ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ

ΚΑΤΕΥΘΥΝΣΗ 1η: ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΙΣΤΟΥ

Ανάπτυξη πληροφοριακού συστήματος διαχείρισης υποτροφιών

Διπλωματική Εργασία

Δημήτριος Φραγγιαδάκης

Αθήνα, 2022

HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS
POSTGRADUATE PROGRAMME INFORMATICS AND TELEMATICS
COURSE 1: WEB TECHNOLOGIES AND APPLICATIONS

Scholarship management information system

Master Thesis

Dimitrios Frangiadakis

Athens, 2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗ

ΚΑΤΕΥΘΥΝΣΗ 1η: ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ ΙΣΤΟΥ

Τριμελής Εξεταστική Επιτροπή

Χρήστος Μιχαλακέλης (Επιβλέπων)
Αναπληρωτής καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ανάργυρος Τσαδήμας
ΕΤΕΠ, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Γεώργιος Κουσιουρής
Επίκουρος καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ο Δημήτριος Φραγγιαδάκης

δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

3) Όπου υφίστανται δικαιώματα άλλων δημιουργών έχουν διασφαλιστεί όλες οι αναγκαίες άδειες χρήσης ενώ το αντίστοιχο υλικό είναι ευδιάκριτο στην υποβληθείσα εργασία.

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον κ. Χρήστο Μιχαλακέλη για όλη του τη βοήθεια και υποστήριξη κατά την ακαδημαϊκή μου πορεία.

Επίσης θα ήθελα να ευχαριστήσω θερμά και τον κ. Ανάργυρο Τσαδήμα, ο οποίος καθοδήγησε τεχνικά τη διπλωματική αυτή εργασία σε κρίσιμα σημεία.

Τις θερμές μου ευχαριστίες έχει και ο Ηλίας Βλάχος, για την άριστη συνεργασία και υποστήριξη κατά τη διάρκεια της διπλωματικής εργασίας.

Θα ήθελα επίσης να ευχαριστήσω και το τελευταίο μέλος της τριμελούς εξεταστικής επιτροπής, τον κ. Γεώργιο Κουσιουρή, ο οποίος θα αναλάβει εξίσου την αξιολόγηση της πτυχιακής μου εργασίας.

Τέλος, ευχαριστώ όλα τα μέλη του διδακτικού και ερευνητικού προσωπικού του Χαροκοπείου Πανεπιστημίου για τις γνώσεις που μου μετέδωσαν, καθώς και την υποστήριξη που παρείχαν στις σπουδές μου.

Πίνακας περιεχομένων

Περίληψη στα Ελληνικά

Η εργασία αυτή πραγματεύεται την ανάπτυξη μία διαδικτυακής εφαρμογής για τη διαχείριση υποτροφιών από ιδρύματα ανά την επικράτεια, η οποία αναπτύχθηκε υπό την αιγίδα του Study in Greece.

Αυτή τη στιγμή τα ιδρύματα χρησιμοποιούν διάφορες πλατφόρμες για να δημοσιοποιήσουν τις ανακοινώσεις τους για τις παρεχόμενες υποτροφίες τους. Αυτό δεν αποτελεί τη βέλτιστη λύση, γιατί δυσχεραίνει το έργο των υποψηφίων στην αναζήτηση της κατάλληλης υποτροφίας. Έτσι αναπτύχθηκε η πλατφόρμα που αναλύεται σε αυτή τη διπλωματική εργασία.

Οι βασικοί χρήστες αυτής της πλατφόρμας είναι ο γενικός διαχειριστής, τα ιδρύματα και οι υποψήφιοι. Σε αυτή την πλατφόρμα ένας χρήστης αποκτάει έναν ιδρυματικό λογαριασμό - σε συνεννόηση με το γενικό διαχειριστή της πλατφόρμας - και έπειτα του δίνεται η δυνατότητα να διαχειρίζεται εξ' ολοκλήρου τις υποτροφίες του ιδρύματος, δηλαδή να δημιουργεί, να προσαρμόζει, να διαγράφει ή να απενεργοποιεί υποτροφίες. Τέλος, οι δυνητικοί υποψήφιοι μπορούν να αναζητούν την υποτροφία που τους ενδιαφέρει, με βάση διάφορα κριτήρια που εκείνοι επιλέγουν.

Για να γίνει αυτό το έργο πραγματικότητα, επιλέχθηκε να χρησιμοποιηθεί το Django, το οποίο αποτελεί ένα framework για την ανάπτυξη διαδικτυακών εφαρμογών, βασισμένο στη γλώσσα προγραμματισμού Python. Ο κύριος όγκος των δεδομένων μας αποθηκεύεται σε μία σχεσιακή βάση δεδομένων PostgreSQL. Επίσης επιλέχθηκε να χρησιμοποιηθεί το Keycloak, ένα ασφαλές σύστημα κεντρικής διαχείρισης χρηστών πολλαπλών εφαρμογών**.**

Τέλος, η εφαρμογή παραμετροποιήθηκε για να λειτουργεί υπό τη μορφή container με τη χρήση του δημοφιλούς λογισμικού Docker και παράλληλα υλοποιήθηκε ένα αρχείο configuration για τη διευκόλυνση της ανάπτυξης με τη χρήση του docker-compose, ενός λογισμικού για τη διαχείριση εφαρμογών σε containers.

Λέξεις κλειδιά: Django, Python, PostgreSQL, εφαρμογή, διαδίκτυο

Abstract ή Περίληψη στα Αγγλικά

This thesis deals with the development of a web application for the management of scholarships from institutions throughout the country, which was developed with the support of Study in Greece.

Currently, institutions use various platforms to publicize their announcements about the scholarships they offer. This is not an optimal solution, as it makes it difficult for applicants to find the right scholarship for them. Thus, the platform described in this thesis was developed.

The main user entities of this platform are the general administrator, the institutions and the applicants. In this platform, a user acquires an institutional account – approved by the general administrator of the platform - and is then given the possibility to fully manage the scholarships of the institution, i.e., to create, edit, delete or deactivate scholarships. Finally, potential applicants can search for the scholarship they are interested in, based on various criteria of their choice.

To develop this project, Django was chosen, which is a framework for developing web applications based on the Python programming language. The bulk of our data is stored in a relational PostgreSQL database. Keycloak, a secure centralized user management system for multiple applications, was also chosen to be utilized.

Finally, the application was configured to also run in a container format, using the popular Docker software and a configuration file was implemented to facilitate development using docker-compose, a software for managing/orchestrating applications in containers.

Keywords: Django, Python, PostgreSQL, application, web

Κατάλογος εικόνων

Συντομογραφίες/Ακρωνύμια

ΠΣ	Πληροφοριακό Σύστημα
SiG	Study in Greece
ΙΚΥ	Ίδρυμα Κρατικών Υποτροφιών
IDE	Integrated Development Environment
MVT	Model-View-Template
MVC	Model-View-Controller
CRUD	Create – Read – Update – Delete

Εισαγωγή

Σε αυτή την εργασία θα γίνει μία ανάλυση του πληροφοριακού συστήματος διαχείρισης υποτροφιών, το οποίο θα μπορούν να χρησιμοποιήσουν διάφορα ιδρύματα για να δημιουργούν και να επεξεργάζονται τις υποτροφίες τους.

Study in Greece

Το ΠΣ αναπτύσσεται υπό την αιγίδα του Study in Greece, τον επίσημο φορέα διεθνοποίησης κι εξωστρέφειας της ελληνικής ανώτατης εκπαίδευσης. {1}

Βασικά πεδία της αποστολής του SiG είναι:

- Η παροχή πληροφοριών σχετικών με τις σπουδές και τη διαμονή στην Ελλάδα.
- Η οργάνωση, υποστήριξη και προώθηση εκπαιδευτικών και πολιτισμικών δραστηριοτήτων.
- Ο εντοπισμός και η προώθηση εκπαιδευτικών προγραμμάτων σε διάφορους επιστημονικούς τομείς, τα οποία απευθύνονται σε διεθνείς φοιτητές.

Το πρόβλημα

Πριν την ανάπτυξη του ΠΣ, τα διάφορα ιδρύματα επιλέγουν διαφορετικούς τρόπους να γνωστοποιούν τις υποτροφίες τους, κυρίως μέσω των δικών τους ιστοσελίδων, όπως το ίδρυμα Fullbright Greece {2} ή το ΙΚΥ {3}.

Αυτό καθιστά ιδιαίτερα δύσκολο τον εντοπισμό της κατάλληλης υποτροφίας από τους υποψήφιους, καθώς πρέπει πρώτα να εντοπίσουν τα κατάλληλα ιδρύματα που προσφέρουν υποτροφίες και έπειτα να προσπελάσουν τις υποτροφίες του καθενός χωρίς να μπορούν κάπως να τις φιλτράρουν με βάση τα ενδιαφέροντα και τις προτιμήσεις τους.

1. Το Πληροφοριακό Σύστημα

Για να βελτιωθεί η εμπειρία αναζήτησης υποτροφιών από ιδρύματα στην Ελλάδα, αποφασίστηκε να αναπτυχθεί το παρόν ΠΣ, το οποίο θα εξορθολογούσε τη διαδικασία ανάρτησης υποτροφιών από τα εκάστοτε ιδρύματα, καθώς και θα διευκόλυνε τους υποψήφιους στην αναζήτηση των υποτροφιών που τους ενδιαφέρουν.

Το ΠΣ μπορεί εύκολα να διακριθεί στα εξής υποσυστήματα:

- Υποσύστημα γενικής διαχείρισης
- Υποσύστημα ιδρυμάτων
    - Δημιουργία υποτροφίας
    - Διαχείριση υποτροφιών
- Σελίδες χρηστών
    - Αρχική
    - Αναζήτηση

Τα παραπάνω υποσυστήματα θα αναλυθούν περεταίρω στη συνέχεια της εργασίας.

2. Περιβάλλον εφαρμογής

Πριν ακόμα ξεκινήσει η ανάπτυξη, ορίστηκαν οι βασικές τεχνολογίες που θα χρησιμοποιούταν. Γνωρίζοντας ότι ο στόχος είναι η εφαρμογή να είναι orchestrated μέσω Kubernetes στο παραγωγικό περιβάλλον, χρησιμοποιήθηκε Docker για το containerization της εφαρμογής, καθώς και docker-compose για να είναι η διαδικασία ανάπτυξης όσο πιο κοντά στο παραγωγικό περιβάλλον, παραμένοντας φιλικό προς τον προγραμματιστή.

2.1 Docker

Το λογισμικό Docker {4} αποτελεί μία από τις δημοφιλέστερες λύσεις στο containerization εφαρμογών.

Το πρόβλημα που λύνει είναι ότι πολλές εφαρμογές συμπεριφέρονται ελαφρώς διαφορετικά ανάλογα με το περιβάλλον μέσα στο οποίο τρέχουν. Το docker ενθυλακώνει την εφαρμογή σε μία λογική μονάδα λογισμικού που αποκαλεί container, με αποτέλεσμα να είναι environment agnostic και να τρέχει και τον ίδιο προβλεπόμενο τρόπο σε όλα τα περιβάλλοντα που υποστηρίζουν το docker engine.

Επίσης διευκολύνει πολύ και την εμπειρία του προγραμματιστή, καθώς ένας προγραμματιστής δε χρειάζεται πλέον να ξέρει ακριβώς τι πρέπει να κάνει σε κάθε πιθανό διαφορετικό περιβάλλον για να το κάνει να λειτουργήσει με τον προβλεπόμενο τρόπο, το μόνο που χρειάζεται να κάνει είναι να χρησιμοποιήσει μία containerized έκδοση της εφαρμογής του, σχεδιασμένη για development (η οποία κάνει rebuild project σε κάθε αλλαγή).

```
FROM python:3.8.12-buster

# set work directory
WORKDIR /app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN pip install --upgrade pip
COPY ./requirements.txt .
RUN pip install -r requirements.txt

# copy project
COPY ./scholarmarket .

CMD ["python", "manage.py", "runserver",  "0.0.0.0:8000"]
```

Εικόνα 1: Dockerfile

2.2 Docker-compose

Το docker-compose είναι ένα ελαφρύ εργαλείο ενορχήστρωσης containerized εφαρμογών. Όταν οι εφαρμογές μας εκτελούνται σε containers είναι πολύ σημαντικό να μπορούμε να συντονίζουμε τα διάφορα containers μεταξύ τους, να διασφαλίζεται η επανεκκίνηση τους σε περίπτωση σφαλμάτων, αλλά ακόμα και η σειρά έναρξης τους αν κάποιο εξαρτάται από ένα άλλο – για παράδειγμα, μία διαδικτυακή εφαρμογή μπορεί να εξαρτάται από τη βάση δεδομένων.

Παρόλο που το docker-compose μπορεί να χρησιμοποιηθεί και σε παραγωγικό περιβάλλον, στην παρούσα εφαρμογή χρησιμοποιείται μόνο για το προγραμματιστικό περιβάλλον, καθώς μία από τις απαιτήσεις της εφαρμογής είναι να εκτελείται στην παραγωγή μέσω Kubernetes.

Τα containers που εκτελούνται στο περιβάλλον προγραμματισμού είναι τα εξής:

- **web**: Το container με των κώδικα της εφαρμογής μας
- **postresql**: Το container με τη βάση δεδομένων μας
- **keycloak**: Το container με το σύστημα αυθεντικοποίησης και διαχείρισης χρηστών της εφαρμογής μας

Ο ρόλος του κάθε container θα αναλυθεί περεταίρω στη συνέχεια της εργασίας.

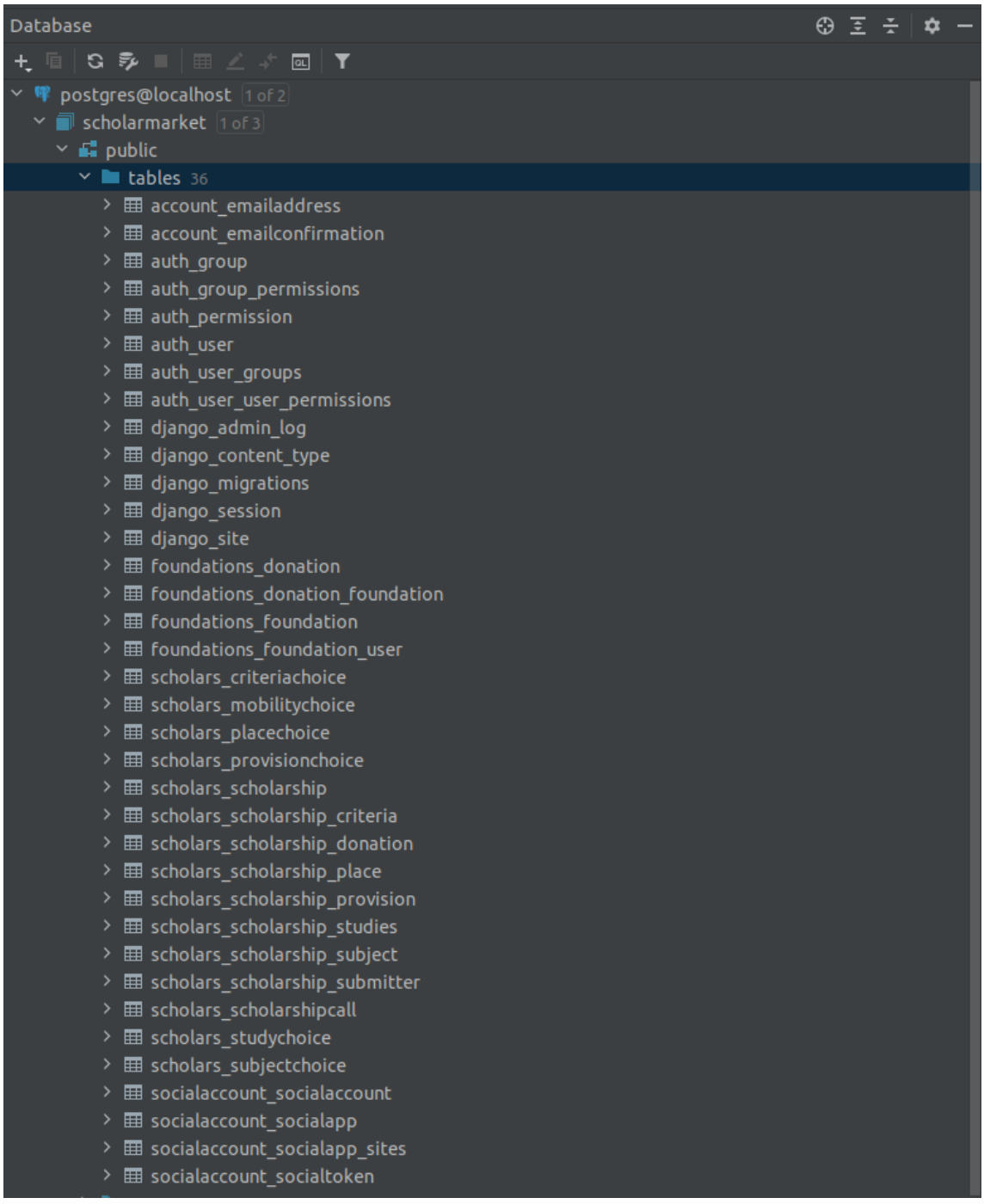
```yaml
version: '3.3'

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./scholarmarket/:/usr/src/app/
    ports:
      - '8000:8000'
    env_file:
      - ./scholarmarket/scholarmarket/.env.dev
    depends_on:
      - postgresql
  postgresql:
    image: docker.io/bitnami/postgresql:11
    ports:
      - '5432:5432'
    volumes:
      - 'postgresql_data:/bitnami/postgresql'
    environment:
      - 'POSTGRESQL_USERNAME=user'
      - 'POSTGRESQL_PASSWORD=test'
      - 'POSTGRESQL_DATABASE=scholarmarket'
  keycloak:
    ports:
      - '8080:8080'
    environment:
      - KEYCLOAK_USER=admin
      - KEYCLOAK_PASSWORD=admin
    image: 'quay.io/keycloak/keycloak:15.0.2'

volumes:
  postgresql_data:
    driver: local
```

Εικόνα 2: docker-compose.yml

2.3 Βάση δεδομένων PostgreSQL

Για την εφαρμογή αυτή επιλέχθηκε η δημοφιλής σχεσιακή βάση δεδομένων PostgreSQL. Στο περιβάλλον μας δημιουργείται ένας χρήστης και μία βάση δεδομένων, πάνω στην οποία ο χρήστης έχει όλα τα απαραίτητα δικαιώματα (permissions).

Η διαχείριση της βάσης δεδομένων γινόταν μέσω του υπο-προγράμματος για διαχείριση βάσεων δεδομένων του PyCharm, του πιο ολοκληρωμένου Python IDE.



Εικόνα 3: PyCharm Database Explorer

2.4 Python, pip & dependency management

Η γλώσσα η οποία έχει επιλεχθεί για την ανάπτυξη της εφαρμογής είναι η δημοφιλής Python και πιο συγκεκριμένα η έκδοση 3.8.

Επίσης χρησιμοποιείται το pip, που είναι ο πιο δημοφιλής package και dependency manager στο οικοσύστημα της python. Για να μπορούμε να έχουμε ένα προβλέψιμο περιβάλλον, έξω από τα πλαίσια των docker και docker-compose, διατηρούμε ένα αρχείο που ονομάζεται requirements.txt και περιέχει όλα τα τρίτα πακέτα που χρησιμοποιεί η εφαρμογή μας.

```
asgiref==3.3.4
certifi==2021.10.8
cffi==1.15.0
charset-normalizer==2.0.7
crispy-bootstrap5==0.4
cryptography==35.0.0
defusedxml==0.7.1
Django==3.2.4
django-allauth==0.45.0
django-autocomplete-light==3.8.2
django-crispy-forms==1.12.0
django-environ==0.4.5
django-jazzmin==2.4.7
django-modeltranslation==0.17.3
idna==3.3
oauthlib==3.1.1
psycopg2-binary==2.8.6
pycparser==2.21
PyJWT==2.3.0
python-dotenv==0.18.0
python3-openid==3.2.0
pytz==2021.1
requests==2.26.0
requests-oauthlib==1.3.0
six==1.16.0
sqlparse==0.4.1
urllib3==1.26.7
```

Εικόνα 4: requirements.txt

2.4.1 django-allauth

Για το keycloak integration, επιλέχθηκε το πακέτο django-allauth, το οποίο μας επιτρέπει να κάνουμε interface με πάρα πολλούς παρόχους authentication.

Αυτή η επιλογή κάνει την εφαρμογή μας επίσης future-proof, επιτρέποντας αρκετά εύκολα μία δυνητική επέκταση σε κάποιον άλλο τρόπο αυθεντικοποίησης.

2.4.2 django-modeltranslation

Μία ακόμα από τις απαιτήσεις ήταν το περιεχόμενο της εφαρμογής να ήταν δίγλωσσο (ελληνικά και αγγλικά).

Η μετάφραση του στατικού περιεχομένου υποστηρίζεται ήδη από το Django. Η πρόκληση που αντιμετωπίστηκε ήταν να γίνεται η μετάφραση στο επίπεδο των μοντέλων των υποτροφιών και επίσης με το βέλτιστο τρόπο κατά την προβολή τους.

Για να γίνει αυτό με το καλύτερο τρόπο σύμφωνα με τα patterns του Django, να διατηρηθεί καθαρός ο κώδικας και για να διευκολυνθούμε αργότερα στη δημιουργία και προβολή των υποτροφιών, επιλέχθηκε το πακέτο django-modeltranslation.

Το πακέτο αυτό φτιάχνει τα μοντέλα μας, αφού του ορίσουμε ποια πεδία θα είναι δίγλωσσα σε κάθε μοντέλο της εφαρμογής μας μέσω configuration files. Πλέον τα μοντέλα μας έχουν 3 πεδία για κάθε δίγλωσσο πεδίο όπως το παρακάτω παράδειγμα:

- title
- title_en
- title_el

Στο παράδειγμα μας, το πεδίο title παραμένει κενό ή το περιεχόμενο του μας είναι αδιάφορο. Τα πεδία title_en και title_el περιέχουν τον τίτλο του μοντέλου μας αντίστοιχα και κατά την προβολή του μοντέλου, προσπελάζοντας το πεδίο title, παίρνουμε τον τίτλο στην επιλεγμένη

γλώσσα, δηλαδή το πεδίο title λειτουργεί σαν ένα δυναμικό symlink, ανάλογα με την επιλεγμένη γλώσσα από το χρήστη της εφαρμογής.

2.4.3 python-dotenv

Ένα πολύ χρήσιμο πακέτο είναι το python-dotenv. Αυτό επιτρέπει στην εφαρμογή μας να χρησιμοποιεί Dotenv (.env) αρχεία για να παίρνει μεταβλητές περιβάλλοντος μέσω ενός configuration αρχείου.

Οι μεταβλητές περιβάλλοντος είναι πολύ σημαντικές για μία εφαρμογή, γιατί ο προγραμματιστής μπορεί να περάσει δυναμικά τιμές στην εφαρμογή του, ειδικότερα σε περιβάλλοντα cloud και orchestration, στην περίπτωση μας είτε με docker-compose στο development ή με Kubernetes στο production.

Οπότε ανάλογα το που εκτελείται η εφαρμογή μας, μπορούμε με έναν πολύ εύκολο τρόπο να περάσουμε διαφορετικές παραμέτρους στην εφαρμογή μας.

Για παράδειγμα, μπορεί να θέλουμε να χρησιμοποιήσουμε ένα διαφορετικό database instance στο development και στο production. Για να το πετύχουμε αυτό θα θέσουμε τις παρακάτω μεταβλητές σε ένα .env αρχείο:

```
DB_ENGINE=django.db.backends.postgresql
DB_HOST=postgresql
DB_NAME=scholarmarket
DB_USER=user
DB_PASSWORD=test
```

Εικόνα 5: .env DB values

Και για να αλλάξουμε μεταξύ instances αρκεί να αλλάξουμε τα πεδία DB_HOST, DB_NAME, DB_USER και DB_PASSWORD στα αντίστοιχα dev και prod.

```
SECRET_KEY=django-insecure-****

DB_ENGINE=django.db.backends.postgresql
DB_HOST=postgresql
DB_NAME=scholarmarket
DB_USER=user
DB_PASSWORD=test
DB_PORT=5432
```

Εικόνα 6: .env.dev

3. Αρχιτεκτονική της εφαρμογής

Η εφαρμογή είναι κυρίως κατασκευασμένη με τα παρακάτω δομικά στοιχεία

3.1 Django

Ο κύριος κορμός της εφαρμογής είναι το Django Framework. Το Django είναι ένα web application framework το οποίο επιτρέψει κανείς να φτιάξει πολύπλοκες, γρήγορες, ασφαλείς και scalable εφαρμογές σε πολύ μικρό χρονικό διάστημα, προσφέροντας πολλή προϋπάρχουσα λειτουργικότητα. {5}

Το Django ακολουθεί το MVT paradigm, που είναι πολύ κοντά στο οικείο MVC. Τα αρχικά του σημαίνουν Model-View-Template. Οι οντότητες μας (Scholarships, Users, Foundations κλπ) αποτελούν τα Models, η λογική εκτέλεσης του προγράμματος μας βρίσκεται στα Views (πολύ κοντά στη λογική των Controllers του MVC) και οι προβολές τους βρίσκονται στα Templates (τα αντίστοιχα Views στο MVC).

3.1.1 Django Applications

Το Django οργανώνεται σε λογικές μονάδες που αποκαλούνται Django Applications που μπορούν να επαναχρησιμοποιηθούν και σε άλλα projects. {6}

Σε αυτή την εφαρμογή έχουμε 3 applications:

- users: Περιέχει τα μοντέλα, τα templates και τη λογική για τη διαχείριση των χρηστών της εφαρμογής
- foundations: Περιέχει τα μοντέλα, τα templates και τη λογική για τη διαχείριση των ιδρυμάτων
- scholars: Περιέχει τα μοντέλα, τα templates και τη λογική για τη διαχείριση των υποτροφιών

3.2 Django templating language

Το Django templating language είναι η λύση που προσφέρει το Django για την παραγωγή HTML σελίδων με δυναμικό περιεχόμενο.

Ο DTL κώδικας γράφεται στα Templates του Django και μοιάζει αρκετά με το δημοφιλές python templating engine Jinja2 χωρίς να ταυτίζονται.

```
{% extends 'base.html' %}
{% load static %}
{% load i18n %}

{% block title %}
    Foundation - Scholarships
{% endblock %}

{% block content %}
    {% for scholarship in scholarships %}
        <div class="row">
            <div class="col col-12">
                <div class="card">
                    <div class="card-header">
                        <h3 style="display: inline">{{
scholarship.title }}</h3>
                        <span class="float-end"><a href="{% url
'scholar-user-details' scholarship.id %}" class="btn btn-outline-
sig">Άνοιγμα</a></span>
                        <span class="float-end"><a href="{% url
'scholar-user-details' scholarship.id %}" class="btn btn-outline-
sig">Επεξεργασία</a></span>
                    </div>
                    <div class="card-body">
                        <p>{{ scholarship.description }}</p>
                    </div>
                </div>
            </div>
        </div>
        <div class="spacer-sm"></div>
    {% endfor %}

{% endblock %}
```

Εικόνα 7: Django templating language

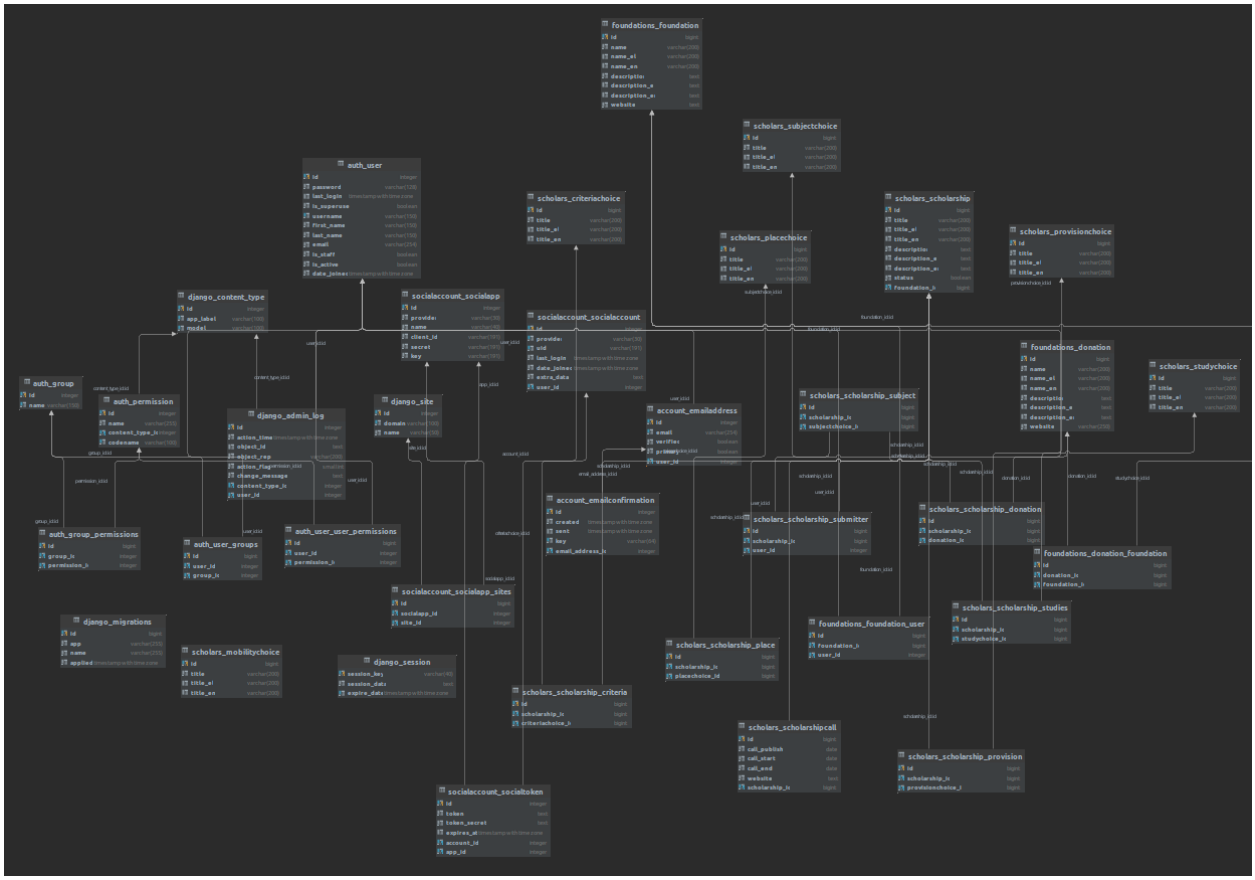
3.3 Βάση δεδομένων

Πολύ μεγάλη σημασία στην ταχύτητα ανάπτυξης, στην αποδοτικότητα, αλλά και στην επεκτασιμότητα του λογισμικού συνεισφέρει ο σωστός σχεδιασμός δομής της βάσης δεδομένων. Έπειτα από μελέτη των απαιτήσεων της εφαρμογής προέκυψε τα παρακάτω:

3.3.1 Σχεδιασμός βάσης δεδομένων

Η αρχιτεκτονική της εφαρμογή μας ακολουθεί τη λογική που αποτυπώνεται στην παρακάτω εικόνα. Λόγω του μεγέθους του σχήματος, είναι δύσκολο να δειχθούν συνολικά όλοι οι

πίνακες και οι μεταξύ τους σχέσεις, αλλά είναι δυνατό να εξαχθούν συμπέρασμα σχετικά με την πολυπλοκότητα της εφαρμογής.



Εικόνα 8: Σχήμα βάσης δεδομένων

3.3.2 Database Migrations

Το Django μας δίνει τη δυνατότητα να χρησιμοποιούμε database migrations για να κρατάμε το state της βάσης στον κώδικα μας και να μη χρειάζεται να έχουμε τη δομή των πινάκων σε αρχεία SQL.

Τα database migrations γίνονται generate από τα Models μας με την παρακάτω εντολή:

```
$ python manage.py makemigrations foundations scholars users
```

Εικόνα 9: Migration Generation

Αυτή η εντολή φτιάχνει τα migration για κάθε app που έχουμε μέσα στο Django project μας.

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [

migrations.swappable_dependency(settings.AUTH_USER_MODEL)
,
    ]

    operations = [
        migrations.CreateModel(
            name='Foundation',
            fields=[
                ('id',
models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
                ('name', models.CharField(default='',
max_length=200, verbose_name='Ονομασία Ιδρύματος')),
                ('name_el', models.CharField(default='',
max_length=200, null=True, verbose_name='Ονομασία
Ιδρύματος')),
                ('name_en', models.CharField(default='',
max_length=200, null=True, verbose_name='Ονομασία
Ιδρύματος')),
                ('description',
models.TextField(verbose_name='Περιγραφή Ιδρύματος')),
                ('description_el',
models.TextField(null=True, verbose_name='Περιγραφή
Ιδρύματος')),
                ('description_en',
models.TextField(null=True, verbose_name='Περιγραφή
Ιδρύματος')
    ]
```

Εικόνα 10: Migration example

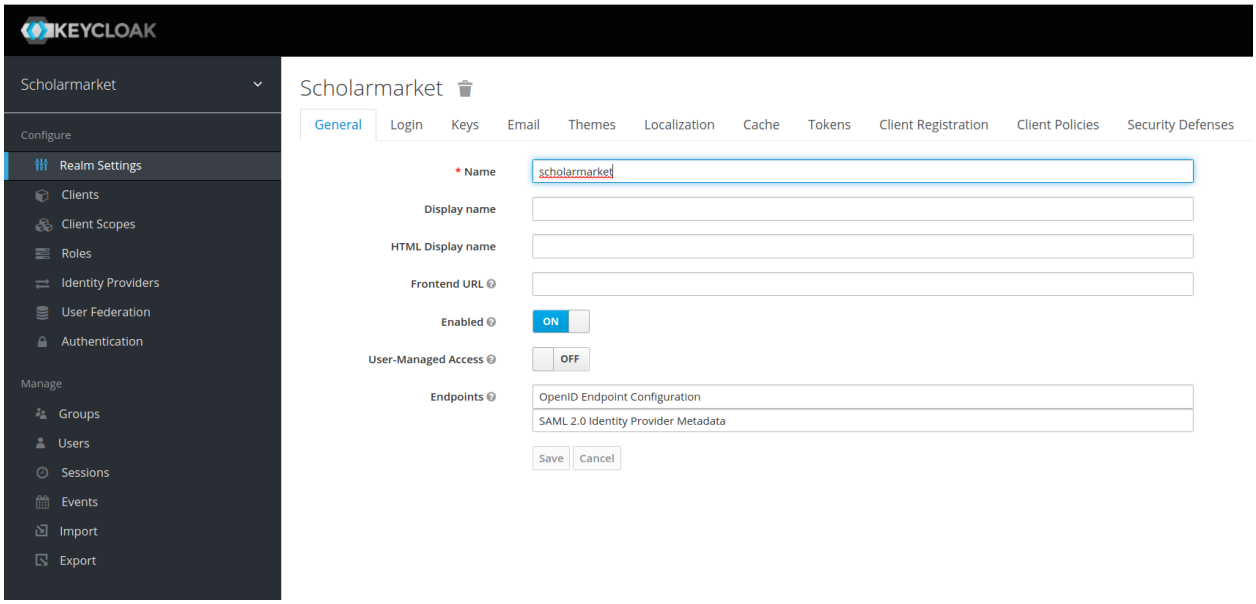
Έπειτα χρειάζεται να τρέξουμε τα migrations μας για να γεμίσει η κενή μας βάση με τη δομή των πινάκων, με την παρακάτω εντολή:

```
python manage.py migrate
```

Εικόνα 11: Εκτέλεση Migrations

3.4 Keycloak

Για τη διαχείριση των χρηστών της εφαρμογής χρησιμοποιείται το Keycloak. Το Keycloak είναι ένα λογισμικό που γίνεται deploy on premises και αναλαμβάνει το user management για δυνητικά πολλά applications.



Εικόνα 12: Keycloak admin panel

Το flow που ακολουθείται είναι η δημιουργία του χρήστη στο keycloak από το διαχειριστή. Έπειτα ο χρήστης κάνει sign-in στην εφαρμογή και ο διαχειριστής ενεργοποιεί το λογαριασμό του χρήστη από το διαχειριστικό περιβάλλον του Django αυτή τη φορά. Πλέον ο ιδρυματικός χρήστης έχεις πλήρη δικαιώματα να δημιουργήσει και να διαχειριστεί υποτροφίες.

4. Λειτουργικότητες

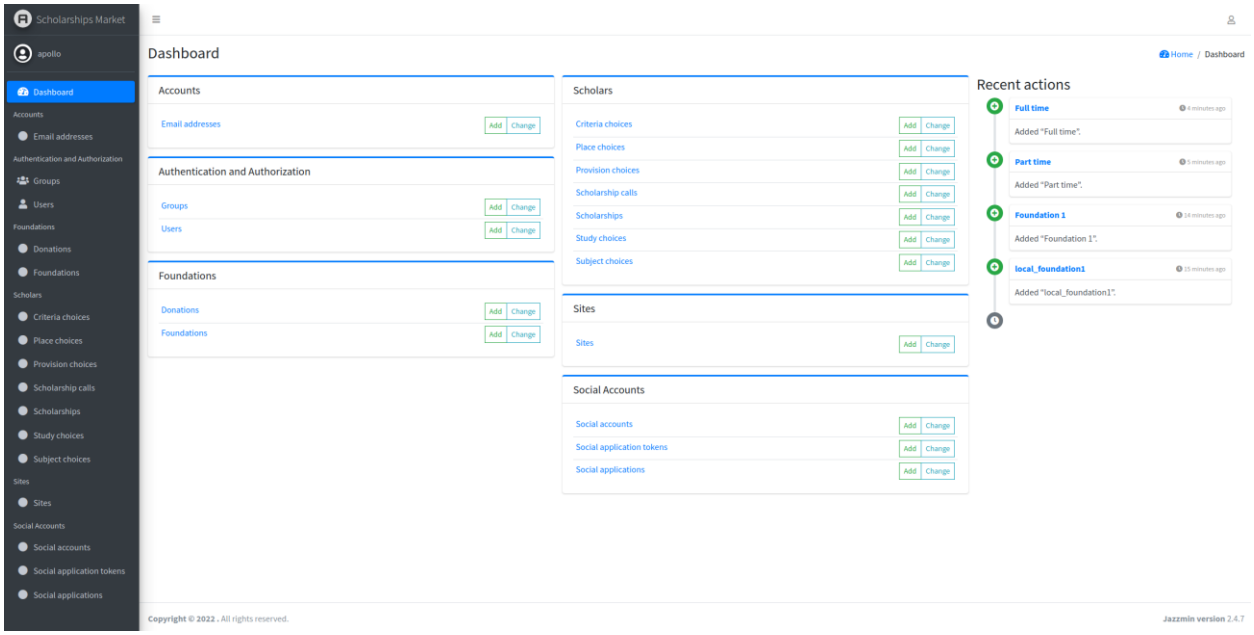
Στην παρακάτω ενότητα θα γίνει ανάλυση των δυνατοτήτων του ΠΣ που αναπτύχθηκε.

4.1 Διαχειριστής συστήματος

Ο διαχειριστής του συστήματος είναι υπεύθυνος για τη γενική διαχείριση του συστήματος, τη διαχείριση χρηστών, ιδρυμάτων, επιλογών και υποτροφιών.

4.1.1 Γενική διαχείριση

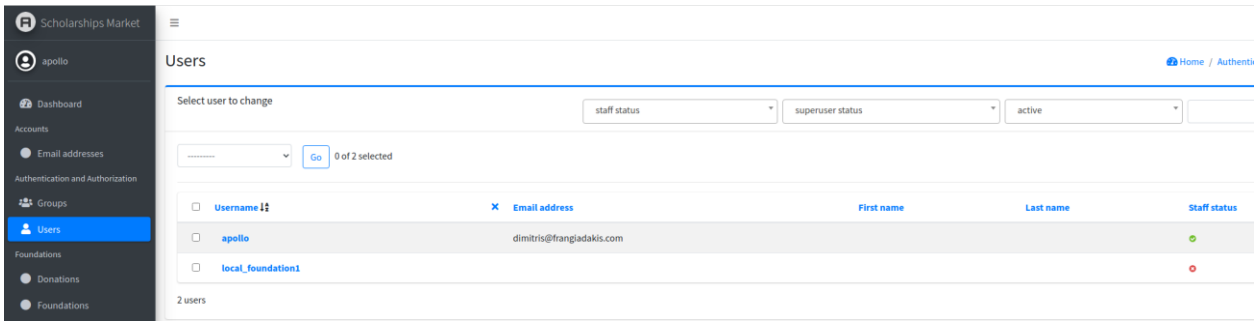
Ο γενικός διαχειριστής του συστήματος έχει τον απόλυτο έλεγχο μέσω του Django admin, μίας πλατφόρμας που παρέχεται από το Django και δίνει full CRUD access σε όλα τα μοντέλα της εφαρμογής.



Εικόνα 13: Django Admin

4.1.2 Διαχείριση χρηστών

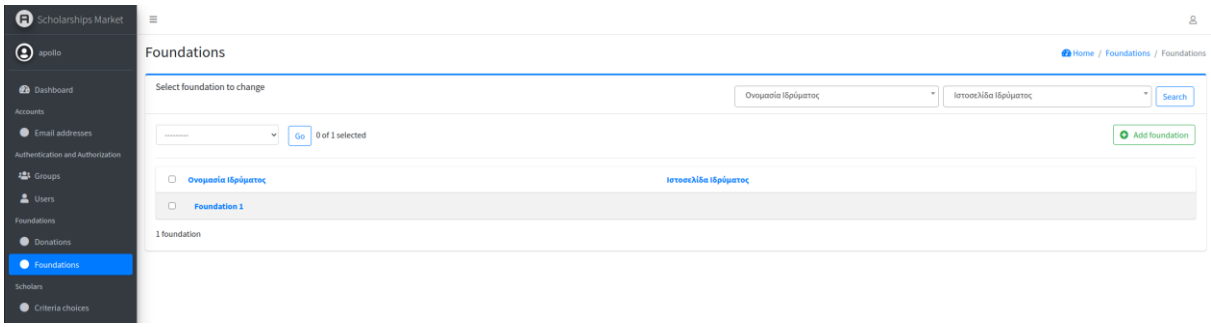
Ο γενικός διαχειριστής μπορεί επίσης να διαχειρίζεται τους χρήστες της εφαρμογής:



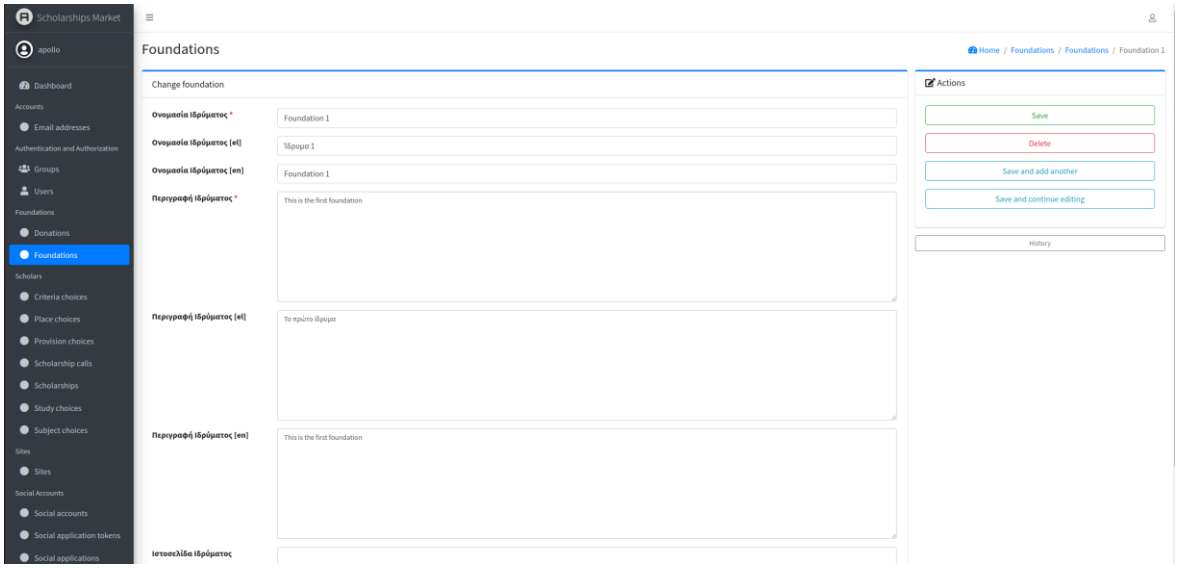
Εικόνα 14: Διαχείριση χρηστών

4.1.3 Διαχείριση ιδρυμάτων

Ο γενικός διαχειριστής μπορεί να διαχειρίζεται τα ιδρύματα της εφαρμογής:



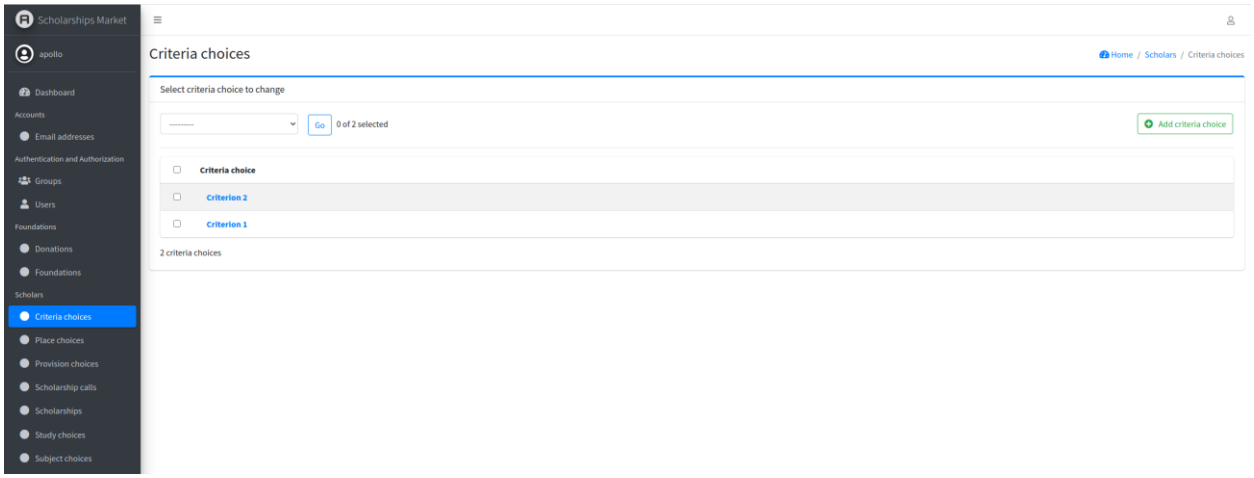
Εικόνα 15: Διαχείριση ιδρυμάτων (1)



Εικόνα 16: Διαχείριση ιδρυμάτων (2)

4.1.4 Διαχείριση παραμέτρων υποτροφιών

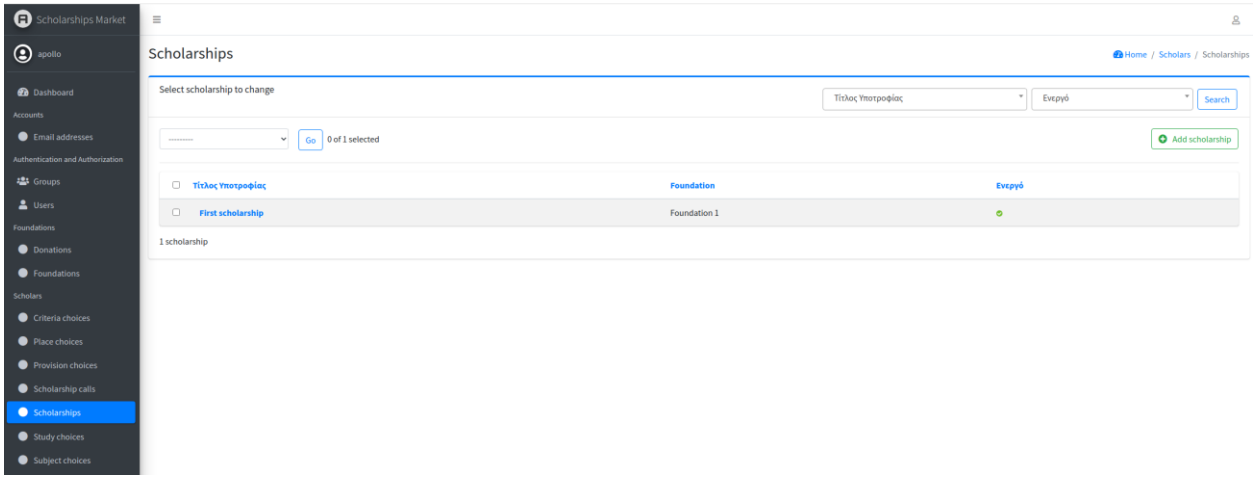
Ο διαχειριστής μπορεί να δημιουργήσει ή να επεξεργαστεί τις παραμέτρους που αφορούν υποτροφίες όπως κριτήρια επιλογής, τον τόπο της υποτροφίας, το είδος των παροχών, τους τύπους των σπουδών και τις θεματικές.



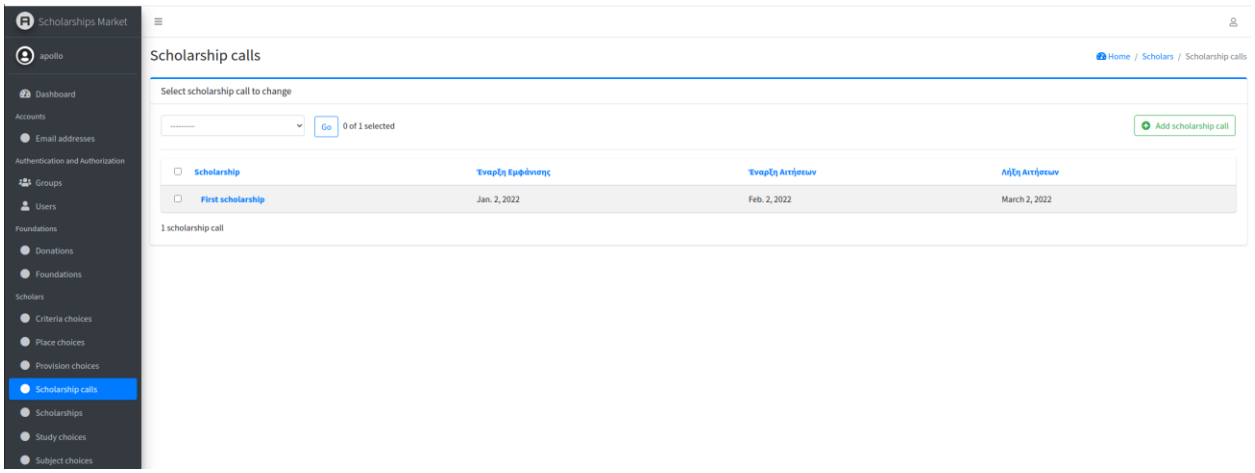
Εικόνα 17: Επεξεργασία παραμέτρων υποτροφιών

4.1.5 Διαχείριση υποτροφιών και calls

Ο διαχειριστής μπορεί να δημιουργήσει ή να επεξεργαστεί υποτροφίες για λογαριασμό ιδρυμάτων, καθώς επίσης να δημιουργήσει ή επεξεργαστεί calls για συγκεκριμένες υποτροφίες.



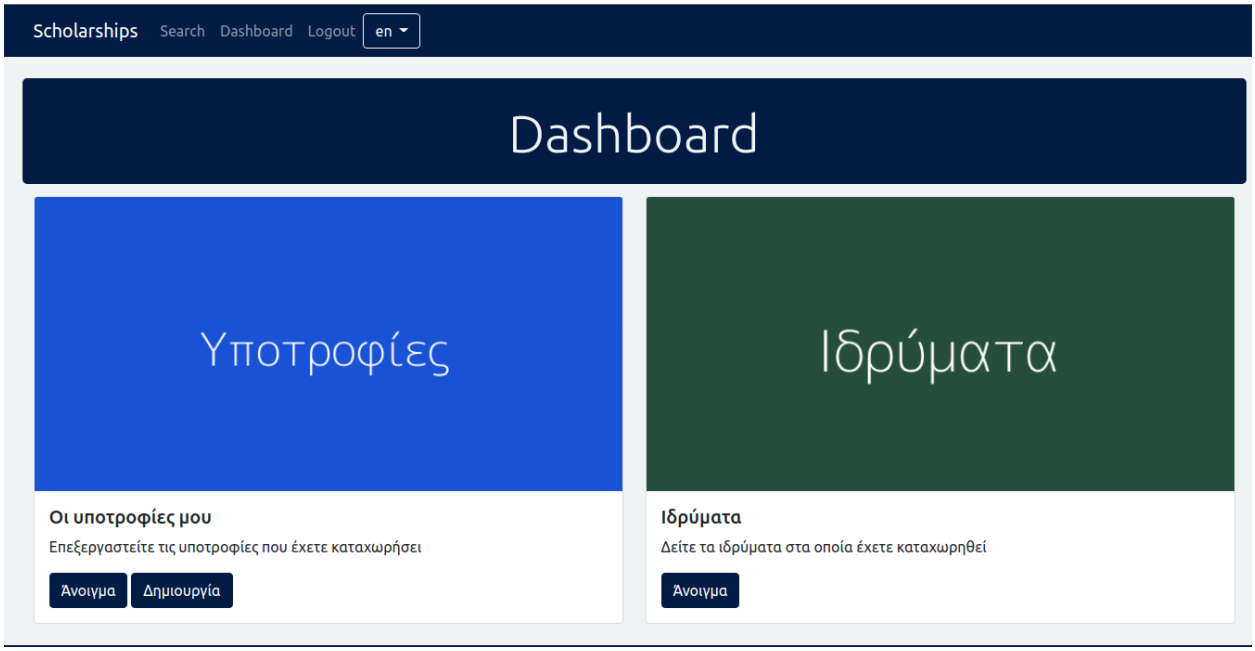
Εικόνα 18: Επεξεργασία υποτροφιών



Εικόνα 19: Επεξεργασία call υποτροφιών

4.2 Ιδρυματικός χρήστης

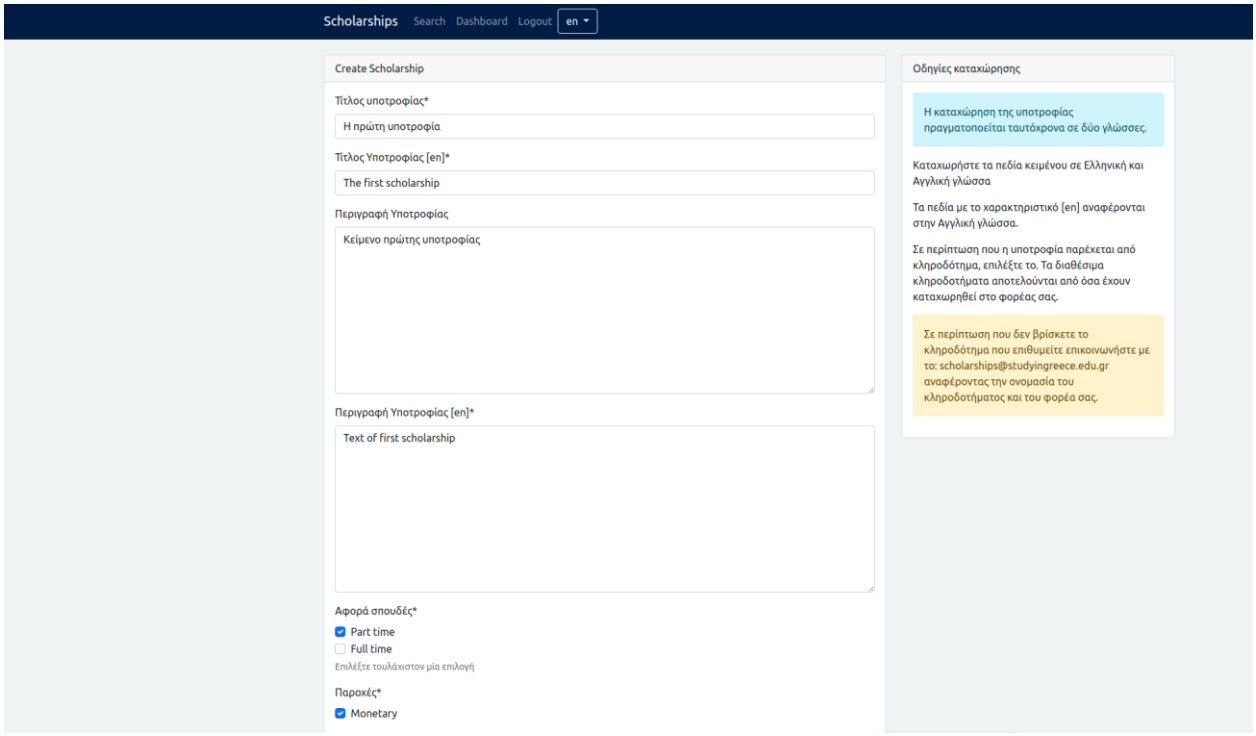
Ο ιδρυματικός χρήστης μπορεί να δημιουργήσει ή να επεξεργαστεί μία υποτροφία. Μπορεί επίσης να ενεργοποιήσει ή να απενεργοποιήσει την περίοδο αιτήσεων για μία υποτροφία.



Εικόνα 20: Dashboard ιδρυματικού χρήστη

4.2.1 Δημιουργία και επεξεργασία υποτροφίας

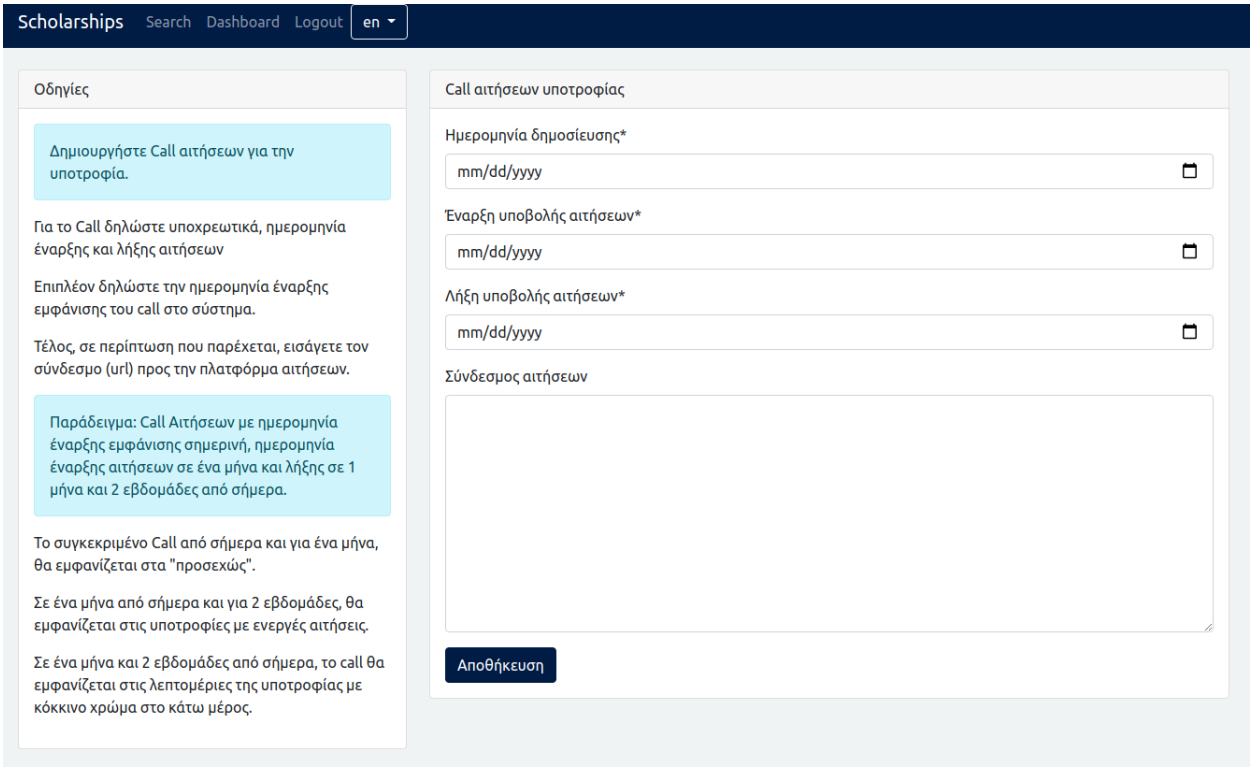
Ο ιδρυματικός χρήστης μπορεί να δημιουργήσει μία υποτροφία συμπληρώνοντας όλα τα υποχρεωτικά πεδία στη φόρμα.



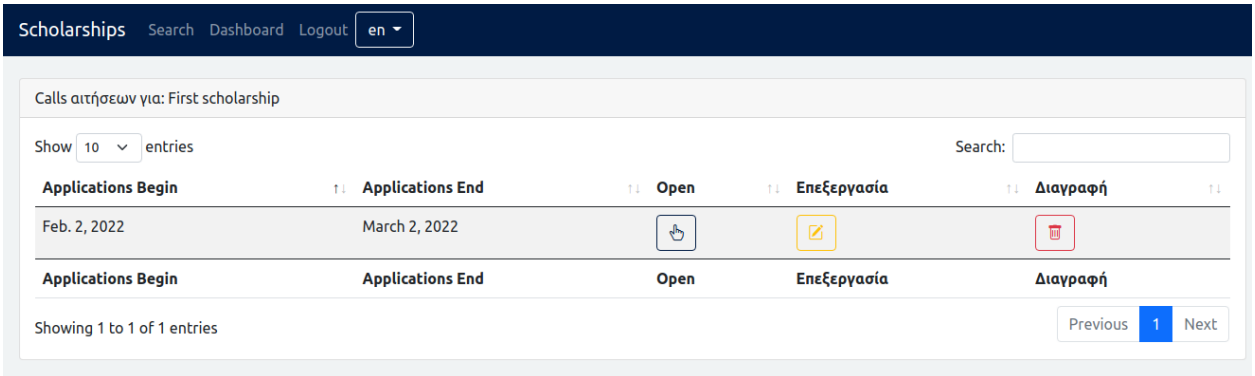
Εικόνα 21: Δημιουργία υποτροφίας

4.2.2 Δημιουργία και επεξεργασία calls αιτήσεων υποτροφίας

Ο ιδρυματικός χρήστης μπορεί να δημιουργήσει και να επεξεργαστεί calls για τις αιτήσεις των υποτροφιών του.



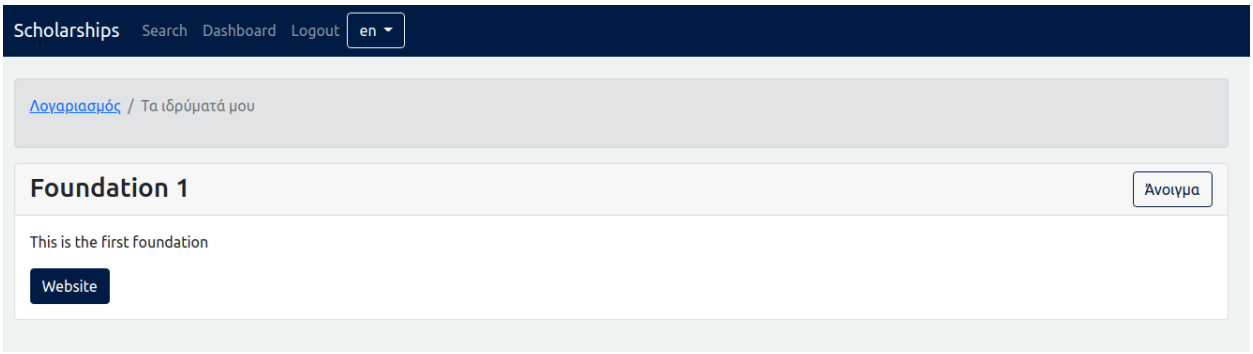
Εικόνα 22: Δημιουργία call αίτησης



Εικόνα 23: Προβολή calls αιτήσεων

4.2.3 Προβολή ιδρυμάτων χρήστη

Ο ιδρυματικός χρήστης μπορεί να δει σε ποια ιδρύματα ανήκει.



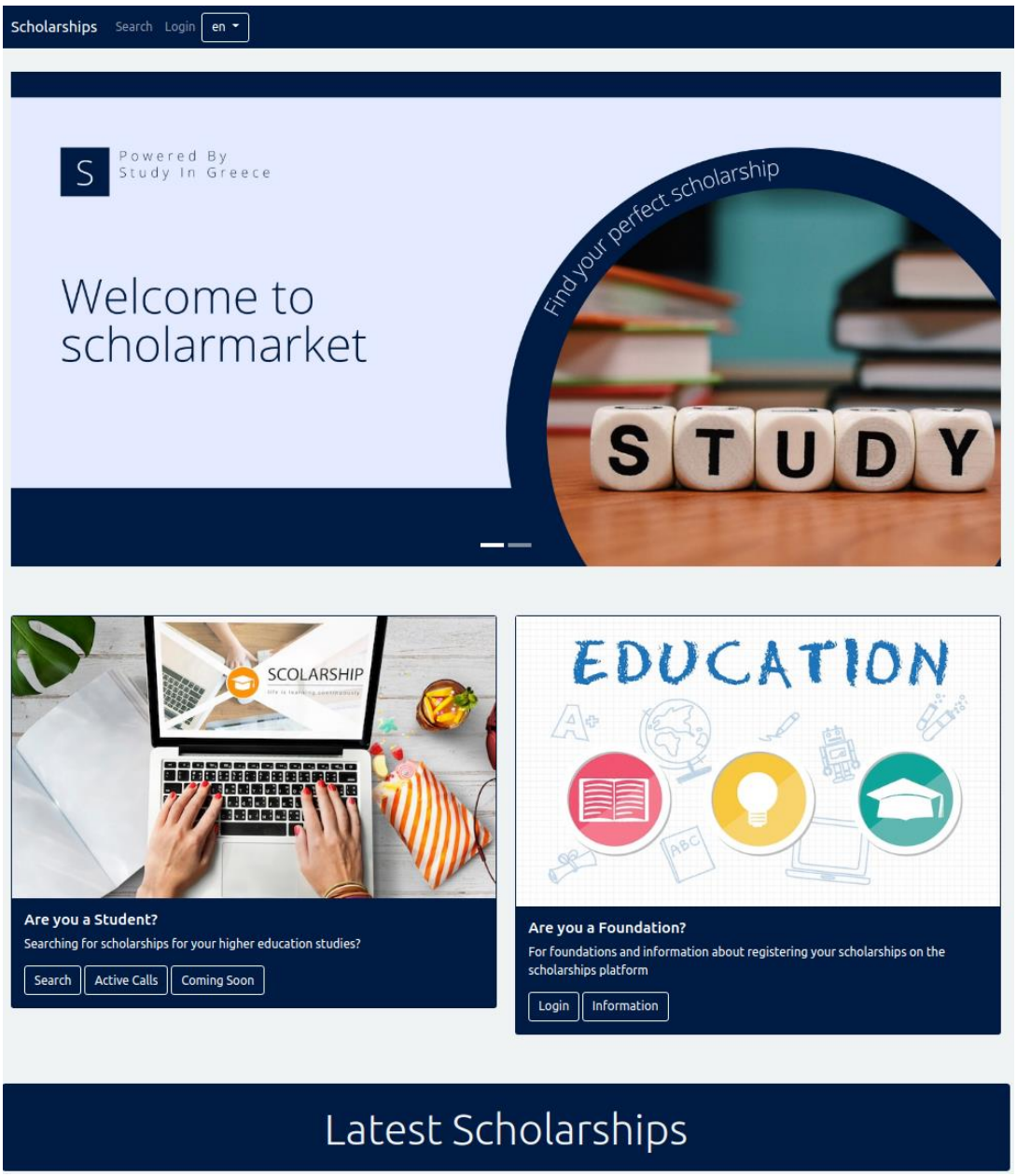
Εικόνα 24: Ιδρύματα χρήστη

4.3 Υποψήφιος

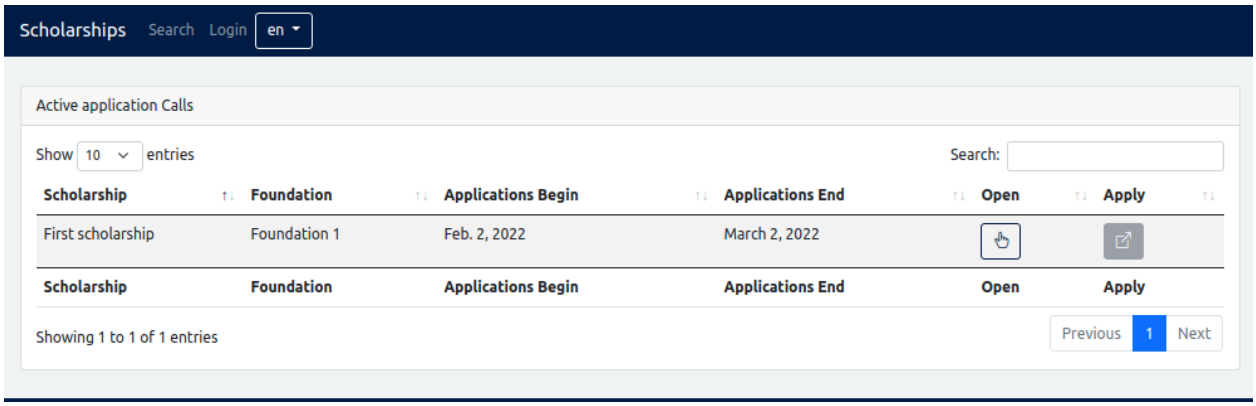
Ο υποψήφιος (ή μη αυθεντικοποιημένος χρήστης) είναι ο πιο συνήθης χρήστης της εφαρμογής. Τη χρησιμοποιεί για να ενημερωθεί για τις διαθέσιμες υποτροφίες που προσφέρονται από τα ιδρύματα και έχουν δημιουργήσει οι ιδρυματικοί χρήστες στην πλατφόρμα.

4.3.1 Αρχική σελίδα

Στην αρχική σελίδα της εφαρμογής ο απλός χρήστης/υποψήφιος, μπορεί να δει κάποιες πληροφορίες για συγκεκριμένες κατηγορίες υποτροφιών, όπως αυτές που έχουν active calls και εκείνες που θα ξεκινήσουν προσεχώς.



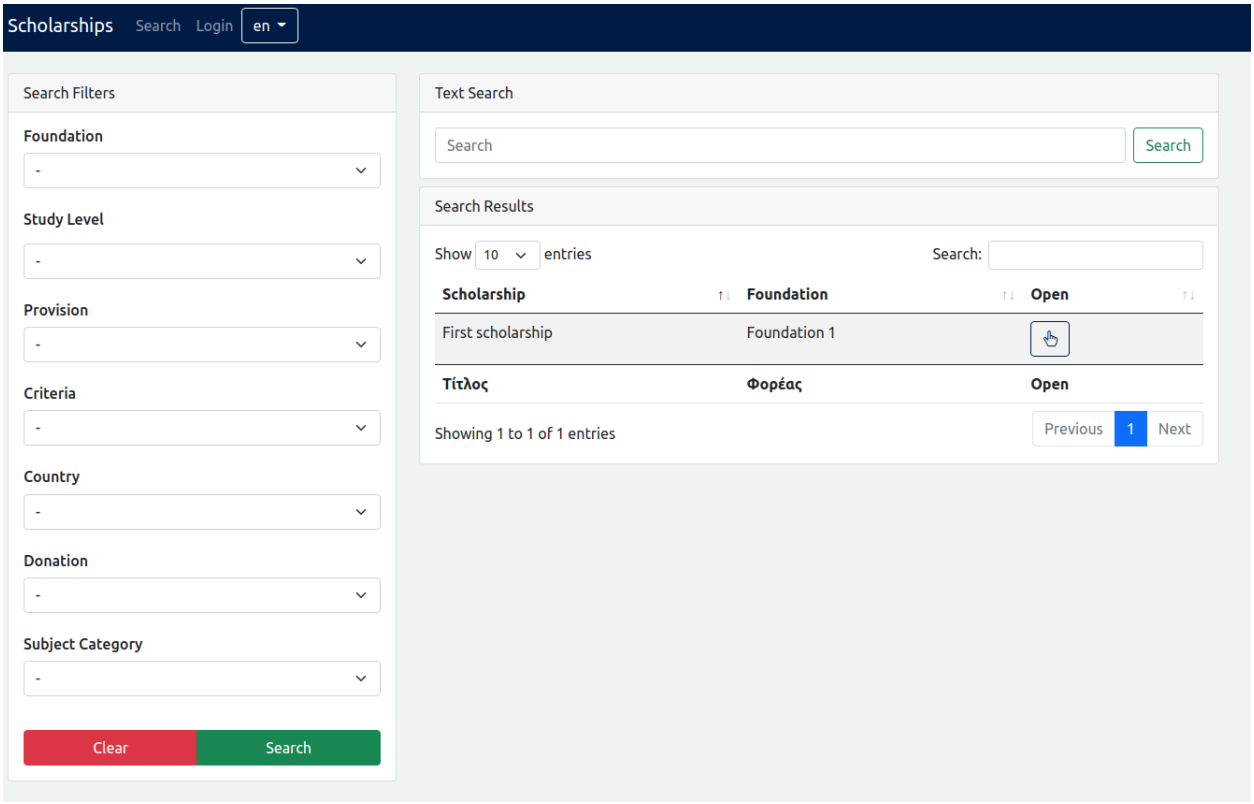
Εικόνα 25: Αρχική σελίδα



Εικόνα 26: Λίστα με τα active calls

4.3.2 Αναζήτηση υποτροφιών

Ο υποψήφιος μπορεί να αναζητήσει την υποτροφία που του ταιριάζει με βάση τα κριτήρια/φίλτρα που επιθυμεί. Μπορεί επίσης να κάνει αναζήτηση με βάση τον τίτλο της υποτροφίας ή την περιγραφή της.



Εικόνα 27: Αναζήτηση υποτροφίας

5. Μελλοντικές βελτιώσεις

5.1 Kubernetes

Μία βελτίωση που είναι ήδη στα σκαριά είναι η προετοιμασία του project για να γίνει το deployment του και η διαχείριση του με Kubernetes. Αυτό θα αυξήσει τη διαθεσιμότητα της εφαρμογής μέσω αυξημένου fault tolerance και θα της επιτρέψει να scaleάρει ανάλογα τη ζήτηση και τη χρήση της.

5.2 Υποσύστημα διαχείρισης αιτήσεων για υποτροφίες

Μία ακόμα ιδέα είναι η υλοποίηση ενός συστήματος αιτήσεων για τις υποτροφίες. Τα ιδρύματα θα μπορούσαν, εφόσον το επιλέξουν, να δέχονται αιτήσεις από τους υποψήφιους μέσω του συστήματος. Έτσι λύνεται ένα δυνητικό πρόβλημα των υποψήφιων, ότι χρειάζεται να παρακολουθούν πολλά διαφορετικά κανάλια επικοινωνίας για να ενημερώνονται για την εξέλιξη της αίτησης τους.

Βιβλιογραφία

{1} About Study in Greece. Ανακτήθηκε από https://studyingreece.edu.gr/el/schetika-me-emas

{2} Fullbright Greece υποτροφίες για Έλληνες πολίτες. Ανακτήθηκε από https://www.fulbright.gr/el/ypotrofies-gia-ellines-polites

{3} ΙΚΥ υποτροφίες. Ανακτήθηκε από https://www.iky.gr/el/upotrofies-gr/upotrofies-gr

{4} About Docker. Ανακτήθηκε από https://www.docker.com/why-docker

{5} About Django. Ανακτήθηκε από https://www.djangoproject.com/

{6} About Django Applications. Ανακτήθηκε από https://docs.djangoproject.com/en/4.0/ref/applications/

Παράρτημα Α’

Βασικά μοντέλα της εφαρμογής

```
class Foundation(models.Model):
    name = models.CharField('Ονομασία Ιδρύματος', max_length=200, default='')
    description = models.TextField('Περιγραφή Ιδρύματος')
    website = models.TextField('Ιστοσελίδα Ιδρύματος', blank=True)
    user = models.ManyToManyField(User)

    def __str__(self):
        return self.name
```

```
class Donation(models.Model):
    name = models.CharField('Ονομασία Κληροδοτήματος', max_length=200)
    description = models.TextField('Περιγραφή Κληροδοτήματος', blank=True)
    website = models.CharField('Ιστοσελίδα Κληροδοτήματος', max_length=250,
blank=True)
    foundation = models.ManyToManyField(Foundation)

    def __str__(self):
        return self.name
```

```
class Scholarship(models.Model):
    title = models.CharField('Τίτλος Υποτροφίας', max_length=200)
    description = models.TextField('Περιγραφή Υποτροφίας', blank=True)
    studies = models.ManyToManyField(StudyChoice)
    provision = models.ManyToManyField(ProvisionChoice)
    subject = models.ManyToManyField(SubjectChoice)
    criteria = models.ManyToManyField(CriteriaChoice)
    place = models.ManyToManyField(PlaceChoice)
    foundation = models.ForeignKey(Foundation, on_delete=models.RESTRICT)
    donation = models.ManyToManyField(Donation, blank=True)
    submitter = models.ManyToManyField(User)
    status = models.BooleanField('Ενεργό', default=True)

    def __str__(self):
        return self.title
```

```
class ScholarshipCall(models.Model):
    scholarship = models.ForeignKey(Scholarship, on_delete=models.CASCADE)
    call_publish = models.DateField('Έναρξη Εμφάνισης', blank=True)
    call_start = models.DateField('Έναρξη Αιτήσεων', blank=True)
    call_end = models.DateField('Λήξη Αιτήσεων', blank=True)
    website = models.TextField('Σύνδεσμος αιτήσεων', blank=True)

    def __str__(self):
        return "%s --- %s - %s" % (self.scholarship.title, self.call_start,
self.call_end)
```

```
class ProvisionChoice(models.Model):

    title = models.CharField('Είδος Παροχής', max_length=200)


    def __str__(self):

        return self.title
```

```
class StudyChoice(models.Model):
    title = models.CharField('Είδος Σπουδών', max_length=200)

    def __str__(self):
        return self.title
```

```
class SubjectChoice(models.Model):
    title = models.CharField('Θεματική Κατηγορία', max_length=200)

    def __str__(self):
        return self.title
```

```python
class CriteriaChoice(models.Model):
    title = models.CharField('Κριτήριο Επιλογής', max_length=200)

    def __str__(self):
        return self.title
```

```python
class PlaceChoice(models.Model):
    title = models.CharField('Σπουδές', max_length=200)

    def __str__(self):
        return self.title
```

```python
class MobilityChoice(models.Model):
    title = models.CharField('Κινητικότητα', max_length=200)

    def __str__(self):
        return self.title
```

Παράρτημα Β’

Σημαντικά κομμάτια κώδικα:

- Δημιουργία υποτροφίας

```
class ScholarshipCreateView(LoginRequiredMixin, CreateView):
    # model = Scholarship
    template_name = 'scholars/private/scholarship_create.html'
    form_class = NewScholarshipForm

    def get_form_kwargs(self):
        kwargs = super(ScholarshipCreateView, self).get_form_kwargs()
        kwargs["request"] = self.request
        return kwargs

    def form_valid(self, form):
        # This method is called when valid form data has been POSTed.
        # It should return an HttpResponse.
        instance = form.save(commit=False)
        instance.save()
        form.save_m2m()
        users = User.objects.filter(pk=self.request.user.id)
        instance.submitter.set(users)
        return super().form_valid(form)

    def get_success_url(self):
        return reverse_lazy('scholars-user')
```

- Αναζήτηση υποτροφίας

```
class ScholarshipsListView(ListView):
    model = Scholarship
    context_object_name = 'scholarships'
    template_name = 'scholars/public/scholarship_list.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['foundations'] = Foundation.objects.all()
        context['studies'] = StudyChoice.objects.all()
        context['provisions'] = ProvisionChoice.objects.all()
        context['criteria'] = CriteriaChoice.objects.all()
        context['cities'] = PlaceChoice.objects.all()
        context['donations'] = Donation.objects.all()
        context['subjects'] = SubjectChoice.objects.all()
        return context

    def get_queryset(self):
        # Free text form is submitted
        search = self.request.GET.get('search')
        if search:
            # Use unaccent like this title_el__unaccent__icontains
            return Scholarship.objects.filter(Q(status=True) &

(Q(title_el__icontains=search) | Q(title_en__icontains=search) |

Q(description_el__icontains=search) |
Q(description_en__icontains=search)))

        # Filter form is submitted
        if self.request.GET.get('foundation'):
            query = Q(status=True)
            for kw in self.request.GET:
                value = self.request.GET[kw]
                if value != '0':
                    query &= Q(**{kw: value})
            return Scholarship.objects.filter(query)

        return Scholarship.objects.filter(status=True).all()
```

- Δημιουργία call για υποτροφία

```
class ApplicationCallCreateView(CreateView):
    model = ScholarshipCall
    form_class = ApplicationCallForm
    template_name = 'scholars/private/call_create.html'

    def form_valid(self, form):
        scholarship = get_object_or_404(Scholarship,
id=self.kwargs['pk'])
        form.instance.scholarship = scholarship
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('scholars-user')
```