AlgoArena

1. Introducere

1.1 Context general

Programarea este un interes comun al ambilor membrii ai echipei noastre. De-a lungul timpului am utilizat diferite platforme dedicate învățării și testării abilităților când vine vorba de algoritmi și structuri de date.

Aceste platforme includ pbinfo.ro și leetcode.com, în cadrul căror există o gamă largă de probleme de rezolvat, iar utilizatorii își încarcă soluțiile (codul sursă în diferitele limbaje de programare suportate) spre a fi evaluate.

Un alt tip de platoformă este Advent of Code, care oferă probleme de rezolvat în fiecare zi a lunii decembrie, până în ziua de Crăciun. În cadrul acestei platforme, fiecare utilizator primește un input diferit, pentru care trebuie să încarce doar rezultatul final. Astfel, fiecare persoană poate rezolva problema folosid limbajul de programare preferat, sau chiar alte unelte, precum Excel sau SQL.

Noi am fost interesați de ideea de rulare și evaluare a codului, deci am decis să dezvoltăm o platformă similară cu prima categorie.

1.2 Obiective

Dezvoltarea unei platforme web care să ofere utilizatorilor posibilitatea de a rezolva probleme de algoritmizare și structuri de date, similară cu cele menționate la secțiunea 1.1. Mai mult decât atât, platforma încurajează utilizatorii să creeze propriile probleme pentru a fi rezolvate de alți utilizatori, în spiritul open-source.

Obiectivele principale ale acestei platforme sunt:

- oferirea unei interfețe ușor de utilizat pentru utilizator
- rezolvarea și evaluarea soluțiilor într-un mediu izolat și evaluarea acestuia, oferind suport pentru mai multe limbaje de programare decât platformele studiate
- oferirea unui sistem de autentificare și autorizare, pentru a proteja datele utilizatorilor
- oferirea unui sistem de persistență pentru probleme și soluții

1.3 Specificații

Utilizatorii au următoarele funcționalități la dispoziție:

- crearea unui cont/ autentificare
- crearea problemelor de rezolvat
- rezolvarea problemelor create de alți utilizatori (sau de ei înșiși) folosind diferite limbaje de programare
- vizualizarea soluțiilor proprii
- editarea profilului personal

Aceste funcționalități principale oferă o bază solidă pentru dezvoltarea ulterioară a platformei, precum adăugarea de funcții sociale, precum sistem de rating, mesagerie, etc.

2. Analiză, proiectare și implementare

2.1 Arhitectura aplicației

Arhitectura aplicației este împărțită în două componente principale, alături de câteva servicii adiacente. Componentele principale sunt frontend-ul (`algoarena-client`) și backend-ul `algoarena-api`, iar serviciile utilizate sunt baza de date și verificatorul soluțiilor.

Fiecare acțiune a utilizatorului are loc în frontend, care trimite cereri HTTP către backend (un API de tip REST). Acesta din urmă se ocupă de securizarea și procesarea cererilor, precum și de comunicarea cu baza de date și evaluatorul soluțiilor.

2.2 Tehnologii utilizate

Frontend-ul este dezvoltat folosind Vue.js, un framework JavaScript pentru construirea interfețelor web. Acesta este unul dintre cele mai populare framework-uri de acest tip, fiind ușor de învățat și de utilizat. De asemenea, Vue.js oferă suport pentru multe alte tehnologii, precum 'pinia' (pentru starea aplicației), 'vue-router' (pentru navigare), si multe alte biblioteci pentru a face interfața cât mai interactivâ și placută. Dependențele sunt gestionate folosind npm, iar cele mai importante sunt:

- `pinia`: o bibliotecă de management a stării aplicatiei, care oferă suport pentru multe alte funcționalități, precum reactivitate, persistență, etc.
- `vue-router`: oferă suport pentru navigare în aplicație, precum rute, parametri, query-uri, etc.
- `tailwind-css`: un framework CSS care oferă un set de clase pentru a stiliza interfețele web intr-un mod rapid și eficient
- `shadcn-vue`: un set de componente pentru Vue.js care vin cu un design modern și plăcut
- `codemirror`: un editor de cod open-source care oferă suport pentru multe limbaje de programare, precum și pentru multe alte funcționalități, precum evidențierea sintaxei, completarea automată, etc.

Vue.js este un framework bazat pe componentele, care oferă posibilitatea de a împărți interfața în componente reutilizabile, avem uneori nevoie de aceseași funcționalități în mai multe locuri. Putem sa facem acest lucru prin crearea de componente ce primesc date și emit evenimente. Insa odata cu cresterea complexitatii aplicatiei, aceasta abordare poate deveni greu de mentinut. Pentru a rezolva aceasta problema, am folosit Pinia, o biblioteca de management a starii aplicatiei. In acest context, o stare a aplicatiei este orice informatie care trebuie sa fie partajata intre mai multe componente. Am ales sa folosim Pinia pentru ca ofera un mod simplu si elegant de a gestiona autea aplicatiei, astfel statusul aplicatiei este usor de mentinut si de modificat. Deoarece avem o autentificare bazate pe tokeni JWT, am folosit Pinia pentru a salva acest token si in LocalStorage (pentru a pastra autentificarea si dupa inchiderea aplicatiei). Acest token este anexat requesturilor catre API pentru a verifica daca utilizatorul este autentificat.

Pentru ca Vue.js este destinat in special pentru aplicatiile de tip SPA (Singe Page Aplication), pentru a gestiona mai multe rute, am folosit Vue Router. Acesta oferă posibilitatea de a defini rute, precum și de a gestiona navigarea între acestea. In contextul aplicatiei, rute au fost definite in folder-ul 'views', unde fiecare componenta reprezinta o pagina. De asemenea, pentru a ne asigura ca accesul la anumite rute este permis doar utilizatorilor autentificati, am folosit un middleware care la fiecare navigare verifica daca utilizatorul este autentificat. Folosind tokenul JWT stocat anterior in LocalStorage, am putut verifica daca utilizatorul este autentificat sau nu printr-un request catre API. In cazul in care utilizatorul este autentificat, acesta este redirectionat catre ruta dorita, in caz contrar, este redirectionat catre pagina de autentificare.

Tailwind CSS este un framework CSS care oferă un set de clase pentru a stiliza interfețele web într-un mod rapid și eficient. Spre deosebire de alte framework-uri CSS, precum Bootstrap sau Materialize, Tailwind CSS oferă un set de clase pentru a stiliza interfețele web, în loc de componente predefinite. Acest lucru oferă un control mult mai mare asupra interfeței, precum și un mod mai rapid de a stiliza aceasta, intrucat nu mai avem nevoie de a crea si scrie clase CSS personalizate pentru fiecare element in parte. De asemenea, Tailwind CSS oferă suport pentru multe alte funcționalități, precum responsivitate, teme, etc. Spre exemplu, daca dorim sa adaugam o margine de 4rem unui element in stanga si in dreapta acestuia, putem folosi clasa 'mx-4'. Acest lucru este mult mai rapid si mai eficient decat a scrie o clasa CSS personalizata pentru acest lucru. Aceste clase au si beneficiul de a fi usor de inteles si de mentinut, intrucat sunt denumite in mod clar si concis, iar atributele acestora nu se suprapun decat in cazuri rare.

Shadcn Vue este un set de componente pentru Vue.js care vin cu un design modern și plăcut. Beneficiul acestei librarii de componente este faptul ca aste componente pot fi stilizate folosind Tailwind CSS pentru a retusa anumite aspecte, precum culorile, marginile, etc. Aceste componente sunt folosite pentru a crea o interfață cât mai interactivă și plăcută, precum și pentru a oferi o experiență de utilizare cât mai buna in orice context din punct de vedere al dispozitivului utilizat. Codemirror este un editor de cod open-source pentru aplicatii web care oferă suport pentru multe limbaje de programare, precum și pentru multe alte funcționalități, precum evidențierea sintaxei, completarea automată, etc. Acesta este folosit pentru a oferi utilizatorilor posibilitatea de a rezolva problemele folosind diferite limbaje de programare, precum și pentru a oferi o experiență de utilizare cât mai buna. Supportul pentru aceste limbaje de programare sunt oferite prin intermediul unor extensii, astfel putem alege care sunt limbajele, temele si alte functionalitati pe care dorim sa le folosim.

Pentru a face legatura intre frontend si backend, am folosit functia `fetch` din JavaScript. Aceasta functie ofera posibilitatea de a face requesturi HTTP catre API-ul nostru, precum si de a trimite date catre acesta. Putem trimite diferite tipuri de requesturi, precum GET, POST, PUT, DELETE, etc. De asemenea, putem trimite diferite tipuri de date, precum JSON, text, etc. Aceasta functie ofera un mod simplu si eficient de a comunica cu API-ul nostru, fara a fi nevoie de alte biblioteci sau framework-uri.

Backend-ul este dezvoltat folosind Spring Boot, un framework Java pentru dezvoltarea de aplicații web. Acesta este unul dintre cele mai populare framework-uri de acest tip, fiind ușor de învățat și de utilizat. De asemenea, Spring Boot oferă suport pentru multe alte tehnologii, precum Hibernate, care este folosit pentru comunicarea cu baza de date. Depenențele sunt gestionate folosind Maven și includ:

- Spring Boot DevTools: oferă posibilitatea de restartări rapide a aplicației, LiveReload (reîncărcare automată a aplicație după orice schimbare a codului sursă) și alte configurări pentru îmbunătățirea experienței de dezvoltare.
- Lombok: este o bibliotecă de adnotări Java care elimină necesitatea de a scrie cod boilerplate (cod care trebuie scris de fiecare dată, dar nu aduce nicio valoare adăugată), precum: getteri, setteri, constructori, etc.
- Spring Web: oferă suport pentru dezvoltarea de aplicații web, precum și pentru crearea de API-uri de tip REST folosind Spring MVC. Apache Tomcat este folosit ca server web încorporat.
- Spring Security: oferă suport pentru securizarea aplicațiilor web, precum autentificare, autorizare, criptare, etc.
- JDBC API: oferă suport pentru comunicarea cu baze de date relaționale folosind JDBC (Java Database Connectivity). Definește un set de interfețe Java pentru comunicarea cu baze de date relaționale.
- Spring Data JPA: oferă persistență de tip JPA (Java Persistence API) pentru aplicațiile Spring. Acesta oferă suport pentru operații CRUD (Create, Read, Update, Delete) și multe altele. Sunt folosite anotări

pentru a defini entitățile și relațiile dintre acestea. Hibernate este folosit ca implementare a JPA, pentru a face mapping-ul între obiectele Java și tabelele din baza de date.
- MySQL Driver: driver-ul specific pentru MySQL.
- Validation: oferă suport pentru validarea datelor de intrare. Acesta oferă suport pentru validarea datelor folosind anotări, precum `@NotNull`, `@Size`, `@Email`, etc.
- Spring Boot Actuator: oferă suport pentru monitorizarea și managementul aplicațiilor Spring Boot. Acesta oferă informații despre starea aplicației, precum: starea serverului, starea bazei de date, starea cache-ului, etc.

Baza de date folosită este MySQL, un sistem de gestiune a bazelor de date relaționale. Acesta este unul dintre cele mai populare sisteme de gestiune a bazelor de date. De asemenea, MySQL oferă suport pentru multe alte tehnologii, precum Hibernate, care este folosit pentru comunicarea cu baza de date.

Evaluatorul soluțiilor este un serviciu separat, numit Judge0, care oferă suport pentru mai multe limbaje de programare și oferă un API de tip REST pentru evaluarea codului sursă. Acesta oferă posiblitatea de self-hosting (găzduire proprie), dar și un serviciu cloud (găzduire de către dezvoltatorii serviciului).

Ambele aceste servicii sunt găzduite local folosind Docker, un sistem de containereizare care oferă un mediu izolat pentru rularea aplicațiilor. Acesta oferă posibilitatea de a rula aplicații într-un mediu izolat, fără a afecta sistemul gazdă. De asemenea, Docker este extrem de util pentru deployment, deoarece aplicațiile nu depind de sistemul gazdă.

2.3 Implementare

Inițial, mediul de dezvoltare ales pentru create API-ului a fost IntelliJ IDEA, unul dintre cele mai populare IDE-uri pentru Java. Acesta oferă suport pentru multe alte tehnologii, precum Spring Boot, Maven, Hibernate, etc. Pentru a începe dezvoltarea, am creat un proiect Spring Boot folosind Spring Initializr. Acesta oferă posibilitatea de a genera un proiect Spring Boot cu toate dependențele necesare, precum Maven, Hibernate, Spring Web, etc. De asemenea, IntelliJ IDEA oferă posibilitatea de conectare la baza de date, pentru a vizualiza și modifica datele din baza de date. Astfel, nu a fost nevoie de instalarea unui alt client pentru baze de date.

Totuși, datorită interfeței inutil de complexe și a integrării inferioare cu git, dar și a lipsei suportului pentru GitHub Copilot Chat, am decis să folosim Visual Studio Code, un editor de cod open-source dezvoltat de Microsoft. Folosind extensii, acesta oferă suport pentru practic orice tehnologie și limbaj de programare. De asemenea, oferă o integrare mult mai bună cu git, precum și suport pentru GitHub Copilot Chat. Pentru dezvoltarea Java a fost folosit Extension Pack for Java, pachet dezvoltat de Microsoft.

Pentru testarea diferitelor funcționalități ale API-ului, unul dintre noi a folosit direct comanda `curl`, iar apoi Visual Studio Code alături de extensia REST Client, pentru a putea scrie și rula cereri HTTP direct din editorul de cod. Între timp, celălalt membru al echipei a `HttPie`, un client HTTP desktop cu o interfata ceva mai moderna decat `Postman`. Acesta oferă posibilitatea de a scrie și rula cereri HTTP, precum și de a vizualiza și modifica răspunsurile. De asemenea, oferă suport pentru multe alte funcționalități, precum autentificare, autorizare, salvare de cereri, etc.

Din punct de vedere al codului sursă, am încercat să urmăm cele mai bune practici de programare. Proiectul Java pentru API a fost împărțit în următoarele pachete:

- auth: conține clasele care se ocupă de autentificare și autorizare. Acestea sunt: `JwtTokenFilter` pentru filtrarea cererilor HTTP in funcție de tokenul JWT, `JwtService` pentru generarea și validarea tokenului

JWT, `SecurityConfig` pentru configurarea securității, precum și `AuthService` pentru a face operații legate de utilizatori si autentificare.

- controllers: conține clasele care se ocupă de procesarea cererilor HTTP. Acestea sunt: `AuthController` pentru tot ce ține de autentificare, autorizare și securizarea conexiunii, `CategoryController`, `ProblemController`, `SubmissionController` și `UserController` pentru procesarea cererilor legate de categoriile de probleme, problemele în sine, soluțiile trimise de utilizatori și respectiv utilizatori. Pentru fiecare tip de acțiune s-a folosit metoda HTTP corespunzătoare, precum GET, POST, PUT, DELETE, etc.

- dto: conține clasele care reprezintă obiectele de transfer de date. Termenul DTO este un acronim penmtru Data Transfer Object, adică o clasă (mai exac un `record`) ce nu coține logică, doar date. Acestea sunt încadrate în alte 5 pachete, pentru categorii, probleme, soluții, utilizatori și pentru interacțiunea cu Judge0. În fiecare dintre primele 4 pachete se găsesc un DTO de bază, care este accesibil doar la nivelul pachetului și este un `abstract record`. Apoi, DTO-urile folosite în cererea de creare a entităților și în răspunsurile la cereri extind acest `record` de bază. DTO-ul folosit în cererea de actualizare extinde mereu DTO-ul de creare, pentru a păstra consistența datelor, dar adaugă un câmp pentru identificatorul entității care urmează să fie actualizată (id).

- mappers: conține clasele care se ocupă de maparea între obiectele de transfer de date și entitățile JPA. Mai exact, fiecare mapper (pentru categorie, problemă, soluție și utilizator) conține o metodă de conversie între DTO-ul de creare și entitatea JPA, precum și între această entitate și DTO-ul de răspuns. Acestea sunt folosite în clasele din pachetul `controllers` pentru a converti datele primite de la client în obiecte Java și invers.

- models: conține clasele care reprezintă entitățile JPA. Aici sunt definite clasele pentru categorii, probleme, soluții și utilizatori, dar și tipurile `enum` folosite, pentru dificultatea fiecărei probleme și rolul utilizatorului. De asemenea, prin adnotări JPA sunt definite relațiile dintre aceste entități (ex: `OneToMany`), precum și coloanele din tabelele din baza de date.

- repositories: conține interfețele care extind JpaRepository pentru a face operații CRUD. Acestea sunt: `CategoryRepository`, `ProblemRepository`, `SubmissionRepository` și `UserRepository`. Acestea sunt folosite în clasele din pachetul `controllers` pentru a realiza operațiile de citire, scriere, actualizare și ștergere cerute de client.

2.4 Testare

Pentru testarea API-ului, am facut requesturi HTTP folosind `curl` și `HttPie`. Am testat raspunsurile primite atat cu date si metode HTTP valide, cat si invalide, pentru a verifica daca API-ul raspunde corect si in cazul unor erori. De asemenea, am testat si securitatea API-ului, pentru a verifica daca aceasta functioneaza corect.

Pentru a gestiona datele din baza de date, am folosit `Dbeaver`, un client pentru baze de date relaționale. Acesta oferă posibilitatea de a vizualiza și modifica datele din baza de date, precum și de a face operații de tip CRUD (Create, Read, Update, Delete). De asemenea, oferă suport pentru multe alte funcționalități, precum exportul și importul de date, vizualizarea și modificarea structurii bazei de date, etc.

Pentru testarea frontend-ului, am folosit `Devtools` din browserul ales (Edge/Firefox), pentru a vizualiza si analiza structura paginii web, precum si pentru a verifica daca aceasta este responsive. Putem sa analizam si

requesturile trimise cate API, precum si raspunsurile primite, pentru a verifica daca acestea sunt corecte. De asemenea, putem sa analizam si starea aplicatiei, precum si eventualele erori care apar.

2.5 Instalare și utilizare

Condițiile prealabile pentru a rula aplicația sunt:

- Java 17
- Docker și Docker Compose
- Node 20

Procurarea aplicației se poate realiza fie folosind arhiva prezentă, fie clonându-le de pe GitHub. Comenzile pentru cele două părți sunt:

```
https://github.com/CatalinIuga/algoarena-client
https://github.com/24online24/algoarena-api
```

Primul pas este creare containerului pentru baza de date. Acest proces este realizat ușor folosind fișierul `docker-compose.yml` din directorul api. Comanda este:

```
docker-compose up -d
```

Parola pentru accesarea MySQL trebuie salvată într-un fișier numit `.env`, pentru care a fost oferit un exemplu (`.env-template`).

Baza de date goală cu numele dorit trebuie creată manual, folosind orice client pentru baze de date relaționale, precum DBeaver, MySQL Workbench, etc. IntelliJ IDEA oferă suport pentru baze de date, precum și pentru crearea și modificarea acestora. De asemenea, imaginea Docker folosită pentru baza de date poate fi folosită și drept client.

În fișierul `src/main/resources/application.properties` pentru care, de asemenea, a fost oferit un exemplu trebuie incluse datele de conectare la baza de date și configurare a acesteia.

Pentru a rula API-ul, se folosește comanda:

```
./mvnw spring-boot:run
```

Pentru ca aceasta să funcționeze, trebuie să fie setată variabila de mediu `JAVA_HOME`. De asemenea, poate fi folosi un IDE (precum Apache Netbeans, IntelliJ IDEA, Visual Studio Code, etc.) pentru a rula aplicația. Acesta va fi disponibilă la adresa `http://localhost:8080`.

Pentru a rula frontend-ul, se folosește comanda:

```
npm install
```

pentru a instala toate dependențele, și apoi

```
npm run dev
```

pentru a rula aplicația. Aceasta va fi disponibilă la adresa `http://localhost:5173`.

În ceea ce privește Judge 0, acesta poate fi găzduit local folosind Docker, dar și folosind serviciul cloud oferit de dezvoltatorii acestuia. Pentru a găzdui local, incepem prin a descărca imaginea Docker folosind comanda:

```
wget https://github.com/judge0/judge0/releases/download/v1.13.0/judge0-v1.13.0.zip
unzip judge0-v1.13.0.zip
```

Apoi, putem rula imaginea folosind comanda:

```
cd judge0-v1.13.0
docker-compose up -d db redis
sleep 10s # wait for db and redis to start
docker-compose up -d
sleep 5s # wait for judge0 to start
```

Dupa ce aceste comenzi sunt rulate, putem accesa API-ul la adresa `http://localhost:2358`. Documentația API-ului este disponibilă la adresa `https://ce.judge0.com/`.

3. Concluzii

3.1 Rezultate

Am reușit să îndeplinim toate obiectivele propuse, precum și o experiență fluidă pentru utilizatori. Astfel aceștie pot să își creeze cont, de pe care pot ieși și reautentifica oricând, să creeze probleme de rezolvat, să rezolve problemele create de alți utilizatori, să își editeze problemele create, să își vadă soluțiile, să își editeze profilul personal, etc.

3.2 Direcții de dezvoltare viitoare

Câteva funcții pe care am dorit să le implementăm, dar timpul limitat nu ne-a permis, sunt:

- multiple teste (test cases) pentru fiecare problemă, atât exemple, cât și teste ascunse (pentru evaluare)
- posibilitatea de a crea seturi de probleme. Acest lucru ar fi util pentru a crea concursuri sau teme de casă. De asemenea, ar fi util pentru a crea probleme care depind una de cealaltă, precum: problema 2 depinde de rezolvarea problemei 1, problema 3 depinde de rezolvarea problemei 2, etc. Mai mult decât atât, seturile de probleme ar putea fi folosite și pentru interviuri tehnice din partea angajatorilor, precum în cadrul platformei Kattis.
- vizualizarea profilului altor utilizatori, precum și a soluțiilor acestora. Acest lucru ar fi util pentru a învăța de la alții, precum și pentru a vedea cum au rezolvat alții aceeași problemă. Această funcție ar putea fi

însoțită de un sistem de rating, precum cel de pe Codeforces, care ar putea fi folosit pentru a evalua calitatea soluțiilor. De asemenea, un sistem de mesagerie ar aduce o componentă socială în plus.
- un panou de administrare, pentru a gestiona utilizatorii, problemele, soluțiile, etc. Acest lucru ar fi util pentru a șterge problemele care nu sunt conforme cu regulile platformei, precum și pentru a șterge utilizatorii care nu respectă regulile platformei.