



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICA ŞI
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

DUOALGO

Absolvent

Livia Măgureanu

Coordonator științific

Lect. dr Adrian-Marius Dumitran

Bucureşti, iunie 2022

Rezumat

Aplicația web descrisă în această lucrare se numește DuoAlgo și are scopul de a facilita pregătirea pentru programarea competitivă. Această platformă se adresează tuturor persoanelor care își doresc să învețe, indiferent dacă sunt elevi de liceu care se pregătesc pentru olimpiadă, studenți interesați de diverse concursuri de algoritmică, sau simpli pașionați de programare.

Aplicația este compusă dintr-o colecție de lecții. Fiecare dintre ele este concepută să fie cât mai ușor de înțeles și să ofere toate resursele necesare pentru asimilarea informației, de la noțiuni de bază la probleme suplimentare.

În același timp, aplicația urmărește progresul fiecărui utilizator și se asigură că acesta nu încearcă să abordeze un subiect pentru care încă nu are suficiente cunoștințe.

Tema acestei aplicații este similară cu ce numeroase alte platforme de pregătire încearcă să ofere prin articole, blog-uri, training path-uri etc., dar, spre deosebire de ele, centralizează toată această informație într-un produs care se axează doar pe partea educațională.

Abstract

The web application described in this paper is called DuoAlgo and is intended to facilitate preparation for competitive programming. This platform is aimed at all people who want to learn, whether they are high school students preparing for the Olympics, students interested in various algorithmic competitions, or just passionate about programming. The application consists of a collection of lessons. Each lesson is designed to be as easy to understand as possible and to provide all the resources needed to assimilate the information, from basics ideas to additional problems.

At the same time, the app tracks the progress of each user and ensures that they are not trying to address a topic for which they do not yet have sufficient knowledge.

The goal of this application is similar to what many other training platforms try to offer through articles, blogs, training paths, etc., but, unlike them, it centralizes all this information in a product that focuses only on the educational (theoretical) side.

Cuprins

1 Introducere	5
1.1 Ideea și scopul	5
1.2 Scurtă descriere a tehnologiilor	6
1.2.1 Backend	6
1.2.2 Frontend	6
1.2.3 Diverse	7
2 Design	8
2.1 Concept	8
2.2 DAG-ul lecțiilor	9
2.3 Împărțirea pe stage-uri	10
2.3.1 Stage 1	10
2.3.2 Stage 2	11
2.3.3 Stage 3	11
2.3.4 Stage 4	12
2.3.5 Stage 5	13
2.3.6 Stage 6	13
2.3.7 Stage 7	14
2.3.8 Stage 8	14
2.4 Structura unei lectii	15
2.5 Monitorizarea progresului	15
2.6 Task-uri de pe platforme externe	16
2.7 Arborele topic-urilor	16
2.8 Baza de date	17
3 Interfață	21
3.1 Rolul user	21
3.2 Rolul administrator	21
3.2.1 Useri și permisiuni	22
3.2.2 Adăugarea lecțiilor	22

4 Detalii de implementare	23
4.1 Arhitectura	23
4.2 Limbaje si framework-uri	25
4.3 Backend	25
4.3.1 Models (interacțiunea cu baza de date)	25
4.3.2 Views	27
4.3.3 Templates	28
4.4 Frontend	28
4.4.1 Comunicarea cu backend-ul	28
4.4.2 Cea mai costisitoare cerere	29
4.5 Development și procesul de deploy	29
4.5.1 VCS	29
4.5.2 Cloud setup	29
5 Concluzii	30
Bibliografie	31

Capitolul 1

Introducere

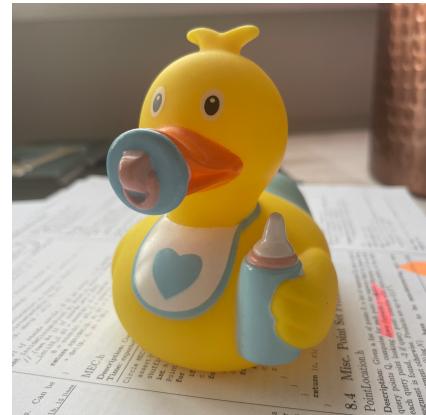
1.1 Ideea și scopul

Ideea aplicației a venit din nevoie de crea un suport de curs de programare competitivă pentru elevii mei de pe care să poată preferabil să învețe și pe cont propriu. Tot căutând resurse educationale am descoperit că, deși există o multitudine de platforme de pregătire și de "training path"-uri, majoritatea grupează articolele / lecțiile în funcție de aria de materie din care face parte subiectul, fără să ofere o ordine organică în care studenții autodidacți să poată învăța algoritmii prezentati. Design-ul aplicației DuoAlgo este conceput pentru a fi user-friendly, iar conținutul, pentru a fi ușor de parcurs și înțeles atât de elevi și studenți, cât și de amatori.

Câteva exemple de platforme care oferă servicii similare, dar cu problema descrisă mai sus pe care încerc să o soluționez, ar fi:

- <https://www.infoarena.ro/training-path>
- <https://cp-algorithms.com/navigation.html>
- <https://codeforces.com/catalog>

Numele aplicației a venit din combinația între cuvântul algoritmică și Duolingo (întrucât conceptul acesta este deja unul cunoscut și foarte similar cu ce îmi propun să realizez). Logo-ul a fost inspirat din practica câtorva programatori (printre care și eu) de a folosi "rubber duck debugging". Folosirea unui element ludic în acest context are avantajul de a face copiii să se simtă mai în largul lor atunci când lucrează și de a adăuga un plus de entuziasm (testat anul acesta pe elevi începători de clasele a 6-a și a 9-a).



1.2 Scurtă descriere a tehnologiilor

Limbaje de programare folosite:
$$\begin{cases} \text{Python 3.8.1 (Django 4.0.4)} & - \text{backend} \\ \text{JavaScript (React 18.2.0)} & - \text{frontend} \end{cases}$$

IDE-uri:
$$\begin{cases} \text{PyCharm from JetBrains} & - \text{backend} \\ \text{WebStorm from JetBrains} & - \text{frontend} \end{cases}$$

1.2.1 Backend

Limbaj **Python 3.8.1**

Framework **Django 4.0.4**

<https://www.djangoproject.com/start/overview/>

Deși presupune o oarecare pierdere a flexibilității, am optat în favoarea folosirii unui framework pentru a ușura dezvoltarea și mențenanța aplicației. Aceasta decizie a fost susținută și de beneficiile pe care le oferă cum ar fi sistemul de autentificare, sistemul de securitate și ORM-ul integrate.

Baza de date **PostgreSQL**

<https://www.postgresql.org>

A fost ales pentru că este probabil cea mai bună bază de date pentru proiecte complexe, dar dimensiuni nu foarte mari. În plus, Django oferă suport integrat pentru ea.

IDE **PyCharm de la JetBrains**

<https://www.jetbrains.com/pycharm/>

Mediu de programare versatil, asigură navigarea ușoară prin proiecte voluminoase, suficient de complex pentru a fi comod, dar nu atât cât să devină greoi. Alte beneficii: VCS, terminal în aplicație, suport de Django, folosit la scară largă în industrie, gratuit ca parte din pachetul educațional de la GitHub.

1.2.2 Frontend

Limbaj **JavaScript**

Framework **React 18.2.0**

<https://reactjs.org>

Deși tehnic vorbind nu este framework, ci colecție de librării, React prezintă în continuare aceleași avantaje, fiind creată și susținută de Facebook. Iese în evidență față de alte opțiuni datorită adoptării la scară largă care a generat o comunitate mare de contribuitori și, prin urmare, la o abundență de resurse.

API **Axios**

<https://github.com/axios/axios>

Bibliotecă open-source pentru comunicarea frontend-ului cu backend-ul. Am mers pe această variantă în detrimentul unora mai tradiționale încărcarea paginii de către backend.

IDE **WebStorm de la JetBrains**

<https://www.jetbrains.com/webstorm/>

Mediu de programare versatil, asigură navigarea ușoară prin proiecte voluminoase, suficient de complex pentru a fi comod, dar nu atât cât să devină greoi. Alte beneficii: VCS, terminal în aplicație, suport de React, folosit la scară largă în industrie, gratuit ca parte din pachetul educational de la GitHub.

1.2.3 Diverse

Registrar **Name.com**

<https://www.name.com>

Serviciu care gestionează domeniul și DNS-ul.

Cloud **DigitalOcean**

<https://www.digitalocean.com>

Serviciu ce oferă infrastructură în cloud.

VCS **git (GitHub)**

<https://github.com/Oepeling/DuoAlgo>

Fun fact: Linus Torvalds a inventat git-ul, o variantă net superioară a sistemelor existente la momentul respectiv, pentru a gestiona dezvoltarea Linux-ului. [2]

Documentație **Latex (Overleaf)**

<https://www.overleaf.com/read/gynbgxfwtwkm>

Capitolul 2

Design

2.1 Concept

În ziua de azi, o parte tot mai mare a populației se orientează către învățarea și aprofundarea pe cont propriu, fie apelând la cursuri online, fie căutând și citind articole pe internet, dar în principal fără îndrumarea unui profesor. Dincolo de părțile evidente, pandemia prin care tocmai am trecut a avut și un efect pozitiv: ne-a arătat cât de mult ne putem simplifica viața profitând de uneltele digitale care ne sunt din ce în ce mai la îndemâna, amplificând fenomenul menționat.

Întrucât nu toată lumea este autodidactă și folosirea resurselor pe care le avem la dispoziție necesită ca individul să aibă deja un anumit nivel de maturitate în domeniul de interes, acest fapt a generat o cerere pe piața online care a dus la popularizarea și perfecționarea cursurilor online, dar și la monetizarea lor. Însă eu sunt de părere că accesul la cunoștere ar trebui să fie gratuit la orice nivel. Cu alte cuvinte, dacă oamenii vor să învețe, ar trebui să fie susținuți și ajutați. Astfel s-a născut această aplicație.

Fiecare om are nevoi diferite și propriul ritm, prin urmare, am încercat să găsesc un mod cât mai puțin restrictiv de a sugera o ordine de parcurgere a lecțiilor de pe platformă. Lecțiile sunt aranjate folosind două structuri:

1. **DAG-ul (directed acyclic graph) lecțiilor** – un graf orientat aciclic în care sunt reprezentate dependințele fiecărei lecții
2. **Stage-uri** – o împărțire pe niveluri de studiu pentru a asigura că toți elevii progresează concomitent pe mai multe arii în loc să se axeze pe una în particular

2.2 DAG-ul lecțiilor

În primul rând, trebuie menționat că în acest prototip, lecția reprezintă unitatea de bază. Prin urmare este relativ scurtă și la obiect, nedivizibilă și cuprinde o singură temă principală. Această abordare este cea care permite ca graful format din dependințele lecțiilor să fie unul aciclic.

Aproape orice lecție vine cu un set de cunoștințe necesare pe care un elev ar trebui să îl parcurgă și (preferabil) stăpânească în prealabil. Aceste cunoștințe formează muchiile grafului orientat aciclic menționat.

După cum se poate intui, numărul dependințelor (directe și indirecte) ale unei lecții sunt cu atât mai multe cu cât nivelul ei este mai avansat.

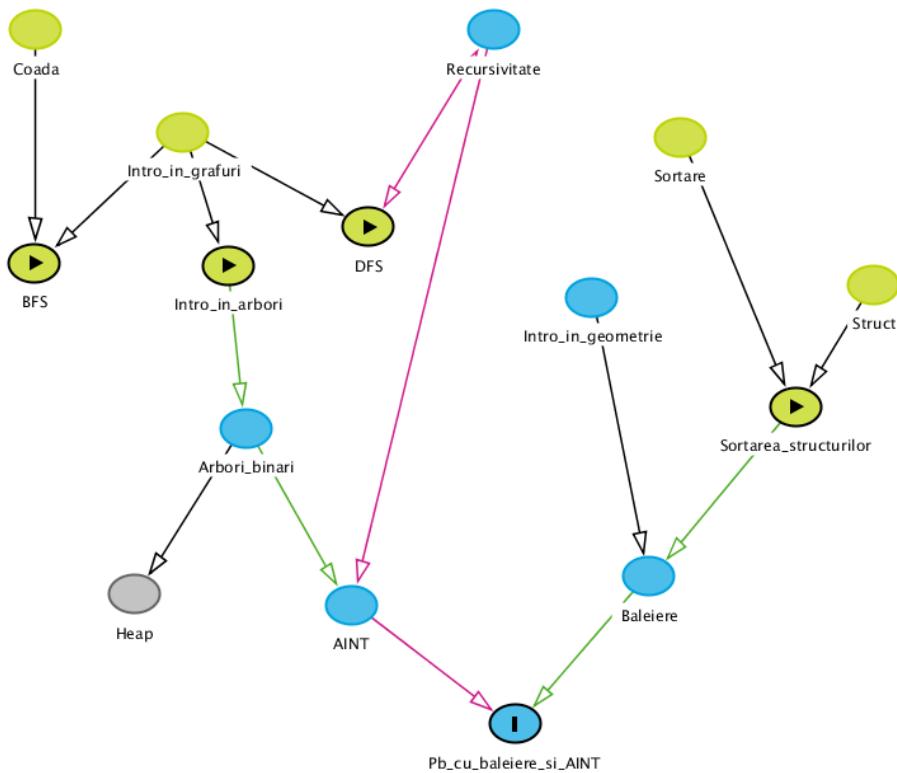


Figura 2.1: Exemplu de subgraf în DAG-ul lecțiilor

Mai sus se află un exemplu de subgraf al cărui scop este ajungerea la o lecție care prezintă o categorie de probleme relativ des întâlnită: probleme care combină baleierea cu o structură de date, în genere arborii de intervale (AINT). Din această categorie fac parte următoarele probleme:

- <https://infoarena.ro/problema/zoo>
- <https://infoarena.ro/problema/parcele>

- <https://infoarena.ro/problema/tri2>
- <https://infoarena.ro/problema/cadrane>

Am reprezentat cu verde lecțiile deja parcuse de elevul în cauză și cu albastru pe cele pe care le-ar mai avea de parcurs pentru a-și atinge scopul. A se remarcă faptul că AINT-ul are ca dependință și recursivitatea și că, deși știm că lecția respectivă a fost parcursă întrucât era necesara pentru DFS, ea nu este un strămoș al nodului AINT, deci parcurgerea acesteia tot va trebui verificată la momentul când elevul ajunge în punctul respectiv.

Folosind această structură, se creează un minim necesar de ordonare al lecțiilor, dar apar două probleme majore. Prima este că un elev în continuare poate să parcurgă o parte foarte restrânsă din noduri, ajungând să nu aibă cunoștințe suficiente de diverse. A doua ar fi că sortarea topologică a acestui graf în continuare nu duce neaparat la o ordine sănătoasă de parcurgere a materiei. Inițial am luat în considerare posibilitatea de a adăuga o serie de muchii fictive pentru a forța ordinea anumitor noduri, însă această soluție nu era consistentă cu modul în care a fost definit acest DAG. Rezolvarea a venit prin a împărți mai departe lecțiile pe nivele (stage-uri), ce vor fi explicate mai pe larg în secțiunea următoare.

2.3 Împărțirea pe stage-uri

Un **stage** reprezintă grupa / nivelul de studii, cuprinde o serie de lecții compacte în DAG (i.e. dacă există 3 noduri a , b și c în să existe muchie de la a la b și de la b la c , dacă a și c se află în același stage, atunci acolo va fi și b). Stage-urile sunt ordonate liniar (trebuie parcuse în ordine).

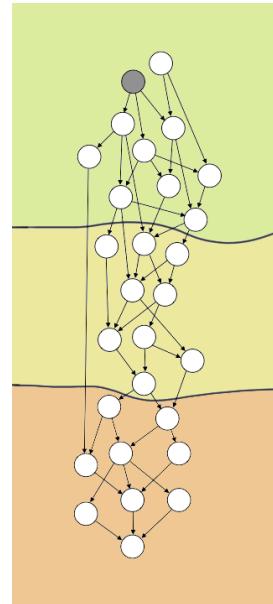
Am împărțit lecțiile în 8 etape care corespund parțial împărțirii materiei pe clase pentru elevii care încep programarea competitivă (pentru olimpiadă) din clasa a 5-a.

2.3.1 Stage 1

Este conceput să aducă o persoană complet începătoare la nivelul la care să stăpânească sintaxa limbajului suport (în cazul nostru C/C++) la un nivel de bază, să poată transpune problemele în idei de rezolvare, ideile în cod și să poată combina algoritmi simpli.

Include, dar nu este limitat la:

- Introducere în sintaxă (citire, afișare, int)
- Operații aritmetice de bază (+, -, *, /, %)



- Divizibilitate, if-uri, operații prescurtate ($+=$, $-=$ etc)
- Prelucrarea cifrelor unui număr cu număr fixat de cifre
- While și aplicații pe prelucrarea cifrelor unui număr
- Cel mai mare divizor comun, cel mai mic multiplu comun
- For, sirul lui Fibonacci și vectori
- Introducere în siruri de caractere
- Funcții (subprograme)

2.3.2 Stage 2

Include, dar nu este limitat la:

- Prelucrare de vectori
- Interclasare
- Intersecție de intervale
- Siruri de caractere
- Stivă
- Coadă
- Căutare binară (clasică)
- Ciurul lui Eratostene
- Indicatorul lui Euler
- Euclid extins
- Matrice. Parcurgeri
- Sume partiale
- Menul lui Mars
- Sume partiale 2D. Dreptunghi de sumă maximă

2.3.3 Stage 3

Include, dar nu este limitat la:

- Căutare binară (cu pas)
- Lee
- Recursivitate
- Fill

- Sortare
- Struct
- Greedy
- Programare dinamică
- Divide et impera
- Subşir crescător maximal
- Stive crescătoare / descrescătoare
- Backtracking
- Permutări, combinări și aranjamente
- Deque. Sliding window
- Introducere în geometrie

2.3.4 Stage 4

Include, dar nu este limitat la:

- Introducere în grafuri
- DFS, BFS
- Multi-source BFS
- BFS 0-1
- Heap
- Dijkstra
- Bellman-Ford
- KMP, Z, Manacher
- Hash tables
- Numere mari
- Păduri de multimi disjuncte
- Combinatorică și probleme de numărare. Triunghiul lui Pascal
- Geometrie
- Optimizări pe programarea dinamică

2.3.5 Stage 5

Include, dar nu este limitat la:

- Roy-Floyd
- Operații pe biți
- Dinamică pe stări exponențiale
- Ciclu hamiltonian (de cost minim)
- STL
- Trie
- Strămoși (binary lifting) și LCA
- Dinamică pe arbore
- APM (Prim, Kruskall, Borouvka)
- AIB
- AINT
- Batog
- Diametru unui arbore
- Liniarizare de arbori
- Baleiere
- NIM
- Componente tare conexe (Kosaraju)
- Ciclu eulerian
- Înfășurătoare convexă

2.3.6 Stage 6

Include, dar nu este limitat la:

- Suffix array
- Teorema chinezescă a resturilor
- Greedy cu invariant
- Căutare binară în paralel
- Mo
- Knuth

- Centroid
- Componente tare conexe (Tarjan)
- Li-Chao
- RMQ
- AINT cu lazy
- SOS (Sum over subsets)
- Sprague-Grundy
- Transformarea produsului în sumă (probleme cu \ln)
- Optimizări mai avansate pe programarea dinamică

2.3.7 Stage 7

Incluzie, dar nu este limitat la:

- Izomorfism de arbori
- Lema mariajelor
- Flux maxim (Ford Fulkerson, Edmonds-Karp)
- Flux maxim (Dinic)
- Cuplaj maxim
- Tăietură minimă
- Flux maxim de cost minim
- Cuplaj maxim de cost minim
- Minimum vertex cover
- Sămenul de la Aliens
- Teorema lui Lucas
- Matrice (produs, ridicare la putere, dinamici liniare)
- Circulație
- Heavy-light decomposition

2.3.8 Stage 8

Acest stage, fiind unul de perfecționare, când elevul deja are cunoștințe de teorie foarte avansate, va include în principal probleme interesante cu tehnici noi, abordări ad-hoc și algoritmi sau structuri de date nou-apărute. Blogurile de pe Codeforces prezintă o abundență de astfel de articole. Câteva exemple se pot găsi aici: <https://codeforces.com/blog/tourist>

2.4 Structura unei lecții

Titlu Clar și ușor de înțeles, are scopul de a oferi suficiente, dar nu prea multe informații despre lecție.

Autor Planul inițial nu includea posibilitatea de a avea mai mulți autori, dar povestindu-le alor persoane din comunitatea de programare competitivă despre acest proiect, am descoperit că mulți ar vrea să se implice (printre care și foarte mulți elevi de liceu care vor să-și susțină colegii mai mici). De asemenea, ar fi păcat să nu profităm de resursele deja existente și de diversitatea de subiecte pe care un număr mai mare de autori o poate acoperi.

Stage Fiecare lecție face parte dintr-un stage. Modul în care sunt organizate stage-urile a fost explicat în secțiunea anterioară.

Topic Având în vedere că o lecție este concepută să fie cea mai mică unitate posibilă, în mod ideal ar trebui să aparțină de un singur topic principal și, eventual, 1-2 secundare. Pentru mai multe detalii despre ce este un topic, vezi secțiunea dedicată.

Conținut Cuprinde textul lecției în sine, scris în format markdown și reprezintă elementul central (aprox. 90% din total).

Cod Exemple de cod explicat (optional). În unele cazuri, codul se poate afla direct în corpul lecției, însă există și cazuri în care discursul este mai mult de natură teoretică și punerea în aplicare are caracter de anexă.

Task-uri Task-uri de antrenament. Include atât probleme explicate în lecție, cât și probleme extra de rezolvat pe cont propriu.

Dependințe Lecțiile care trebuie parcuse în prealabil celei curente.

2.5 Monitorizarea progresului

Pentru fiecare utilizator sunt stocate lecțiile pe care acesta le-a parcurs și, dacă este cazul, lecția curentă care reprezintă o lecție începută și neterminată. Pentru simplitate, vom presupune că un individ nu parurge mai multe cursuri în același timp. Acest fapt ar trebui să vină natural din felul în care sunt concepute aceste cursuri (scurte și la obiect, cu scopul de a fi acoperite dintr-o singură sesiune de învățat). În acest scop, o lecție este marcată ca fiind începută și, odată parcursă, poate fi marcată ca atare de către persoana în cauză.

În paralel are loc și o monitorizare a progresului pe problemele propuse. Întrucât problemele de antrenament pot fi de pe o varietate de platforme, în design-ul actual utilizatorul va trebui să își treacă manual scorul pentru ca acesta să apară și în aplicație.

Pe viitor planul este ca progresul fiecărui utilizator să fie monitorizat automat, cu cât mai puțină implicare umană.

2.6 Task-uri de pe platforme externe

Un task reprezintă o problemă de pe o platformă externă de pregătire, de obicei cu sistem de evaluare, în limba română sau în limba engleză. Informațiile stocate împreună cu un task au fost alese pentru a ajuta atât elevii care încearcă să îl rezolve, cât și administratorii / autorii / profesorii în căutare de probleme.

În primul rând, apar datele de bază ale problemei: $\begin{cases} \text{Numele problemei} \\ \text{Platforma pe care se află problema} \\ \text{Link către problemă} \end{cases}$

În al doilea rând, apar date care sunt utile pentru cineva în căutare de probleme și care

ajută la identificarea problemei și a soluției: $\begin{cases} \text{Scurtă descriere} \\ \text{Tag-uri de rezolvare (ex.: eliminare gaussiană)} \\ \text{Link către soluție} \end{cases}$

În al treilea rând, apar datele care să ajute un elev să rezolve problema (în caz că se blochează). În acest scop, am conceput un sistem de indicii progresive de rezolvare care culminează cu prezentarea soluției în sine și link-ul către codul rezolvării.

2.7 Arborele topic-urilor

Nu are în sine scop pentru utilizator, ci este conceput pentru a servi administratorilor aplicației când vor să încadreze corect o lecție nou-adăugată sau să atribuie tag-uri de rezolvare a problemelor. În plus, permite indexarea amândurora (lecții și probleme) după categoria de care aparțin.

Arborele topic-urilor are 3 niveluri (dacă nu luăm în considerare rădăcina) și a rezultat din combinarea mai multor abordări asemănătoare printre care se numără:

- <https://www.infoarena.ro/training-path>
- <https://cp-algorithms.com/navigation.html>
- <https://codeforces.com/catalog>
- <https://codeforces.com/blog/entry/91363>

Mai jos se află o schemă parțială a arborelui resultant. O mare parte din arborele complet nu a fost reprezentată în exemplul de mai jos din considerente de spațiu.

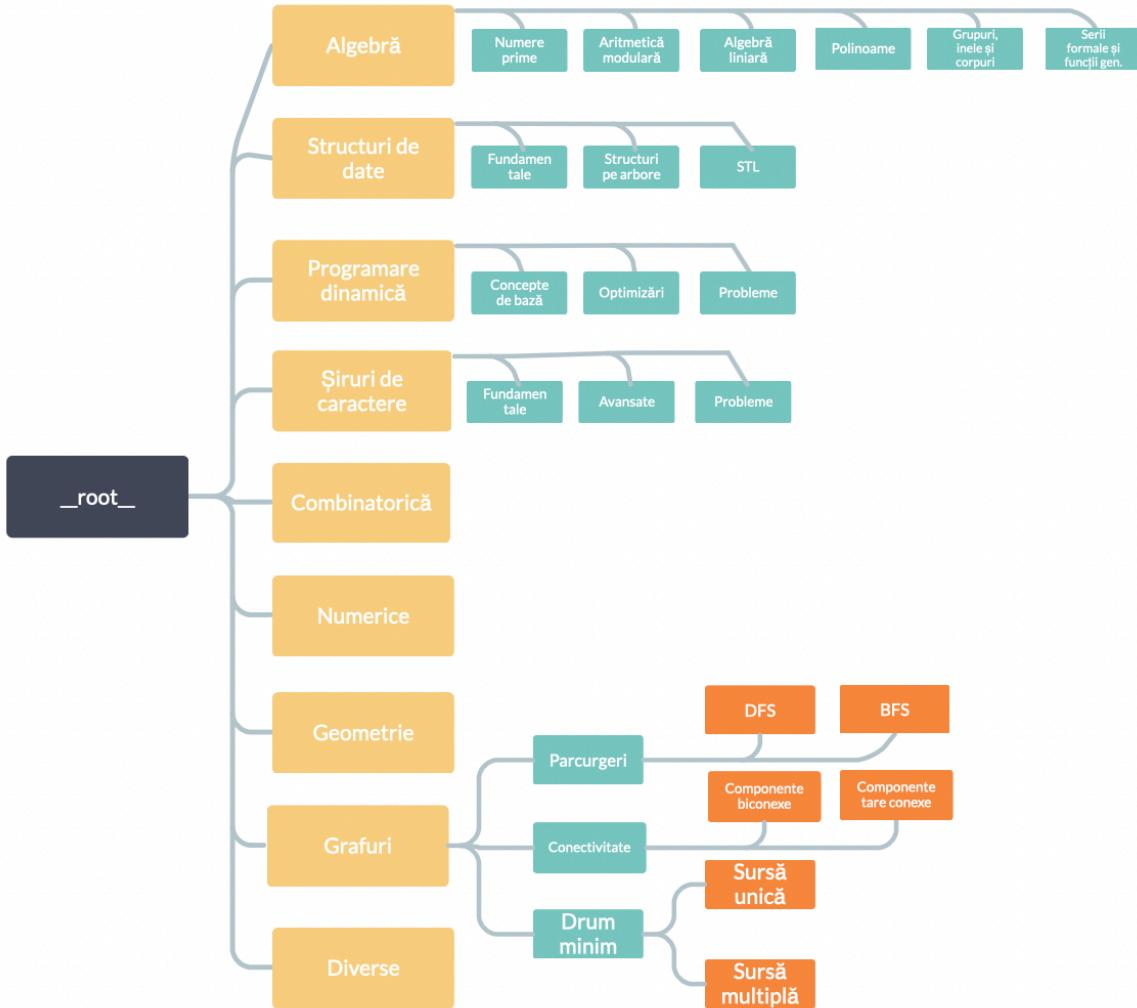


Figura 2.2: Arborele topic-urilor

2.8 Baza de date

Pentru a satisface nevoile de stocare pentru fiecare dintre aspectele de design descrise în secțiunile anterioare, am construit următoarea schemă pentru baza de date a aplicației.

Coloanele marcate cu un simbol în formă de cheie marchează cheia primară a tabelului respectiv, iar simbolul în formă de fulg de zăpadă indică proprietatea de unicitate a coloanei.

După cum se poate observa, toate relațiile și cerințele descrise în secțiunile anterioare sunt prezente în această schemă și vor fi detaliate mai jos.

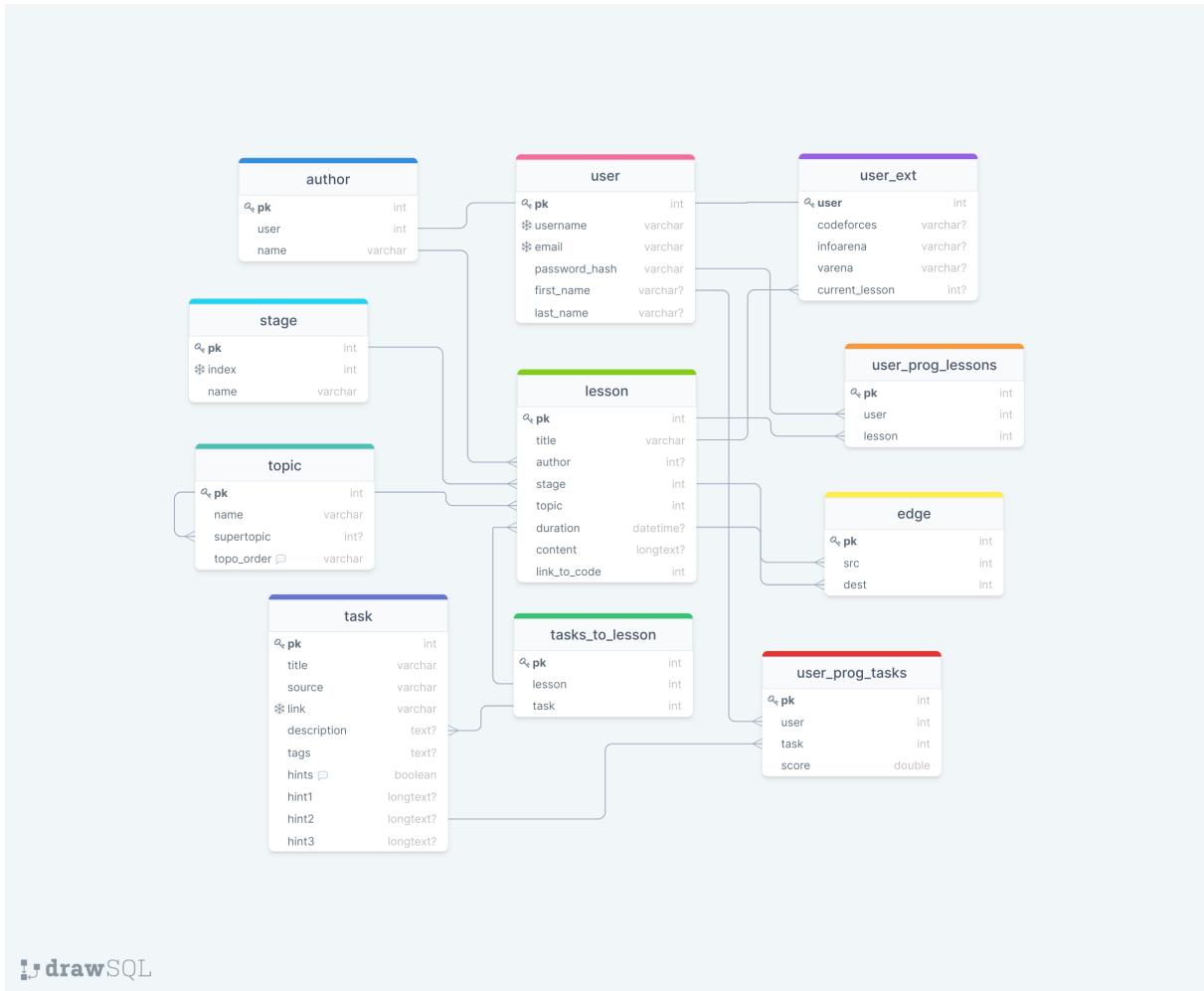


Figura 2.3: Schema bazei de date

- **user** – conține informațiile de bază ale fiecărui user
 - **pk** – primary key
 - **username**
 - **email**
 - **password_hash** – din motive de securitate, este salvat doar un hash al parolei
 - **first_name**
 - **last_name**
- **user_ext** – conține informații adiționale pentru un user
 - **user** – relație one-to-one cu **user**
 - **codeforces** – username-ul de pe codeforces
 - **infoarena** – username-ul de pe infoarena
 - **varena** – username-ul de pe varena

- `current_lesson` – relație many-to-one cu `lesson`
- `user_prog_lessons` – simulează o relație de tip many-to-many pentru a monitoriza progresul unui user
 - `pk` – primary key
 - `user` – relație many-to-one cu `user`
 - `lesson` – relație many-to-one cu `lesson`
- `user_prog_tasks` – simulează o relație de tip many-to-many pentru a monitoriza progresul unui user
 - `pk` – primary key
 - `user` – relație many-to-one cu `user`
 - `task` – relație many-to-one cu `task`
- `author` – entitățile autorilor pot fi optional legate de un cont
 - `pk` – primary key
 - `user` – (optional) relație one-to-one cu `user`
 - `name`
- `stage` – entitățile stage-urilor (asigură că fiecare lecție aparține unui nivel standard)
 - `pk` – primary key
 - `index` – pentru ordonarea mai usoară
 - `name`
- `topic` – entitățile topic-urilor (asigură că fiecare lecție este încadrată într-un topic standard)
 - `pk` – primary key
 - `name`
 - `supertopic` – reprezintă topicul părinte în structura arborescentă
 - `topo_order` – (autogenerat) pentru sortarea în preordine și postordine
- `task` – stochează probleme de pe platforme externe
 - `pk` – primary key
 - `title`
 - `source`

- **link**
 - **description**
 - **tags**
 - **hints** – marchează dacă task-ul are sau nu un sistem de hint-uri
 - **hint1** – (optional)
 - **hint2** – (optional)
 - **hint3** – (optional)
- **lesson** – ține toate componentele unei lecții
 - **pk** – primary key
 - **title**
 - **author** – relație many-to-one cu **author**
 - **stage** – relație many-to-one cu **stage**
 - **topic** – relație many-to-one cu **topic**
 - **duration** – (optional) durata estimată a lecției
 - **content** – corpul lecției în format markdown
 - **link_to_code** – (optional)
- **task_to_lesson** – simulează o relație de tip many-to-many
 - **pk** – primary key
 - **lesson** – relație many-to-one cu **lesson**
 - **task** – relație many-to-one cu **task**
- **edge** – simulează o relație de tip many-to-many între lecții (pentru graful aciclic)
 - **pk** – primary key
 - **lesson** – relație many-to-one cu **lesson**
 - **lesson** – relație many-to-one cu **lesson**

Capitolul 3

Interfață

3.1 Rolul user

Rolul utilizatorului este unul foarte simplu. După cum am menționat, oamenii, în special elevii și studenții, se orientează din ce în ce mai mult spre studiul pe cont propriu într-un mediu online și fix asta intenționează acest proiect să le ofere: o platformă simplistă și ușor de folosit care să le permită să învețe în ritmul lor natural, dar fără să o ia pe arătură.

Întrucât unul dintre scopuri este monitorizarea progresului studentului, accesul este limitat pentru utilizatorii neînregistrați. Mai exact, pentru a folosi platforma, o persoană trebuie să aibă cont, însă, din motive de practicalitate (pentru partajare, citări etc), am decis ca accesul prin link direct la lectii să fie permis.

Traseul unui utilizator începe pe pagina de login. De acolo ajunge pe pagina unde sunt afișate lecțiile. Având în vedere dimensiunea structurii care gestionează lecțiile, ar fi impractic să afișăm întreg DAG-ul deodată. Drept consecință, în interiorul fiecărui stage lecțiile vor fi împărțite pe niveluri folosind o sortare topologică ușor modificată. Pagina va conține aproximativ 3 niveluri, inițial centrate în punctul curent în procesul de studiu. De acolo, un utilizator se poate deplasa spre alte niveluri sau poate alege o lecție pe care să o înceapă, dar doar dacă îndeplinește toate criteriile (să fi parcurs dependințele lecției deja). Când termină de citit, va ajunge la problemele propuse pe care le poate rezolva fie pe cont propriu, fie folosind suportul pus la dispoziție prin indiciile de rezolvare și soluțiile propuse de autorul lecției. La final, va marca sesiunea ca fiind parcursă, deblocând astfel noi captoare.

3.2 Rolul administrator

Rolul unui administrator este acela de a interacționa cu aplicația și a gestiona toate datele ei de la conturi și permisiuni la lectii prin intermediul paginii de admin.

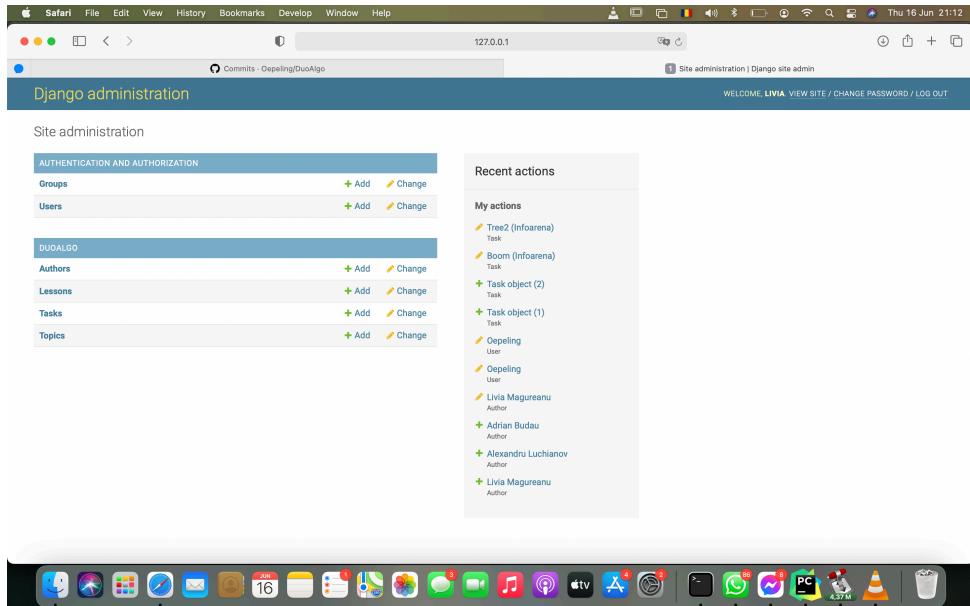


Figura 3.1: Pagina de admin

3.2.1 Useri și permisiuni

(a) Useri: Shows the 'Select user to change' page for the 'User' model. It lists users with their first name, last name, and email address. A search bar and filter buttons are present. The user 'Oepeling' (Liviu Magureanu) is selected.

(b) Permisiuni: Shows the 'Permissions' configuration page for the selected user 'Oepeling'. It allows assigning permissions to groups like 'Staff status' and 'Represents status'. It also shows the 'User permissions' section where specific permissions like 'Can add lesson' and 'Can change lesson' are assigned.

(a) Useri

(b) Permisiuni

3.2.2 Adăugarea lecțiilor

(a) Adăugarea unei lecții: Shows the 'Add lesson' form. Fields include Title, Author, Topic, Stage, and 'Things to learn before'. A note says 'Hold down "Control" or "Command" on a Mac, to select more than one.' Below the form are buttons for 'Save and add another', 'Save and continue editing', and 'Exit'.

(b) Editare detalii lecție: Shows the 'Edit lesson' form for a lesson titled 'Test2 (Infoarena) Boom (Infoarena)'. It includes fields for Title, Author, Topic, Stage, and 'Things to learn before'. Below the form is a 'Content' area with a rich text editor toolbar.

Figura 3.2: Adăugarea unei lecții

Capitolul 4

Detalii de implementare

4.1 Arhitectura

Arhitectura unui produs software este un factor esențial pentru crearea și întreținerea acestuia și reprezintă modul în care un sistem software este împărțit în diverse componente și definește relațiile dintre acestea.

Django își descrie propria arhitectură ca un software design pattern bazat pe MVC, însă cu câteva modificări. În continuare presupune împărțirea aplicației în trei componente logice interconectate: Model, View, Template.

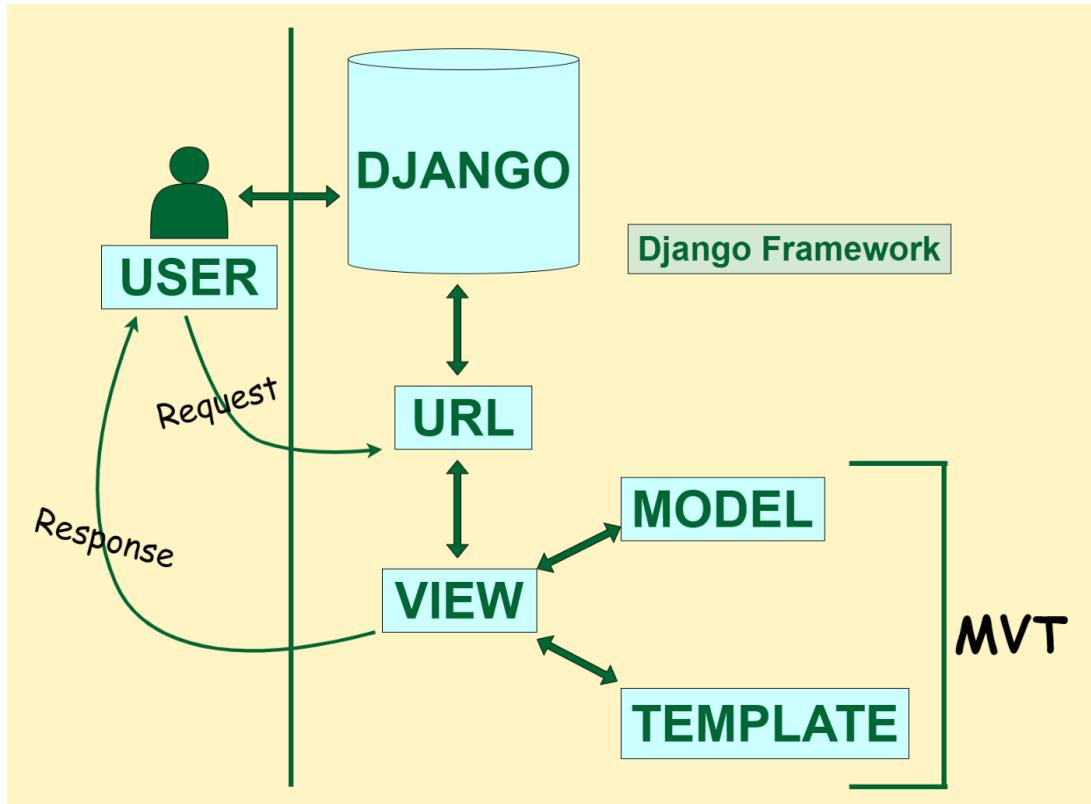


Figura 4.1: Arhitectura MVT în Django [3]

În varianta clasică, atunci când utilizatorul trimite un request, acesta ajunge în `urls` care procesează path-ul și datele din url (ex.: '/lesson/15/') și îl trimită către funcția sau clasa corespunzătoare din `views`. De acolo, request-ul este procesat, de obicei pentru a face query-uri sau update-uri în baza de date, prin intermediul componentei `models`. În final, `view` întoarce un `HttpResponse` (rendează o pagină) generat de componenta `template`.

Arhitectura acestei aplicații presupune separarea frontend-ului de backend. Aceasta decizie a fost motivată de trei factori:

1. separarea completă a procesului de dezvoltare
2. crearea unei interfețe grafice mai complexe, implementată complet în `javascript`
3. o arhitectură mai versată care permite adăugarea mai ușoară de elemente noi și, în viitor, a unei aplicații de mobil care depinde de același server

Astfel, în design-ul prezentat mai devreme apare un strat adițional care gestionează toată interacțiunea cu utilizatorul.

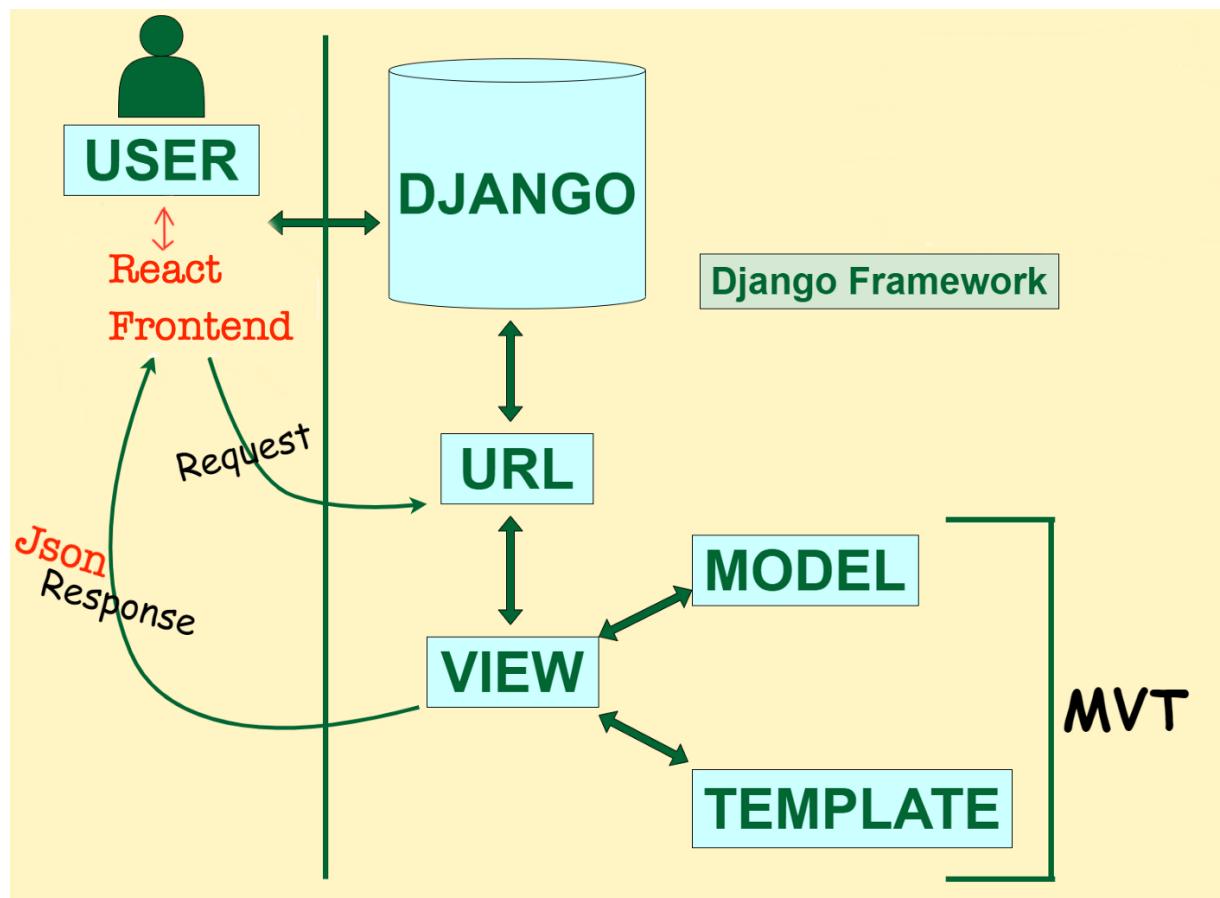


Figura 4.2: Arhitectura aplicației

4.2 Limbaje si framework-uri

Backend **Python 3.8.1**

Django 4.0.4 – <https://www.djangoproject.com/start/overview/>

Frontend **JavaScript**

React 18.2.0 – <https://reactjs.org>

4.3 Backend

4.3.1 Models (interacțiunea cu baza de date)

Prezentare generală

Documentație:

<https://docs.djangoproject.com/en/4.0/topics/db/models/>

<https://docs.djangoproject.com/en/4.0/ref/models/fields/>

Modelele reprezintă componenta care asigură comunicarea cu baza de date. Un model este o clasa în python care moștenește clasa de bază `models.Model` și o serie de caracteristici. Un element al unui model se numește `field` și, de obicei, corespunde în mod direct unei coloane din baza de date.

Field-urile se împart în următoarele categorii:

1. Simple

- (a) Numerice – `PositiveSmallIntegerField`, `BooleanField` etc.
- (b) Text – `CharField`, `TextField` etc.
- (c) Dată și oră – `DateTimeField`, `DurationField` etc.
- (d) Web specifice – `EmailField`, `JSONField`, `GenericIPAddressField` etc.
- (e) Optative

2. Relaționale

- (a) `ForeignKey`
Păstrează o relație de tipul **many-to-one**
- (b) `ManyToMany`
Emulează relația de tipul **many-to-many**. În spate, Django creează în baza de date un tabel intermediu (ex.: `author_books_9cdf`)
- (c) `OneToOne`
Impune relația de tipul **one-to-one** cu ajutorul unui câmp de tipul `ForeignKey` cu proprietatea adițională de unicitate

Modelele din aplicație

```
class Topic(models.Model):
    name = models.CharField("Topic name", max_length=100)
    supertopic = models.ForeignKey('self', on_delete=models.CASCADE, (...))
    # Field populat automat pentru sortarea topologica a aborelui
    topo_order = models.CharField(default="", (...))

class Task(models.Model):
    title = models.CharField("Task title", max_length=50)
    source = models.CharField("Task source", max_length=50)
    description = models.TextField("Task description (optional)", (...))
    link = models.URLField("Link to task", unique=True)
    solution = models.URLField("Link to solution (optional)", (...))

    tags = models.TextField("Tags (for now)", blank=True)

    hints = models.BooleanField("Does it have a hint system?", default=False)
    hint1 = models.TextField("First hint (optional)", (...))
    hint2 = models.TextField("Second hint (optional)", (...))
    hint3 = models.TextField("Third hint (optional)", (...))

class Stage(models.Model):
    name = models.CharField("Stage name", max_length=50)
    index = models.IntegerField("Stage index", unique=True)

class Author(models.Model):
    name = models.CharField("Author name", max_length=50)
    user = models.OneToOne(User, verbose_name="User account (if exists)", ...)

class Lesson(models.Model):
    topic = models.ForeignKey(Topic, verbose_name="Topic", (...))
    stage = models.ForeignKey(Stage, verbose_name="Stage", (...))

    title = models.CharField("Title", max_length=100)
```

```

author = models.ForeignKey(Author, verbose_name="Author")

duration = models.DurationField("Lesson duration (optional)", (...))
content = MDTextField("Lesson content in .md format", (...))
link_to_code = models.URLField("Link to sample of code (optional)", (...))

tasks = models.ManyToManyField(Task, verbose_name="List of related tasks")

dependencies = models.ManyToManyField('self', (...))

# User creat automat de sistemul de autentificare încorporat în Django
class UserExt(models.Model):
    user = models.OneToOneField(User, primary_key=True, ...)

    codeforces = models.CharField("Codeforces handle", (...))
    infoarena = models.CharField("Infoarena username", (...))
    varena = models.CharField("Varena username", (...))

    completed_lessons = models.ManyToManyField(Lesson, (...))
    current_lesson = models.ForeignKey(Lesson, ...)

```

4.3.2 Views

Documentație:

<https://docs.djangoproject.com/en/4.0/#the-view-layer>
<https://docs.djangoproject.com/en/4.0/topics/class-based-views/intro/>

Componența `view` are rolul de a încapsula toată logica responsabilă de procesarea și trimiterea unui răspuns pentru un request trimis de utilizator. În cazul de față, toate răspunsurile trimise sunt în format JSON și sunt procesate apoi de către frontend-ul în React, API-ul nefiind expus utilizatorului. (A nu se confunda cu pagina de admin)

Câteva exemple ar fi:

- Login
- Înregistrarea unui utilizator nou
- Lista lecțiilor (deja organizată pe stage-uri și nivele)
- Datele unei lecții (în raport cu utilizatorul logat)
- Marchează lecția (pentru utilizatorul logat)
- Alte query-uri adiționale mai specifice

4.3.3 Templates

Documentație:

<https://docs.djangoproject.com/en/4.0/#the-template-layer>

<https://docs.djangoproject.com/en/4.0/topics/templates/>

Nefolosite în aplicația aceasta.

4.4 Frontend

4.4.1 Comunicarea cu backend-ul

Comunicarea cu backend-ul se face cu ajutorul Axios (<https://axios-http.com>), un client HTTP care funcționează pe bază de promisiuni. O promisiune este un proxy (obiect menit să țină locul a ceva) pentru valoarea, nu neapărat cunoscută la momentul respectiv, care va fi completată atunci când operația se termină (cu sau fără succes). O promisiune poate fi într-o dintre următoarele trei stări:

pending	– starea inițială, nici completată, nici respinsă
fulfilled	– operația a fost efectuată cu succes ⇒ valoare reală
rejected	– operația a eșuat

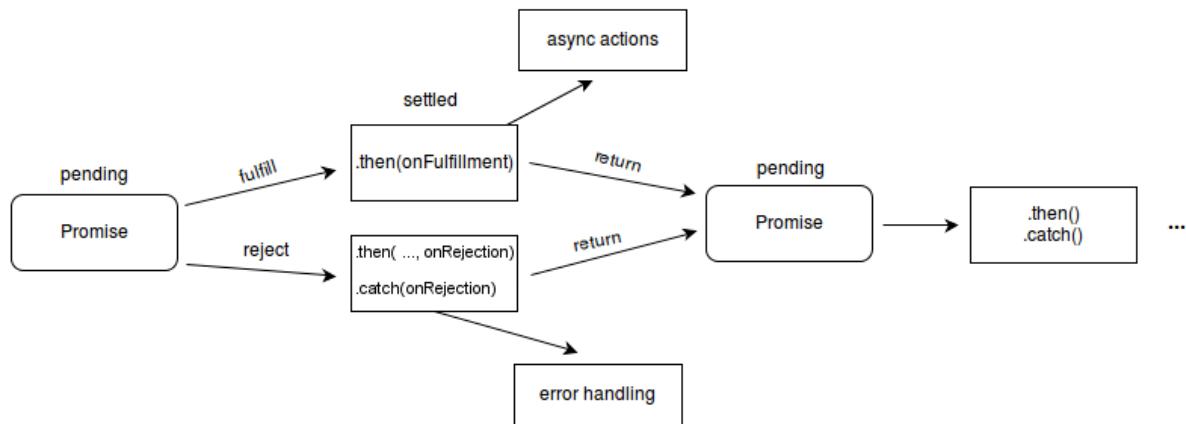


Figura 4.3: Schema de funcționare a unui sistem bazat pe promisiuni [1]

Acest sistem de funcționare permite ca încărcarea aplicației și rularea codului de JavaScript să se desfășoare asincron cu trimiterea cererilor către backend. În teorie, timpul de așteptare după un răspuns este la fel de mare, în practică, din faptul că cererile pot fi intuite și făcute în avans (înainte să fie nevoie de datele respective), partea cea mai costisitoare este mascată pentru utilizator și tot procesul pare mai rapid decât este în realitate. În plus, faptul că pagina începe să fie afișată înainte să fie completată creează iluzia unei tranziții naturale.

4.4.2 Cea mai costisitoare cerere

Cea mai costisitoare cerere către backend este cea pentru lista lecțiilor, care ar trebui să întoarcă lista completă și datele care trebuie afișate pentru fiecare pe pagina principală (stage, nivel, titlu, autor, muchiile sale în DAG, o scurtă descriere). În plus, frontend-ul nici nu folosește toată această informație deodată deoarece pe un ecran sunt afișate doar câteva lecții. Prin urmare, am decis să sparg această cerere în mai multe cereri mai mici.

La intrarea pe pagina principală, se trimit o cere către backend care să întoarcă progresul (punctul în care a ajuns) utilizatorului și datele lecțiilor din proximitatea celei curente. Aceste date servesc la încărcarea inițială a paginii. O cerere separată pentru datele imediat deasupra / dedesubtul celor deja încărcate, care au potențialul să intre în vizorul utilizatorului. Acest design ne permite să simulăm existența unei singure pagini prin care te poți plimba continuu în sus și în jos.

4.5 Development și procesul de deploy

4.5.1 VCS

Am folosit git ca sistem de gestionare al versiunilor în procesul de dezvoltare. Deși poate să nu pară atât de util într-un proiect de o persoană, este un mod foarte bun de a dezvolta în continuare, după ce aplicația este lansată, și de a controla ce versiune este live (pe server) și ce versiune este încă în lucru.

Repository-ul de pe GitHub: <https://github.com/Oepeling/DuoAlgo>

4.5.2 Cloud setup

Am ales să folosesc DigitalOcean (<https://www.digitalocean.com>) pentru gestionarea infrastructurii deoarece:

- ✓ poate hostui o baza de date PostgreSQL
- ✓ poate hostui serverul de Django și are un sistem integrat de a interacționa cu VCS-ul (în cazul de față GitHub)
- ✓ este user-friendly
- ✓ oferă o varietate mare de tutoriale
- + o serie de alte calități încă neexplorate

Capitolul 5

Concluzii

Deși design-ul poate fi mult îmbunătățit și va fi mult de muncă de acum înainte cu scrierea efectivă a conținutului lecțiilor, DuoAlgo este un proiect cu mult potențial care se adresează unei necesități din ce în ce mai presante în comunitatea de programare competitivă din România. De-a lungul anului am fost întrebată de mai mulți elevi dacă le pot trimite materiale de pe care să învețe singuri, fie înainte, fie pentru a recupera lipsuri din urmă. Sper ca aplicația această să reprezinte o soluție viabilă pentru alți copii aflați în situații similare.

O altă utilitate a acestei aplicații este că, din toamnă, va putea fi folosită în cadrul cursului facultativ de Programare Competitivă din facultatea noastră.

În ceea ce privește pașii de viitor, odată complet funcțională, testată (pentru că momentan nu există niciun fel de teste automate) și lansată aplicația, vreau să mă concentrez pe a găsi un model mai bun de a urmări și verifica progresul unui utilizator, eventual prin adăugarea de teste și concursuri. (Încă nu am o idee mulțumitoare, iar dacă aş fi avut, probabil ar fi apărut în această licență.)

Bibliografie

- [1] MDN contributors, *Promise*, URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (accesat în 27.5.2022).
- [2] Ben Straub Scott Chacon, „Pro Git (2 edition)”, în a 2-a ed., Apress, 2014, cap. 1.2 Getting Started - A Short History of Git.
- [3] Fahadul Shadhin, *The MVT Design Pattern of Django*, 2021, URL: <https://pythonplainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582> (accesat în 16.6.2022).