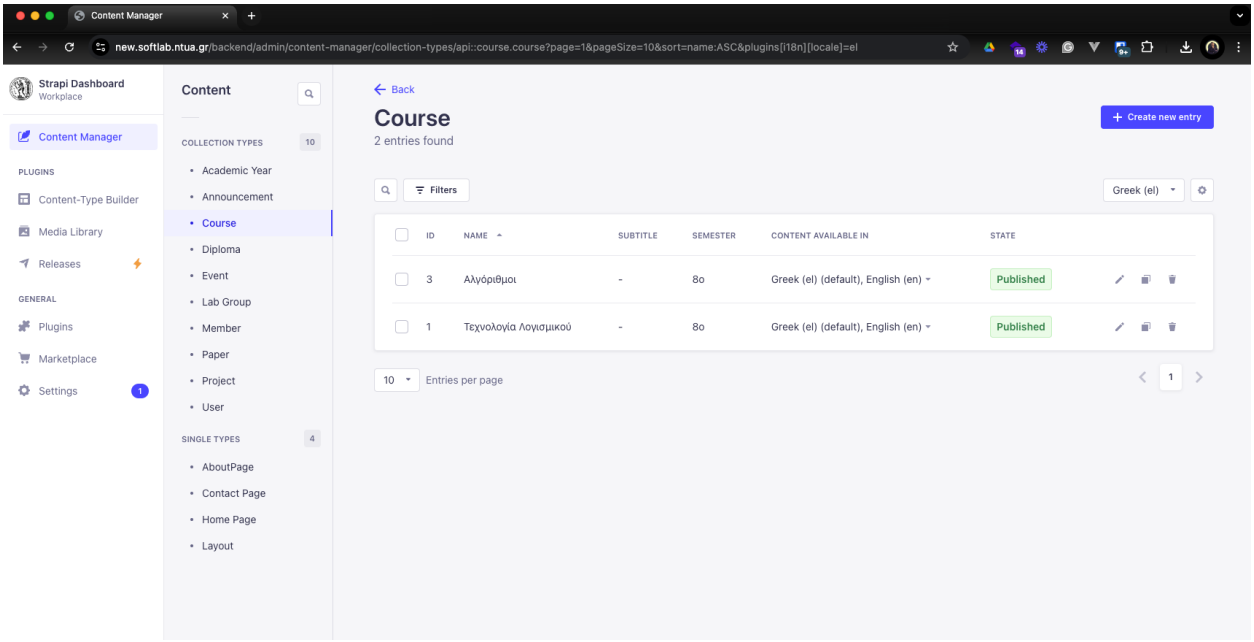

Figure 5.2: Content Manager

5.1.1.1 Creating an Entry

The Create new entry button is displayed at the top right of the list view interface. It allows the administrator to create a new entry for the selected collection type.

Clicking on the new entry button will redirect the administrator to the edit view, where he can add the latest entry's content.

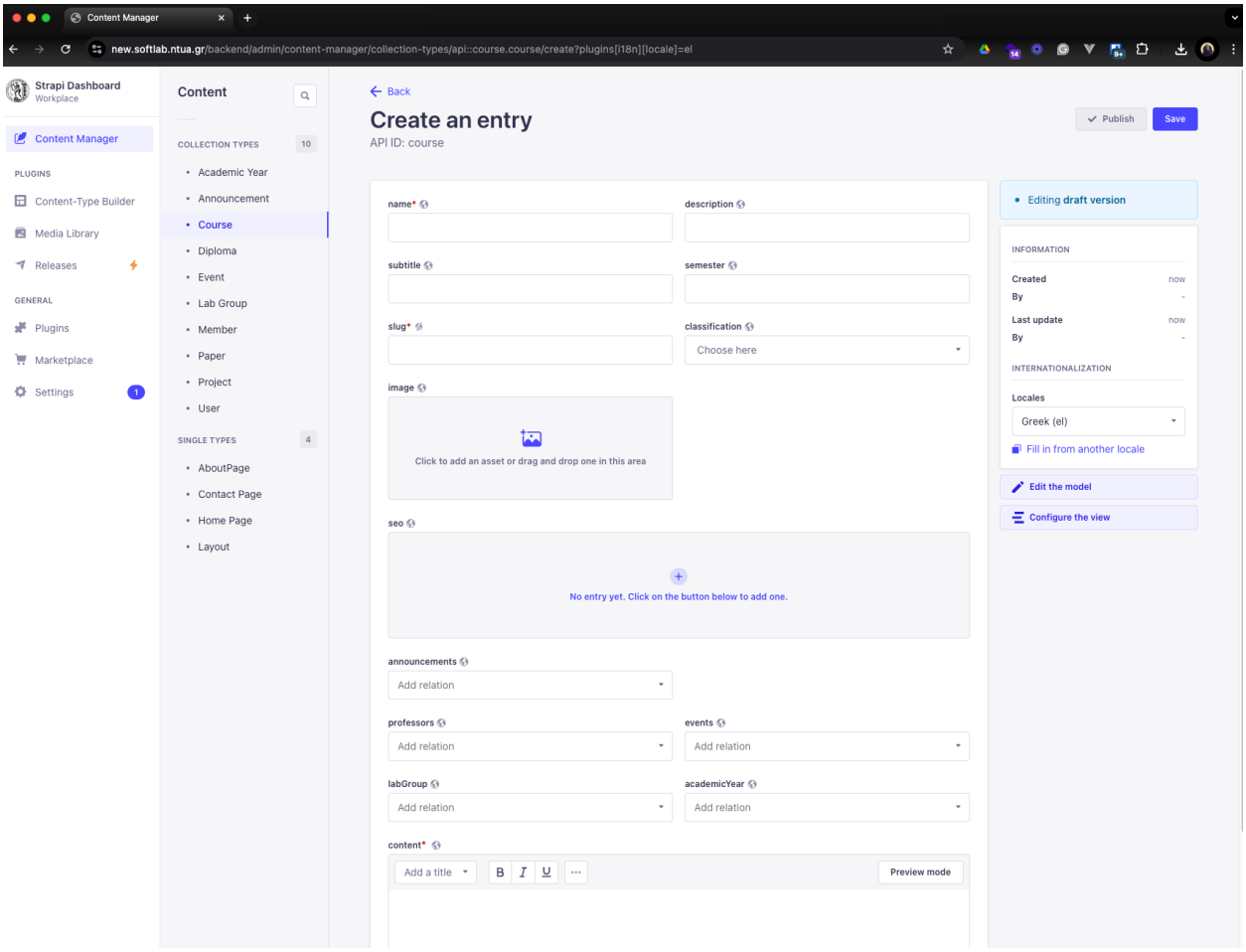

Figure 5.3: Creating a new course

5.1.1.2 Multilingual Support

The entry for the selected locale can be edited in the content manager editor. The locale is selected in the Locales dropdown [20].

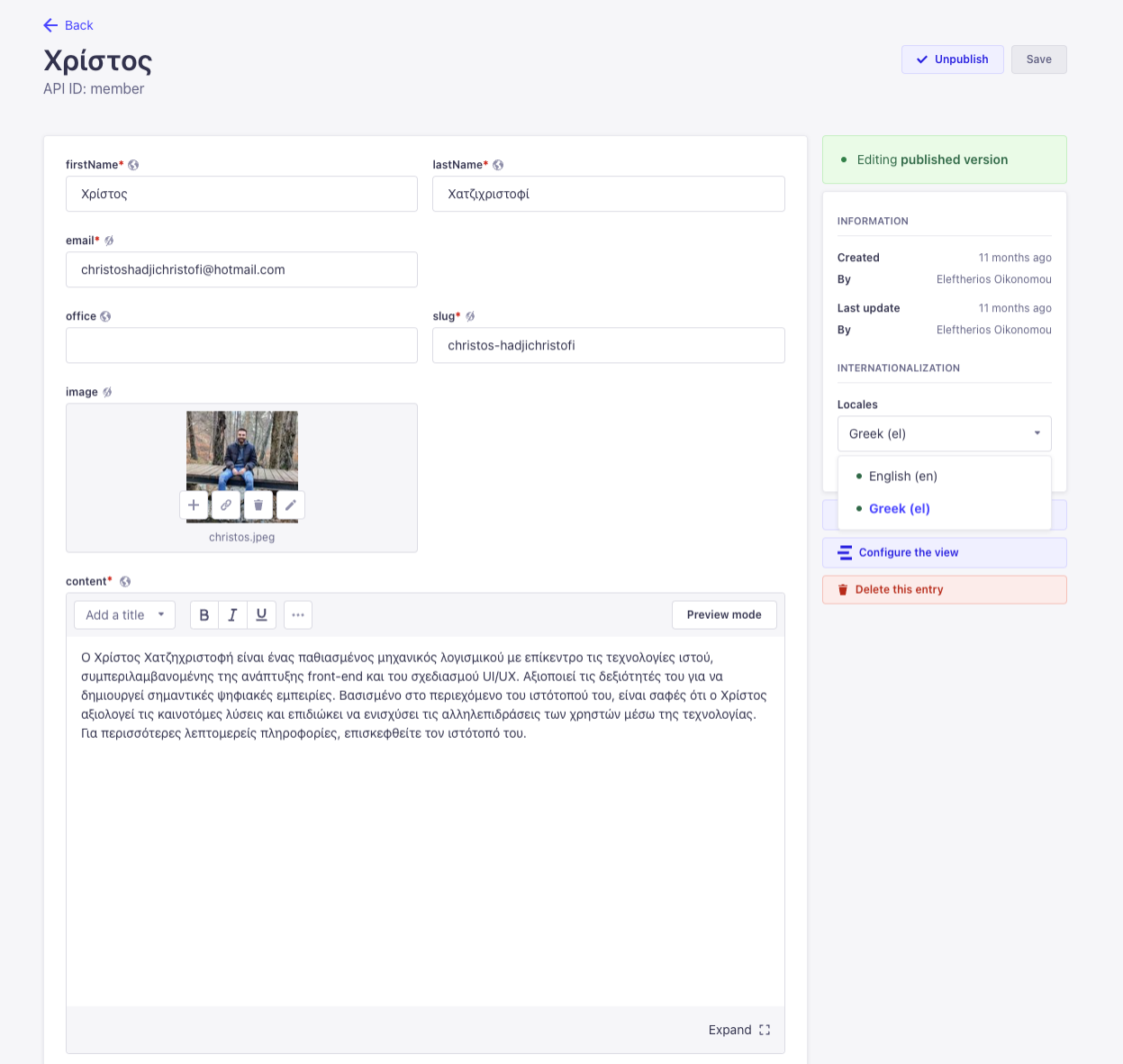

Figure 5.4: Member in Greek Locale

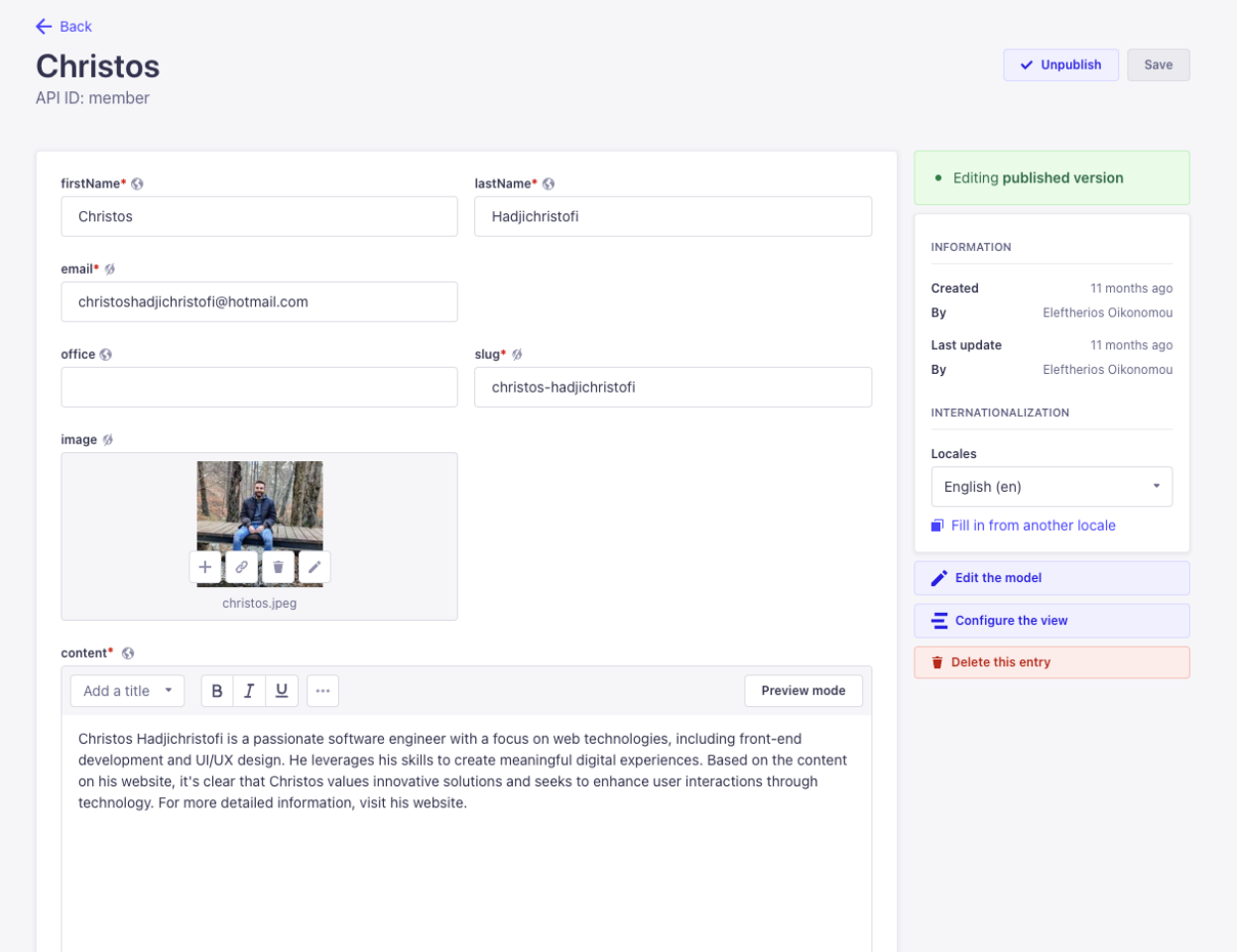

Figure 5.5: Member in English Locale

5.1.1.3 Publishing an Entry

When an entry is published, it will be shown in the front-end application when the next update is released. To publish an entry, click the publish button at the top right of the page.

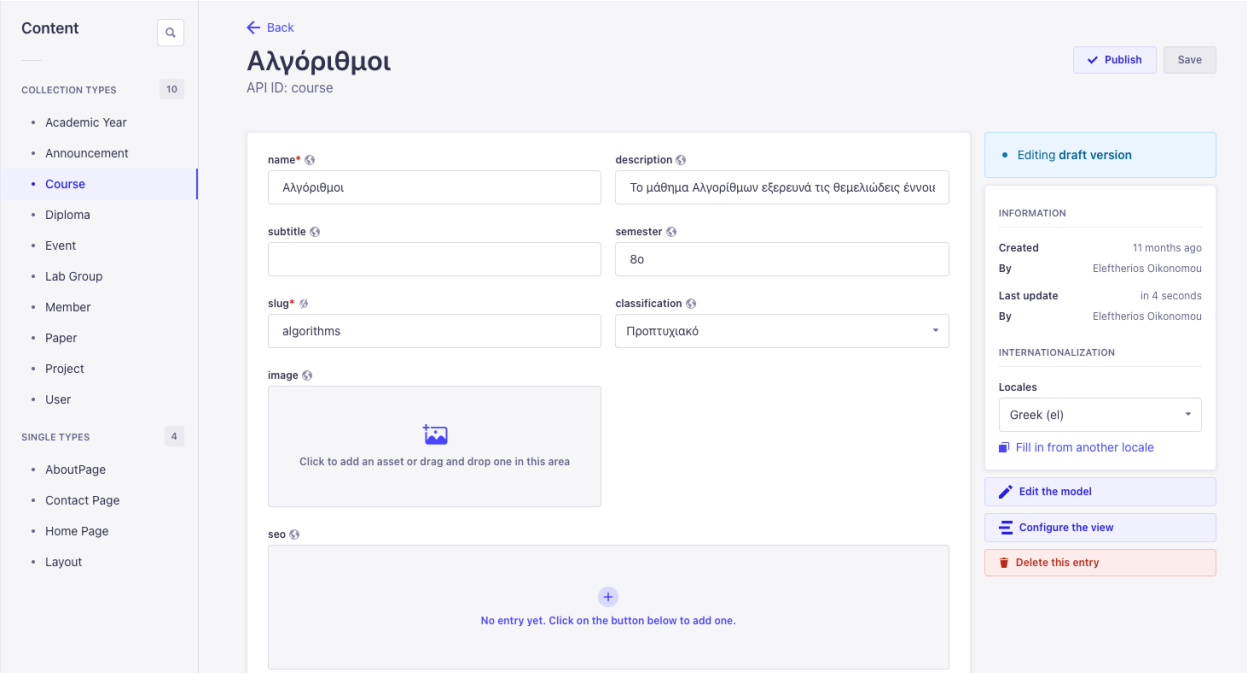

Figure 5.6: Publishing an entry

5.1.2 Content-type Builder

The Content-type Builder [21] is a core functionality of Strapi. It is always activated by default and cannot be deactivated. However, it is only accessible when the application is in a development environment.

Administrators can access the Content-type Builder from Content-type Builder in the main navigation of the admin panel.

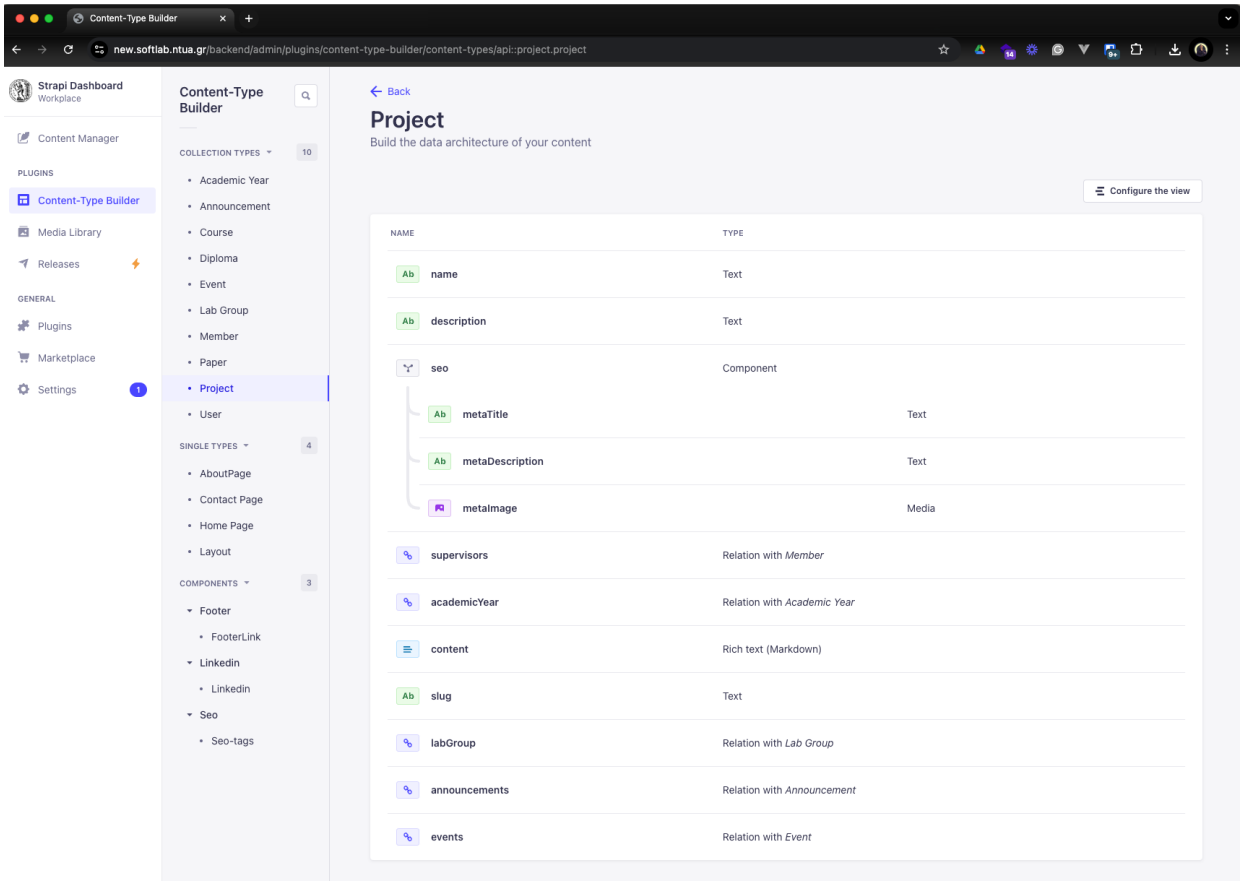

Figure 5.7: Content type builder

Administrators can create and manage content types, including collection types, single types, and components, from the Content-type Builder.

Collection types are content-types that can manage several entries. Single types are content-types that can only manage one entry. Components are data structures used in multiple collection types and single types.

5.1.3 Administrator Account Creation

To create a new administrator user [22], navigate to settings → Users → Invite User and fill in the form with the necessary details.

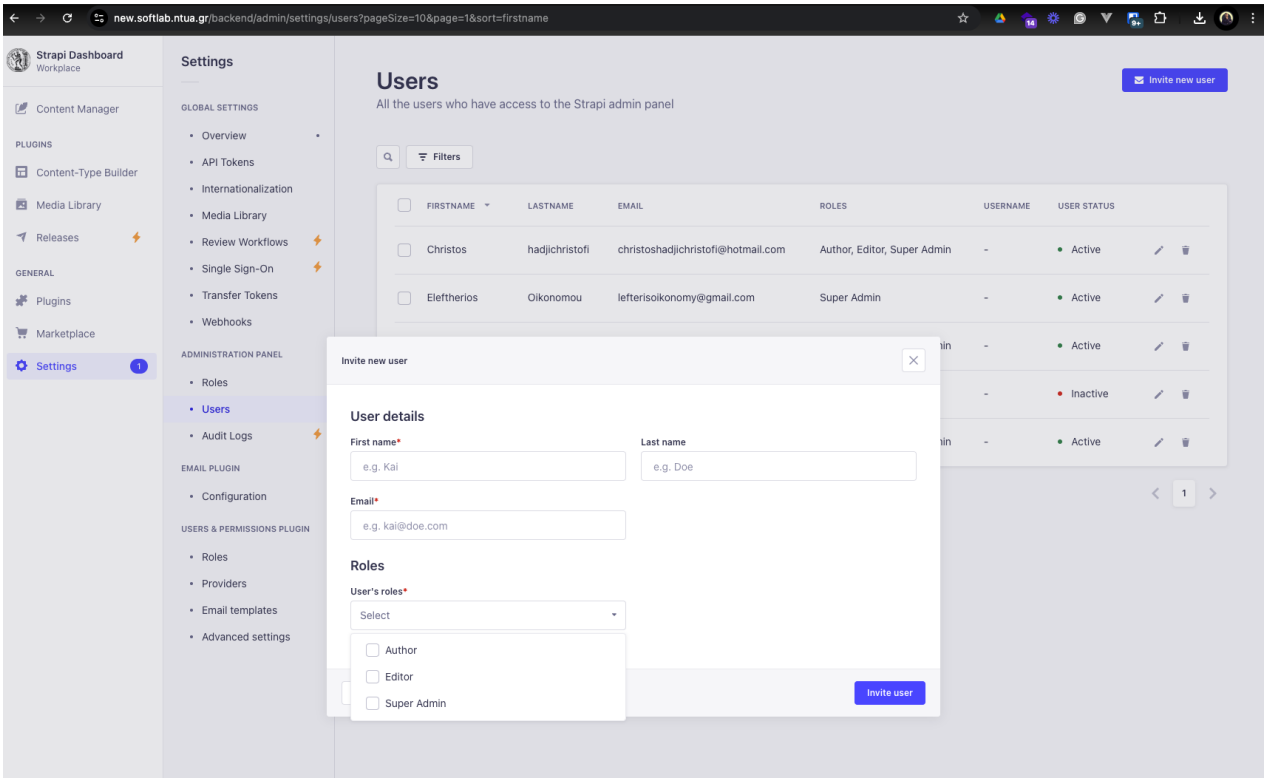

Figure 5.8: Creating a User

Then, a link will be created. Using this link, a new admin can be created with the permission and user details just submitted.

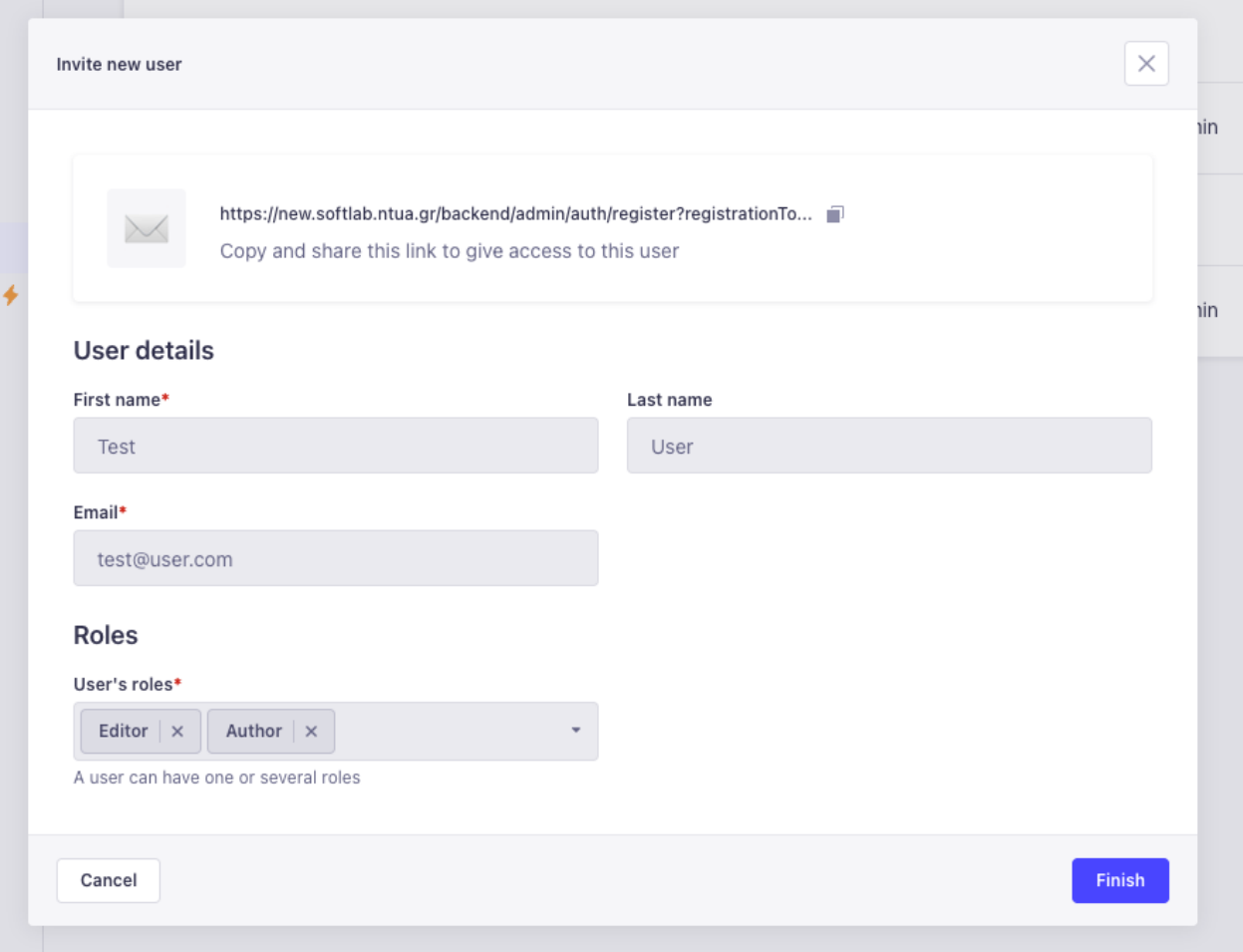

Figure 5.9: Link that can be used for new user registration

5.2 Front-end Application

The front-end application of the SoftLab NTUA website was developed with a focus on usability, modern design, and seamless integration with the content management system (CMS). The main objectives of the front-end include delivering a fast and intuitive user experience, ensuring accessibility, and supporting multilingual functionality.

5.2.1 Home Page

The home page is the primary gateway for users accessing the SoftLab NTUA website. It provides a structured and visually appealing layout that highlights essential information. All the content on the home page can be managed by the CMS.

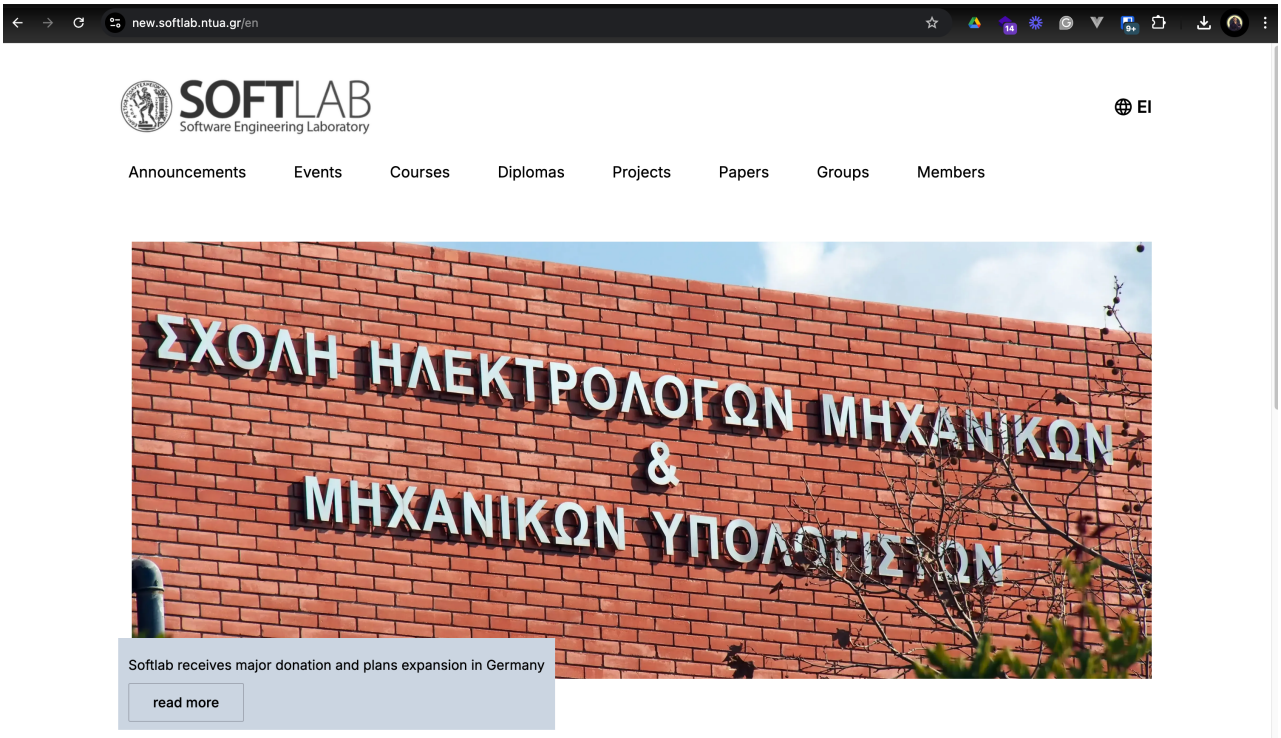

Figure 5.10: Home Page - Hero Section

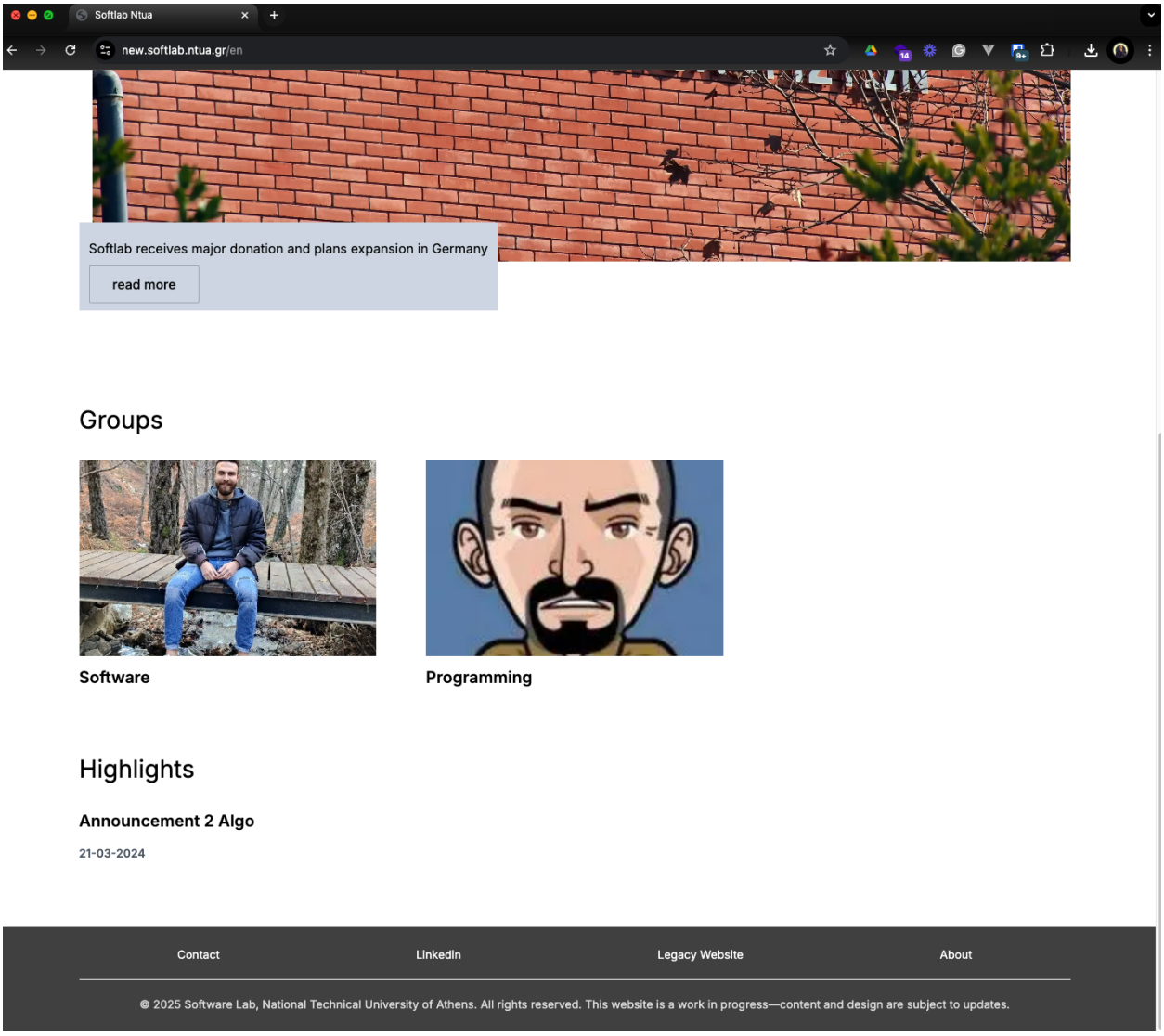

Figure 5.11: Rest of the Home Page

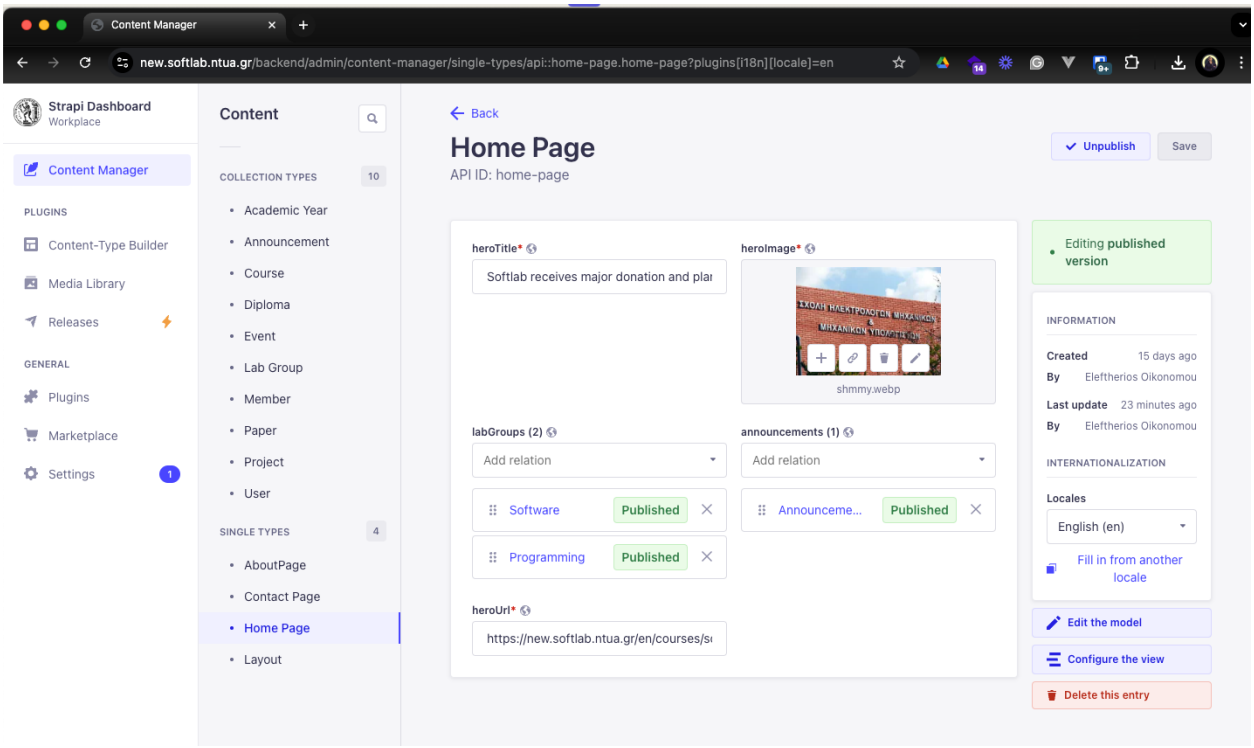

Figure 5.12: CMS Configuration for Home Page

5.2.2 Lab Group Filtering

The platform includes an advanced filtering system for lab groups, allowing users to efficiently search and explore content based on lab groups. This feature enhances discoverability and makes the user navigation easier.

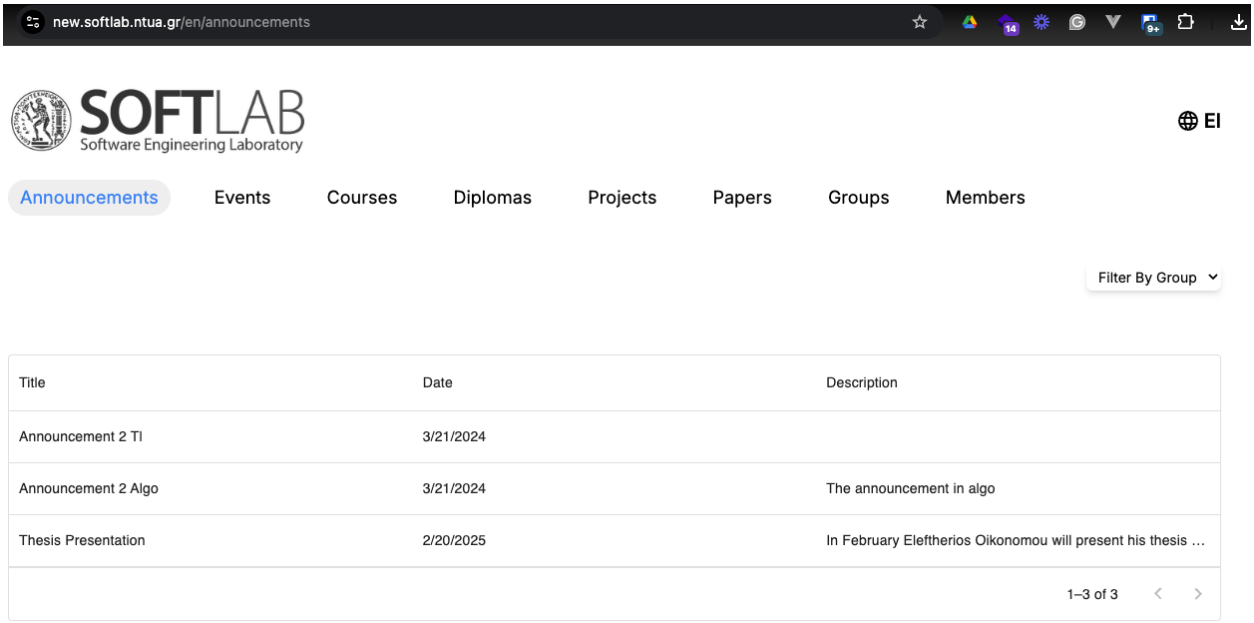

Figure 5.13: Announcements without filtering

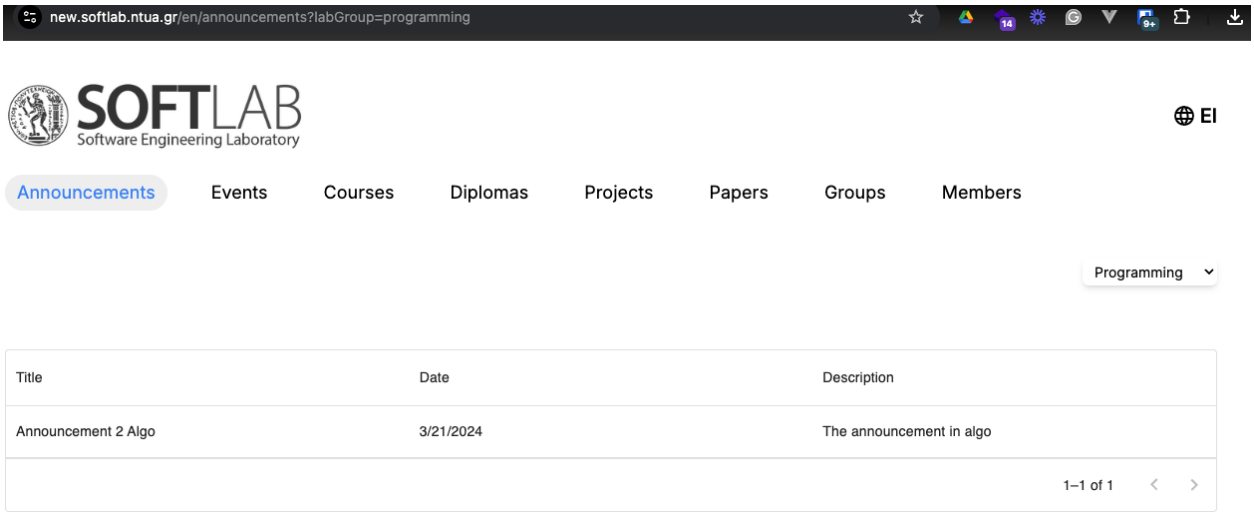

Figure 5.14: Announcements with filtering

5.1.3 Multilingual Support

The website supports multilingual functionality, ensuring content accessibility in both Greek and English. This implementation follows internationalization (i18n) best practices, allowing users to switch languages effortlessly while maintaining a consistent experience.

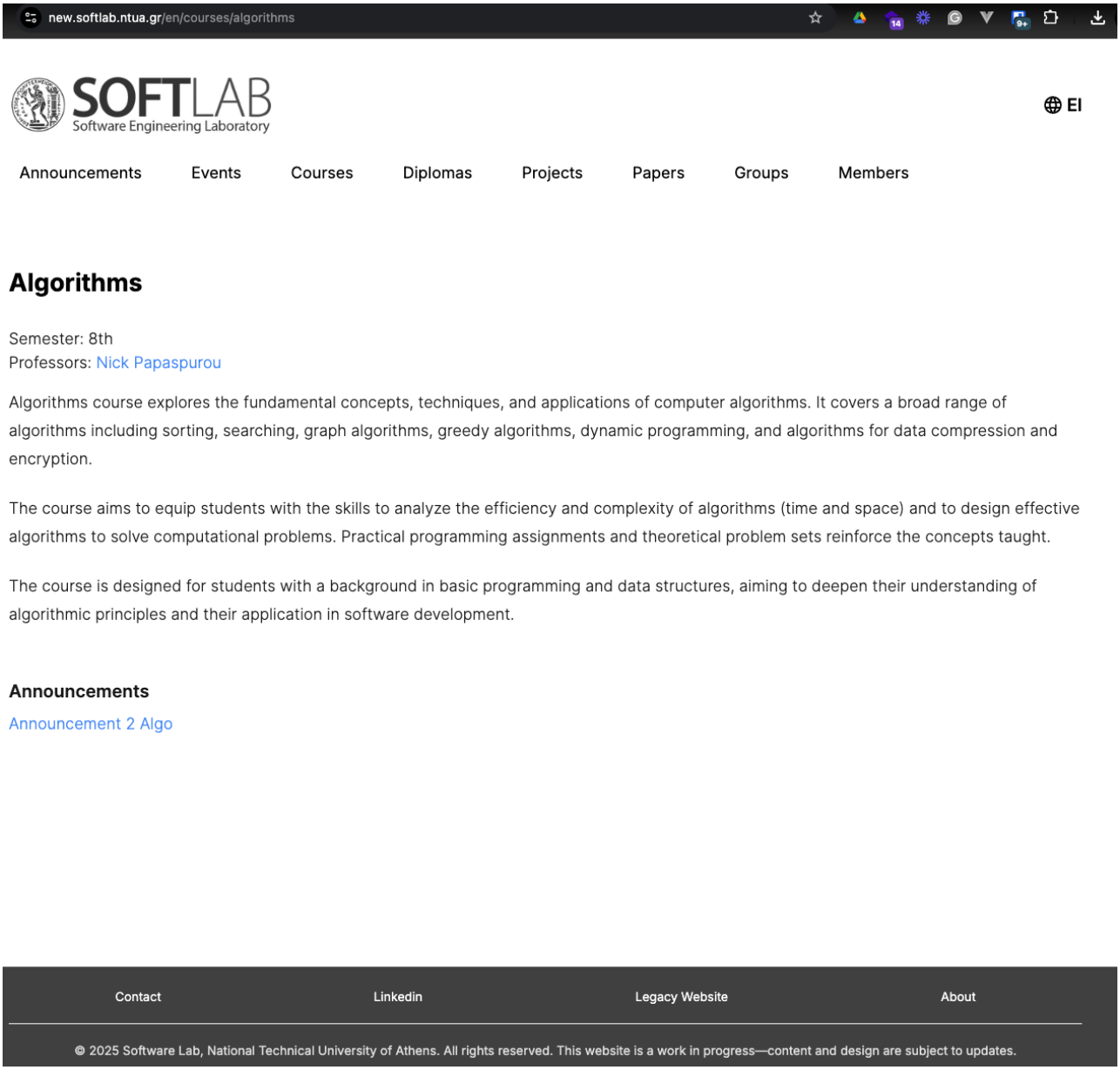

Figure 5.15: Course Page in English

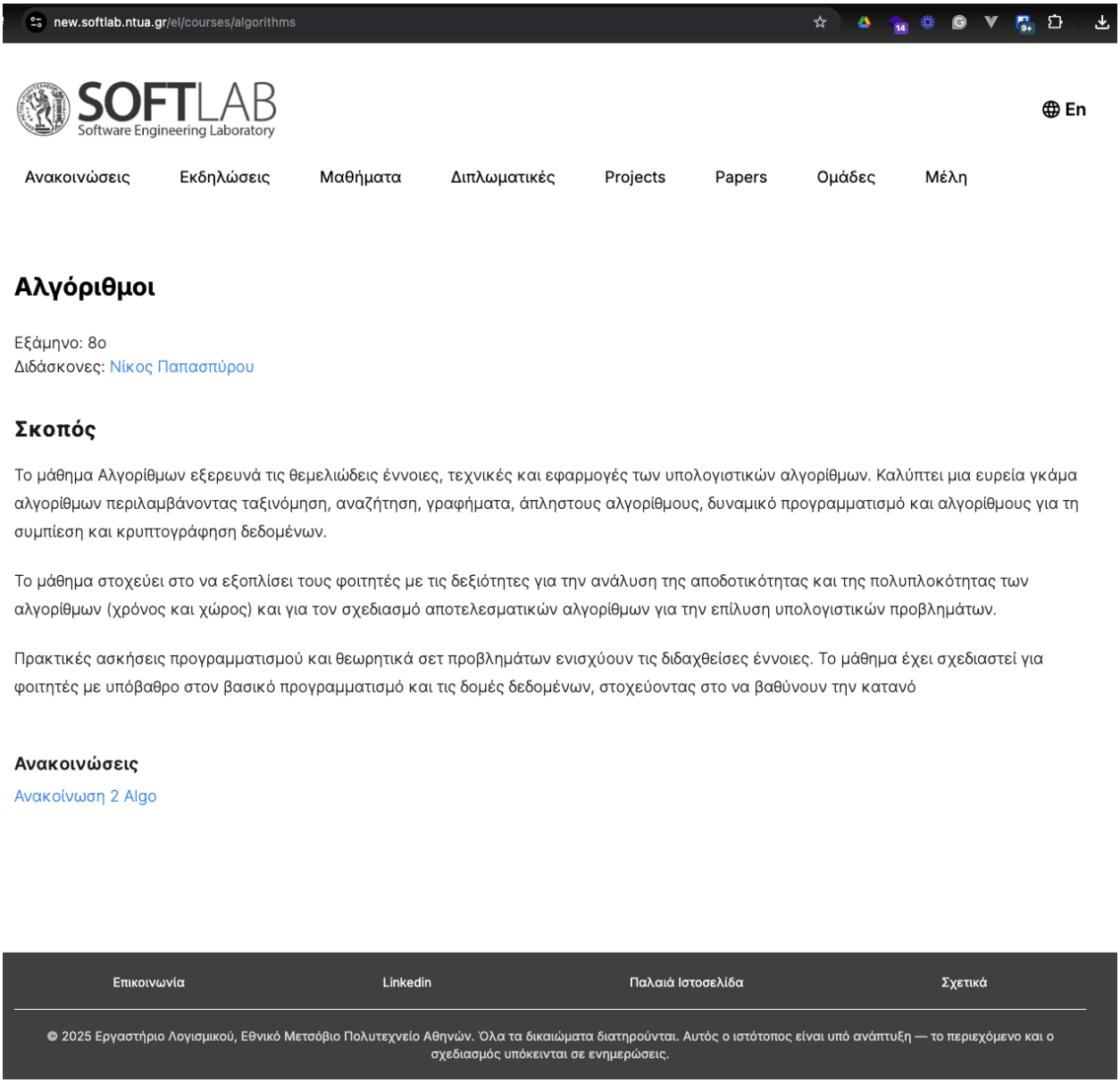

Figure 5.16: Course Page in Greek

5.3 Linkedin Integration

This section will demonstrate the LinkedIn integration. When an administrator publishes an entry with publishOnLinkedin set to true in Strapi, a webhook triggers a Next.js API route, which processes the data and posts it to LinkedIn.

5.3.1 Publishing an Entry in CMS

Follow the instructions in section 5.2 to create and publish an entry with publishOnLinkedin set to true. When the entry is published, the webhook triggers the Next.js API route.

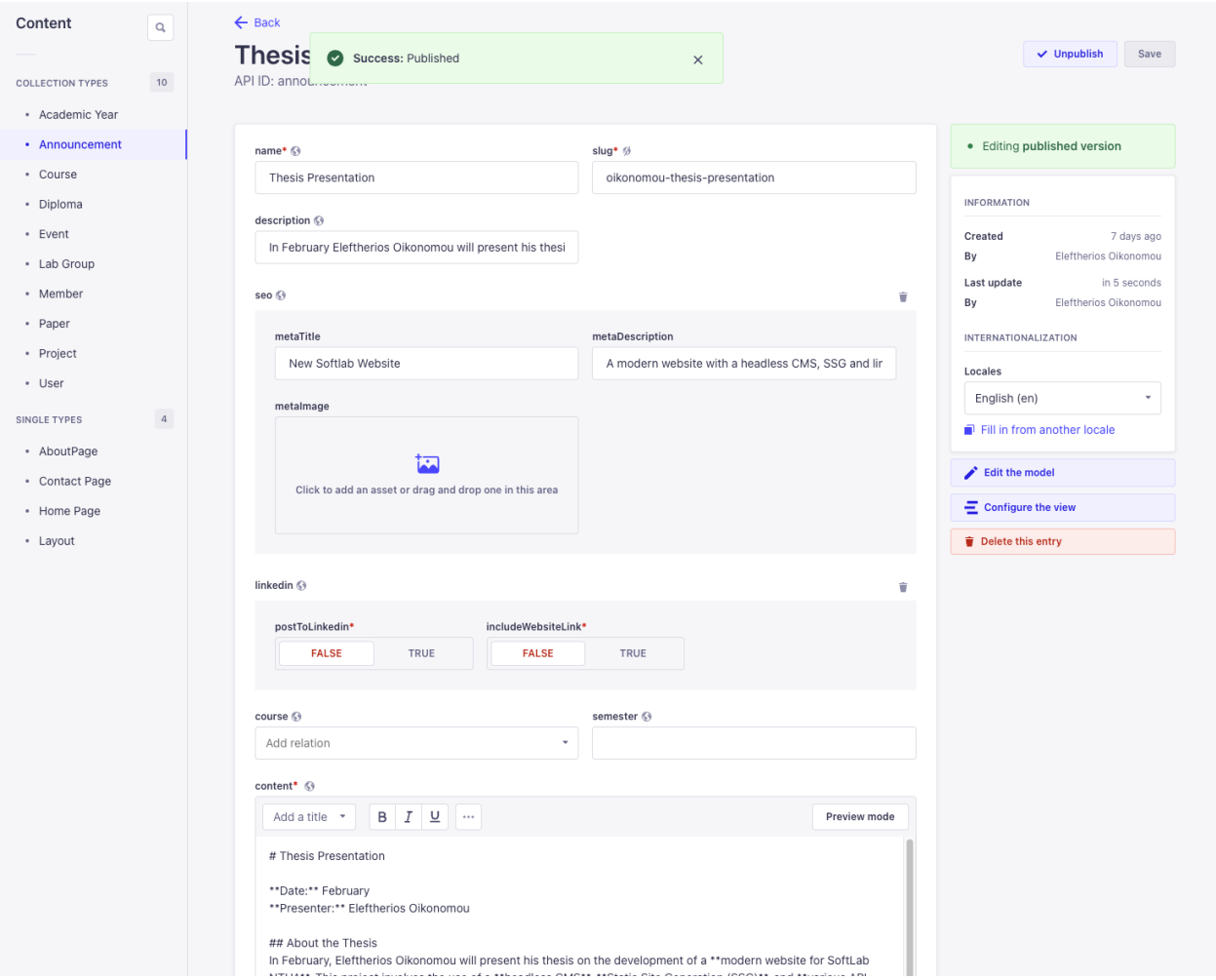

Figure 5.17: Publishing Announcement in the CMS

5.3.2 LinkedIn Post

After the Next.js API route is triggered, the entry's title and description are posted to LinkedIn. The link opens the relevant announcement page on the website.

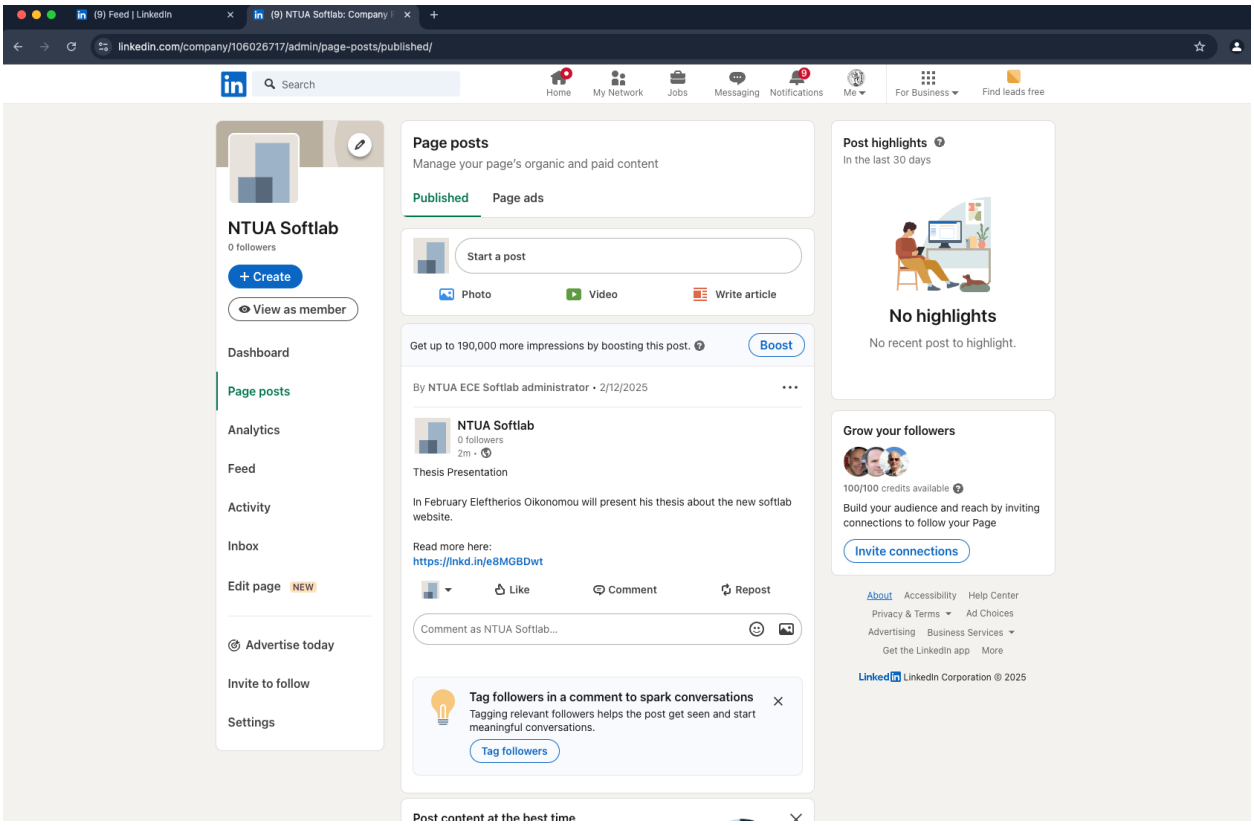

Figure 5.18: LinkedIn Post

Chapter 6 Improvements

This section outlines potential enhancements that could further improve the application's functionality, automation, and integration capabilities. These proposed features would simplify operations, enhance system resilience, and improve overall maintainability.

6.1 Automatic Builds

Implementing an automated build mechanism would allow administrators to trigger front-end updates on demand directly from the admin panel. This functionality would be beneficial when content is updated, ensuring changes are immediately reflected without requiring manual intervention. By automating the build process, the website can maintain real-time content accuracy while reducing administrative overhead.

6.2 Automated Backups

Ensuring data integrity and disaster recovery preparedness requires a fully automated backup process. Periodically saving the SQLite database to a secure location can minimize data loss risks. Scheduled backups would allow administrators to restore the system quickly in the event of unexpected failures, ensuring uninterrupted operation. This could be implemented by utilizing cron jobs.

6.3 Failure Recovery

An automated restoration mechanism would enhance system resilience by detecting website downtime and initiating a recovery process. This could involve monitoring tools that trigger a backup restoration or container restart whenever an outage is detected. Implementing such a system would avoid prolonged downtimes, ensuring high availability and reliability.

6.4 Mail Server

Integrating a mail server directly with Strapi will empower the platform to manage email communications more effectively. This integration would allow for custom emails as well as routine notifications like password resets and user engagement updates.

6.5 More Integrations

Further integrations could enhance the platform's functionality by enabling automated publishing of research papers to relevant academic repositories and journals.

Additionally, integrating NTUA's authentication system for admin panel access would improve security and user management by leveraging institutional credentials and existing authentication portals for authentication.

References

1. CMS. https://en.wikipedia.org/wiki/Content_management_system
2. Headless vs. Monolithic CMS. https://www.sanity.io/headless-cms/headless-vs-traditional-cms
3. React. http://react.dev/
4. Vue. https://vuejs.org/
5. Angular. https://angular.dev/
6. SEO. https://en.wikipedia.org/wiki/Search_engine_optimization
7. Schema org: https://schema.org/
8. SSG. https://www.cloudflare.com/en-gb/learning/performance/static-site-generator/
9. SSR. https://en.wikipedia.org/wiki/Server-side_scripting#Server-side_rendering
10. CSR. https://prismic.io/blog/client-side-rendering
11. What is an API? https://aws.amazon.com/what-is/api/
12. Cloudflare - Reverse Proxy. https://www.cloudflare.com/en-gb/learning/cdn/glossary/reverse-proxy/
13. Next.js. https://nextjs.org/
14. Multilingual Next.js. https://nextjs.org/docs/app/building-your-application/routing/internationalization
15. LinkedIn Community Managements Posts API. https://learn.microsoft.com/en-us/linkedin/marketing/community-management/shares/posts-api
16. Strapi Webhooks. https://docs.strapi.io/dev-docs/back-end-customization/webhooks
17. Nginx. https://nginx.org/en/
18. Cloudflare – DDoS attack https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/
19. Content Manager. https://docs.strapi.io/user-docs/content-manager
20. Strapi Internationalization. https://docs.strapi.io/user-docs/plugins/strapi-plugins#i18n
21. Content Type Builder. https://docs.strapi.io/user-docs/content-type-builder
22. Managing Administrator accounts. https://docs.strapi.io/user-docs/users-roles-permissions/managing-administrators