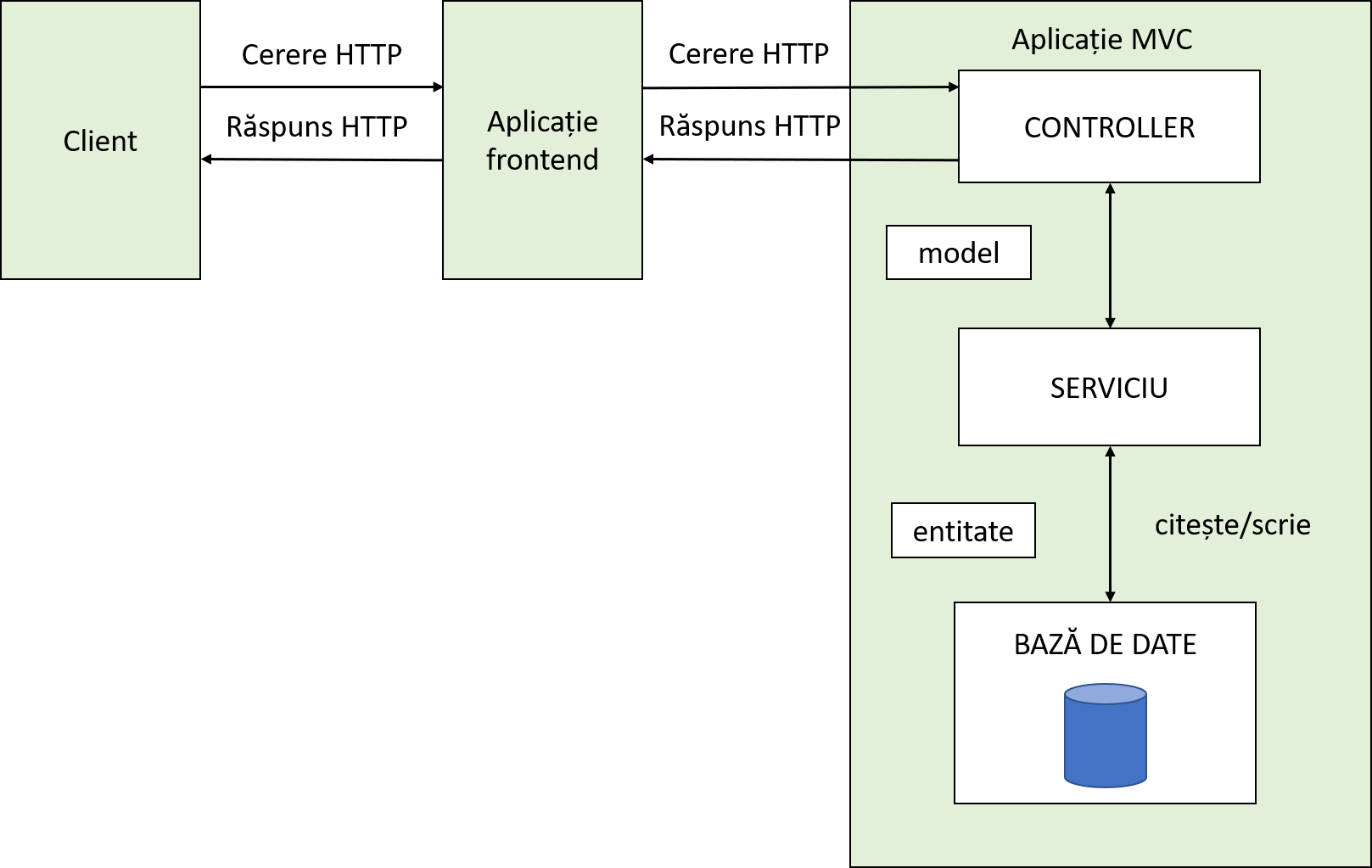
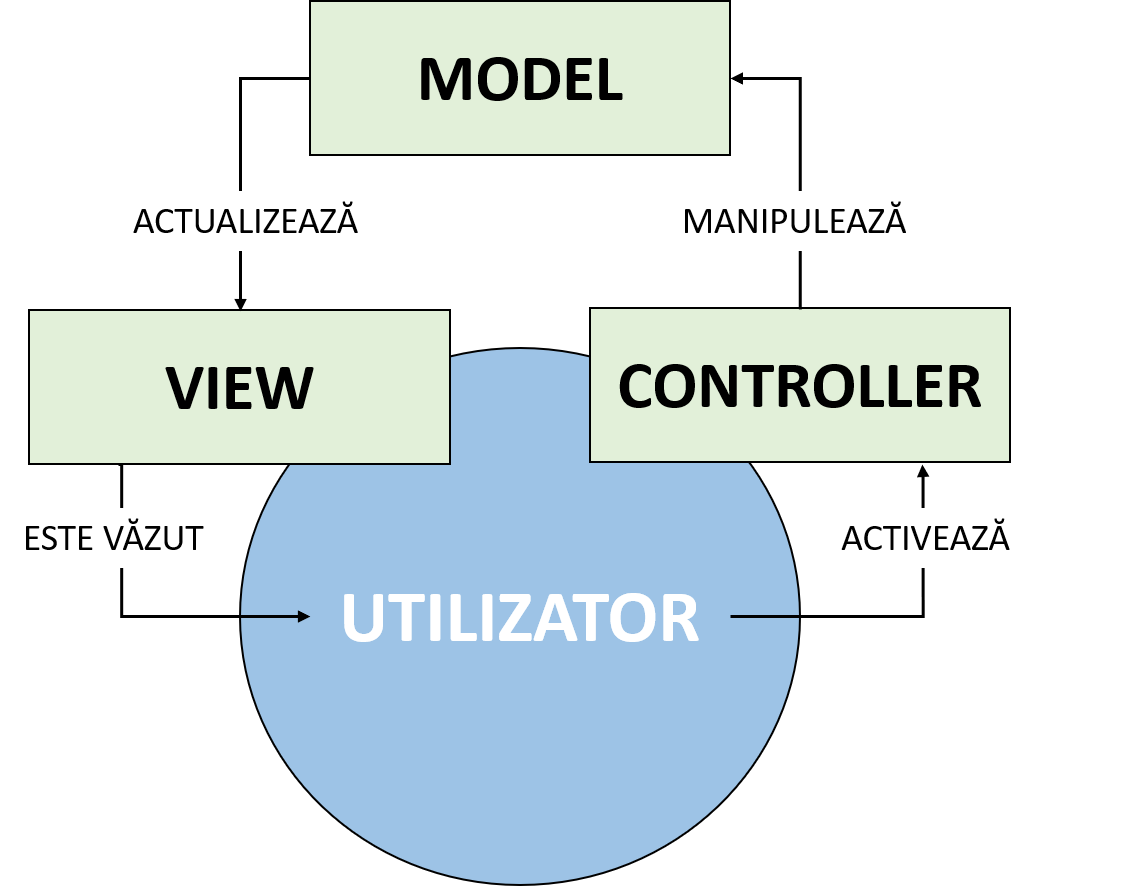
Arhitectura sistemului

Infrastructura proiectului este una modulară, fiecare parte având roluri bine definite, urmărind modelul propus de arhitectura client-server. Utilizatorul solicită printr-o cerere HTTP o resursă de la server-ul de frontend, care în funcție de necesități, trimite mai departe cereri HTTP către server-ul de backend. Server-ul de backend efectuează procesările complexe și interacționează cu baza de date, iar apoi acesta returnează un răspuns server-ului de frontend, care actualizează componentele interfeței grafice în mod corespunzător și le livrează clientului într-un mod eficient.



[Arhitectura platformei educative de testare]

În cadrul proiectului este folosit modelul arhitectural MVC. Rolul acestuia este de a separa logica aplicației de interfața grafică. Astfel, aplicația de backend va fi constituită din două părți principale: model și controller, iar aplicația de frontend va constitui partea de view din MVC. Componenta controller va fi responsabilă cu primirea cererilor și trimiterea răspunsurilor, iar componenta view va descrie interfața grafică. Modelele vor fi folosite pentru a găzdui informația transmisă între celelalte două componente.



[Fluxul acțiunilor în modelul arhitectural MVC]

Tehnologii și API-uri folosite

Pentru implementarea părții de frontend a fost folosită biblioteca Javascript React. Aceasta furnizează un mod facil de creare a unei aplicații de tip SPA (o aplicatie cu o singura pagină) ce interacționează cu browserul web prin rescrierea dinamică, a DOM-ului curent, cu date noi generate fie din client, fie din server. În acest caz, browserul nu este nevoit să încarcă o pagină nouă[[1]](#footnote-1).

Pentru implementarea părții de backend a fost folosit framework-ul ASP.NET Core ce utilizează o implementare a unui server HTTP pentru a gestiona cererile. Serverul furnizează cererile aplicației dezvoltate sub forma unui context HTTP (HttpContext)[[2]](#footnote-2).

O aplicație dezvoltată în React este ideal de integrat cu un backend de tip MVC Web API unde nu există o implementare pentru componenta View[[3]](#footnote-3), iar React poate compensa prin gestionarea părții de interacțiune cu utilizatorul. ASP.NET MVC oferă o modalitate foarte bună de separare a responsabilităților diferitor componente ale aplicației. Această separare este convenabilă pentru dezvoltatori deoarece nu există un mix între codul HTML și logica aplicației[[4]](#footnote-4).

Pentru asigurarea persistenței datelor a fost folosită o bază de date relațională MSSQL, alături de DBMS-ul SQL Server.

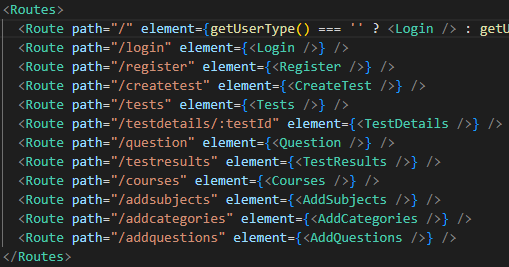
În cadrul aplicației de frontend au mai fost folosite Axios, Bootstrap, Formik, React Router Dom. React Router DOM este un API creat pentru React cu scopul de a ,,simula’’ un browser history. Deoarece aplicația este de tip SPA, browser-ul nu navigheaza prin fișiere html. Axios este o bibliotecă JavaScript folosită pentru a face cereri HTTP din browser. Formik este o bibliotecă Javascript folosită pentru gestionarea facilă a formularelor ce necesită o validare complexă. Bootstrap este un framework CSS utilizat pentru dezvoltarea de aplicații web responsive.

În cadrul aplicației de backend au mai fost folosite BCrypt.Net și Entity Framework Core. BCrypt.Net este o bibliotecă utilizată pentru aplicarea unor funcții de hash asupra parolelor utilizatorilor. Entity Framework Core este un ORM ce oferă dezvoltatorilor un mecanism automat pentru memorarea și accesarea datelor dintr-o bază de date.

De asemenea, au fost furnizate ca dependințe utilitarul clang-format pentru formatarea codului generat și compilatorul C++, g++, folosit prin intermediul mediului de rulare furnizat de Mingw-w64 care asigură funcționalitatea acestuia pe sistemul Windows.

Implementarea părții de frontend

Comanda create-react-app[[5]](#footnote-5) creează fișierele și instalează modulele necesare pentru a crea o aplicație minimală cu configurările standard necesare, scurtând astfel timpul de dezvoltare. Browser-ul primește de la server-ul de frontend fișierul index.html ce conține un div cu id-ul ,,root’’. Întreaga aplicație va fi randată în interiorul acestui div prin intermediul unei funcții din fișierul index.js asociat. Componenta App conține mai multe rute, care accesate, vor declanșa randarea componentelor aferente[[6]](#footnote-6), a căror interfață poate fi vizualizată în Anexe.



[Componentele create pentru interfața platformei educative]

Un avantaj al bibliotecii React este modul de gestionare al navigării între pagini. Prin utilizarea acestei librării este prevenită reîncărcarea totală a unei pagini, lucru care nu este necesar datorită faptului că antetul și finalul componentelor, librăriile JS și CSS nu s-au schimbat. Prin urmare, nu este necesar ca acestea să fie din nou trimise la nivelul clientului.

Rolul principal al unei componente este de a returna codul HTML asociat unei pagini. Celelalte elemente prezente în cadrul unei componente fie gestionează starea acesteia fie transformă anumite date în cod HTML[[7]](#footnote-7).

Un Hook este o funcție specială care permite accesul la niște funcționalități suplimentare oferite de React. Pentru a gestiona starea din interiorul unei componente, a fost folosit hook-ul useState, care returnează două elemente: un obiect căruia îi va fi atribuită o valoare și o funcție de setare a valorii obiectului respectiv. Setarea unui obiect de stare declanșează o rerandare a paginii. De exemplu, tiparul unei întrebări este afișat până în momentul primirii răspunsului de la server, pentru ca ulterior pagina să fie actualizată în mod corespunzător cu întrebarea generată. O altă funcție de tip React Hooks denumită useEffect a fost folosită pentru a executa un ,,efect’’ atât la prima randare a unei componente cât și de fiecare dată când este rerandată. De exemplu, pentru ca un administrator să poată vizualiza categoriile de întrebări create este necesară obținerea datelor de la server. Va fi efectuat un apel către API la prima randare a paginii pentru a fi populată cu intrările corespunzătoare, iar în momentul în care este adaugată o nouă categorie, va fi declanșată o rerandare și prin urmare, un nou apel către API cu scopul de a actualiza pagina.

Implementarea părții de backend

Crearea API-ului

Controllerele reprezintă punctul de intrare în partea de server din aplicația client. Acestea definesc rutele (endpoint-urile) pentru API-ul web și gestionează cererile HTTP asociate, orchestrând toate serviciile din backend. Metodele asociate rutelor sunt securizate cu un atribut de autorizare [Authorized]. Această abordare a fost utilizată pentru asigurarea implicită a metodelor și rutelor definite pe parcursul proiectului. Orice metodă nou adaugată este sigură, dacă nu este facută publică în mod explicit prin atributul [AllowAnonymous]. O rută este un tipar URL care este mapat unei acțiuni dintr-un controller MVC.

[Elemente descriptive ale API-ului platformei de testare]

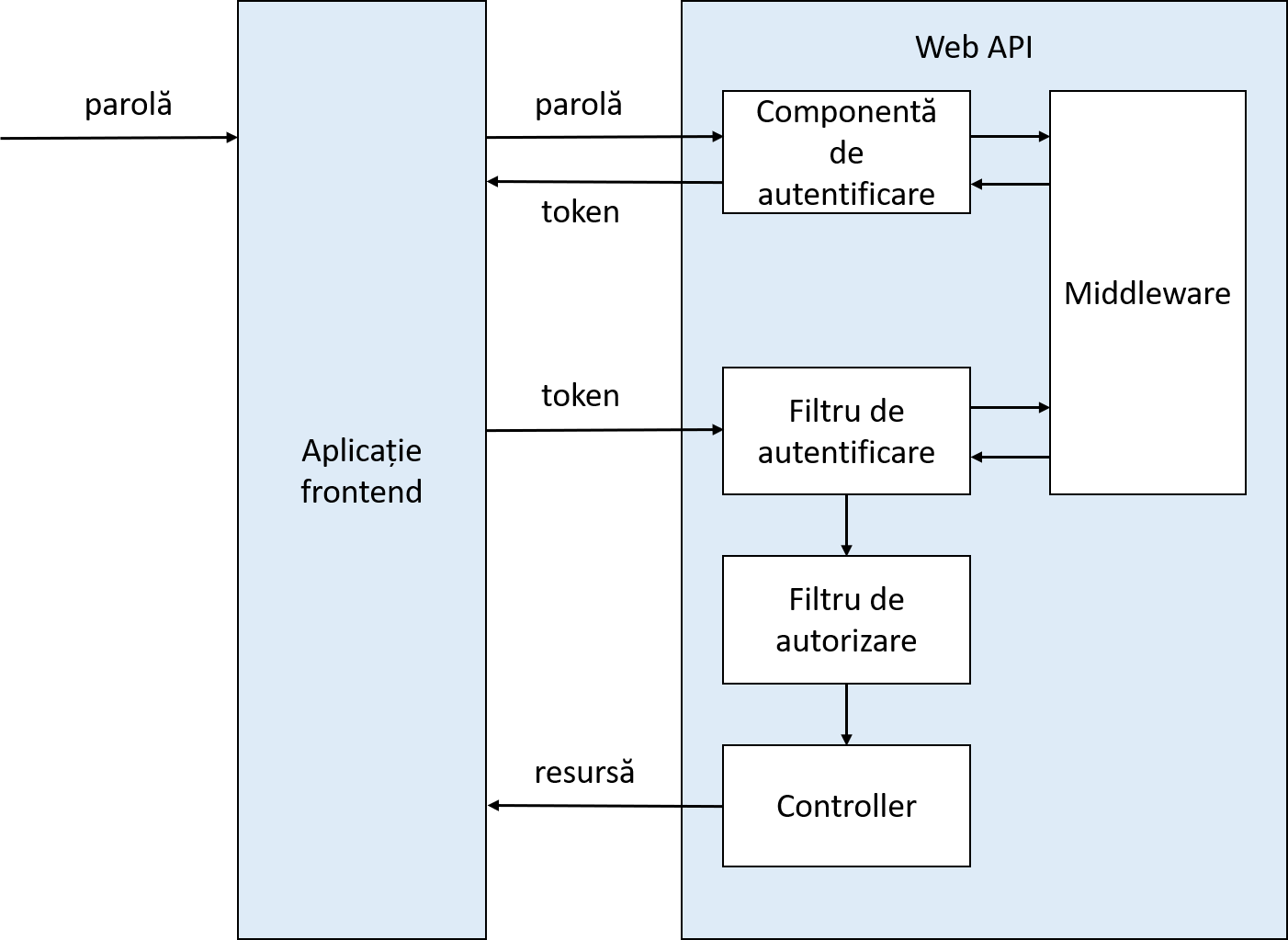
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Utilizatori autorizați | API | | Descriere | Model/Parametrii cerere | Model răspuns |
| administrator | POST | /subjects | Adaugă un subiect | AddSubjectRequest | { message = "The subject has been added" } |
| administrator, profesor | GET | /subjects | Obține subiectele disponibile | - | List<GetSubjectResponse> |
| administrator | POST | /categories | Adaugă o categorie | AddCategoryRequest | { message = "The category has been added" } |
| administrator, profesor | GET | /categories/{subjectId} | Obține categoriile disponibile | subjectId | List<GetCategoryResponse> |
| administrator | POST | /questions | Adaugă o întrebare | AddQuestionRequest | { message = "The question has been added" } |
| administrator, profesor | GET | /questions?subjectId=X&categoryId=X | Obține întrebările asociate unui subiect și unei categorii | subjectId, categoryId | List<GetQuestionResponse> |
| student | GET | /questions/generate?testId=X&username=X&questionTypeId=X | Generează o întrebare de un anumit tip pentru un student care susține un test | testId, username, questionTypeId | GenerateQuestionResponse |
| profesor | POST | /courses | Adaugă un curs | AddCourseRequest | { message = "The course has been created" } |
| student | POST | /courses/enrollment | Înscrie un student într-un curs | AddEnrolledStudentRequest | { message = "The student has been enrolled" } |
| profesor, student | GET | /courses/{username} | Obține cursurile în care un student este înscris sau pe care un profesor le-a creat | username | List<GetCourseResponse>,  List<GetStdCourseResponse> |
| profesor,  student | GET | /courses | Obține toate cursurile disponibile | - | List<GetStdCourseResponse> |
| profesor | POST | /tests | Adaugă un test | AddTestRequest | { message = "The test has been created" } |
| profesor, student | GET | /tests/{username} | Obține testele create de un profesor sau testele asociate cursurilor în care un student este înscris | username | List<GetTestResponse> |
| profesor, student | GET | /tests/test/{testId} | Obține detaliile referitoare la un anumit test precum denumire, dată, oră și rezultate sumare și statistici pentru profesori | testId | GetTestDetailsResponse, GetTestSummaryResponse |
| profesor, student | GET | /results?username=X&testId=X | Obține rezultatele pentru un student care a participat la un anumit test | Username, testid | GetResultResponse |
| student | POST | /results | Adaugă o intrare în tabela de rezultate la începerea unui test de către un student | AddResultReqest | AddResultResponse |
| student | PATCH | /results/details/{resultDetailsId} | Actualizează întrebarea generată cu răspunsul furnizat de student | resultDetailsid, JSONPatchDocument<ResultDetail> | { message = "The result details have been updated" } |
| student | PATCH | /results/{resultId} | Actualizează rezultatul final al studentului cu nota și timpul în care a terminat testul | resultId, JSONPatchDocument<Result> | { message = "The result has been updated" } |
| student | GET | /results/answeredTests/{username} | Obține testele la care a participat un student | username | GetAnsweredTestsResponse |
| - | POST | users/register | Întregistrează un profesor sau un student | RegisterRequest | { message = "Registration successful" } |
| - | POST | users/login | Autentifică un profesor, student sau administrator | LoginRequest | LoginResponse |

Implementarea unei autentificări JWT

Pentru a avea acces la funcționalitățile aplicației, utilizatorii trebuie să se autentifice. În urma unei autentificări reușite, este generat un token JWT valid pentru 7 zile ce este semnat folosind o cheie secretă stocată în Secret Manager. Secret Manager este o unealtă care stochează datele confidențiale pe durata dezvoltării unui proiect ASP.NET Core. Secretele sunt stocate într-o locație separată de directorul proiectului, evitând astfel contorizarea lor de către un sistem de versionare[[8]](#footnote-8). Token-ul JWT este returnat aplicației client, care trebuie să îl includă în câmpul Authorization din header-ul cererilor HTTP ulterioare.

Un middleware va verifica dacă există un token în câmpul Authorization din antetul cererii HTTP. Dacă nu există sau dacă acesta nu este valid, atunci niciun utilizator nu este atașat contextului HTTP și cererea poate accesa doar rutele publice. Dacă există, se extrage numele utilizatorului din token și se atașează contextului HTTP.

Procesul de autorizare constă în a verifica dacă există un utilizator atașat cererii curente (contextului HTTP curent). Dacă autorizarea se face cu succes, nu este întreprinsă nicio acțiune și cererea este transmisă mai departe controller-ului. Dacă autorizarea eșuează, este returnat un răspuns cu statusul 401 (neautorizat).



[Procesul de autentificare și autorizare]

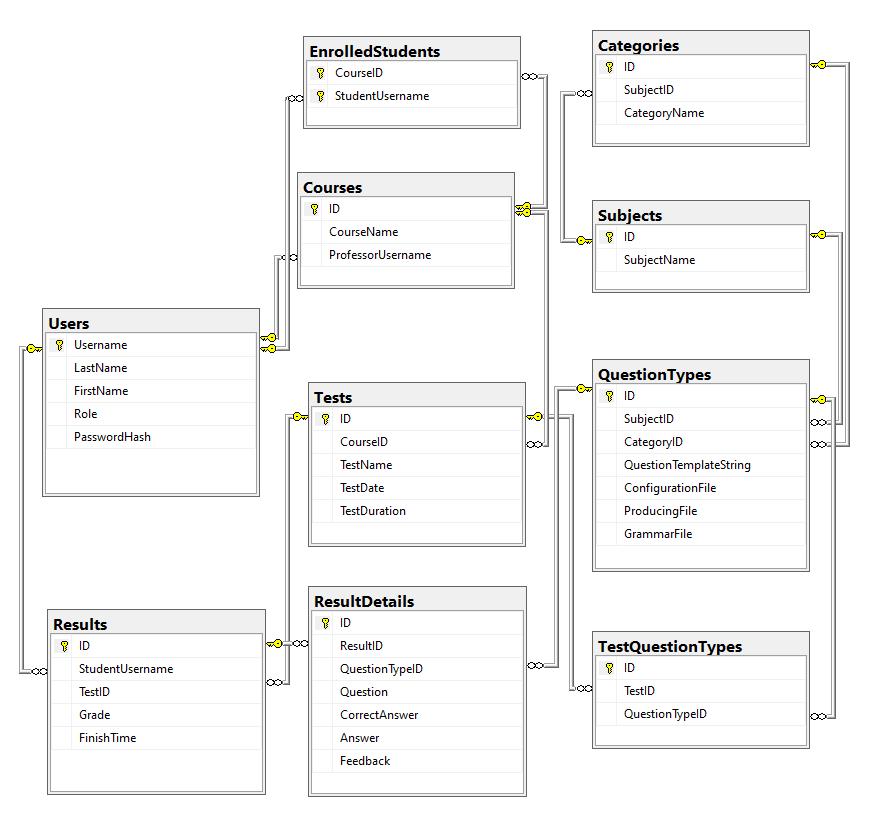
Rutele restricționate doar utilizatorilor autentificați sunt decorate cu un atribut de autorizare ([Authorized]). Opțional, pot fi specificate doar anumite roluri (de exemplu, profesor, student sau administrator) pentru a restricționa și mai mult accesul. Dacă este specificat un rol, ruta e accesibilă doar utilizatorilor cu acel rol, altfel ruta e accesibilă tuturor utilizatorilor autentificați, indiferent de rol. Astfel, o rută securizată acceptă cereri HTTP și furnizează un răspuns doar dacă antetul cererii conține un câmp de autorizare cu un token JWT valid.

Interacțiunea cu baza de date

Modelele reprezintă formele pe care le pot lua cererile și răspunsurile pentru metodele controller-ului. Modelele definesc parametrii cererilor care sunt primite sau datele care sunt returnate. Astfel, în momentul primirii unei cereri, conținutul acesteia poate fi deserializat într-un obiect[[9]](#footnote-9). Modelele mai sunt folosite pentru a transmite date între diferite părți ale aplicației (de exemplu, între servicii și controllere).

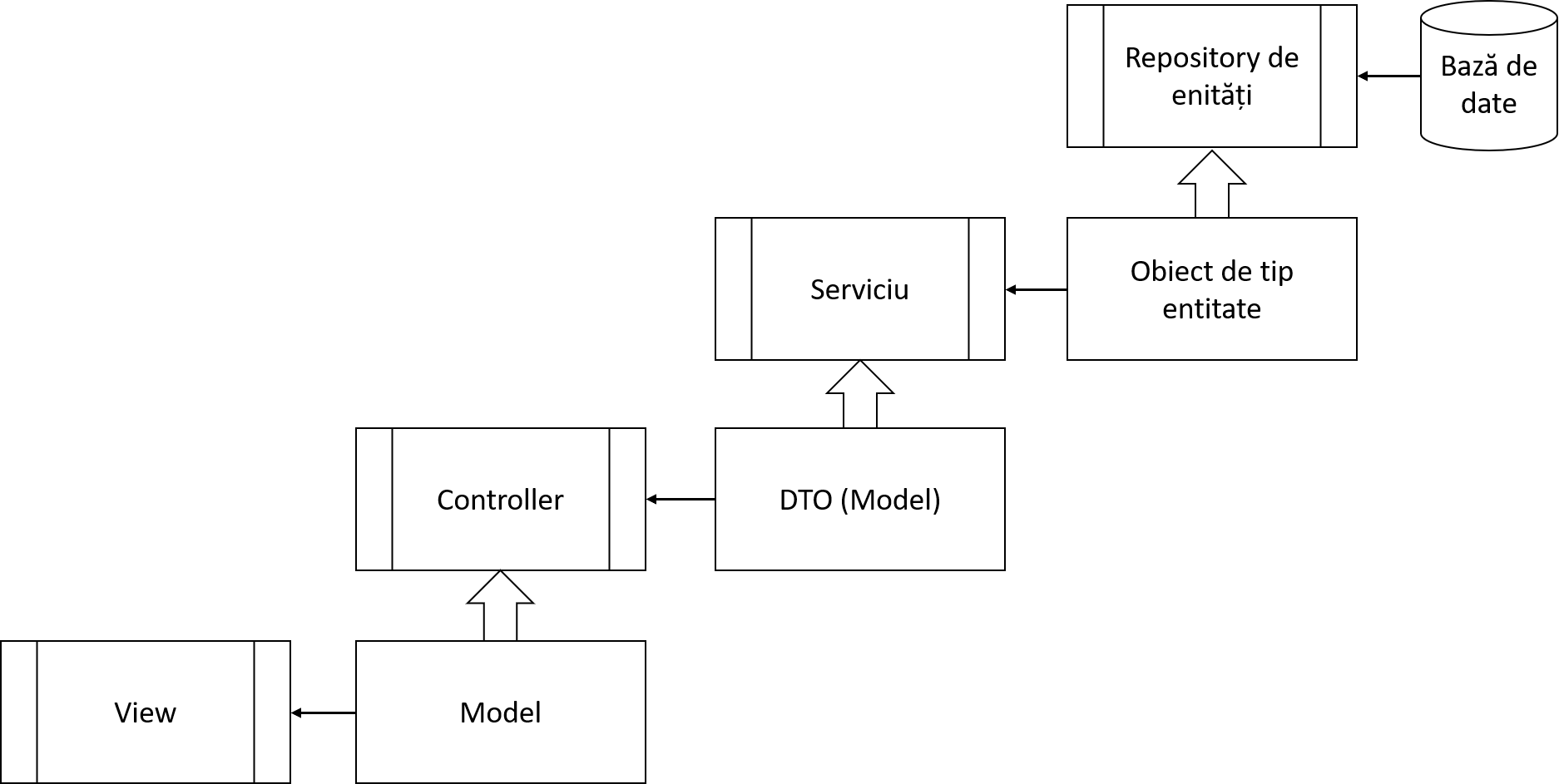
Entitățile modelează datele care sunt stocate în baza de date. Entity Framework Core mapează datele din baza de date relațională cu instanțele unor obiecte, ce sunt utilizate în cadrul aplicației pentru gestionarea datelor și operațiuni CRUD.

Pentru accesarea datelor aplicației este utilizată o clasă asociată contextului bazei de date. Derivă din clasa Entity Framework DbContext și are mai multe proprietăți publice asociate tabelelor, facilitând accesarea și gestionarea datelor în cadrul unei sesiuni[[10]](#footnote-10). Contextul bazei de date este utilizat de servicii pentru a efectua toate operațiunile CRUD.



[Diagrama bazei de date]

Tabelele din baza de date relațională sunt mapate sub forma unor entități. Entitățile sunt trimise către servicii, unde sunt procesate (filtrate, combinate, etc.) și convertite în Data Transfer Objects. DTO-urile (modelele) sunt trimise controller-ului, care la rândul lui le trimite către View.



[Fluxul de transmitere al informației]

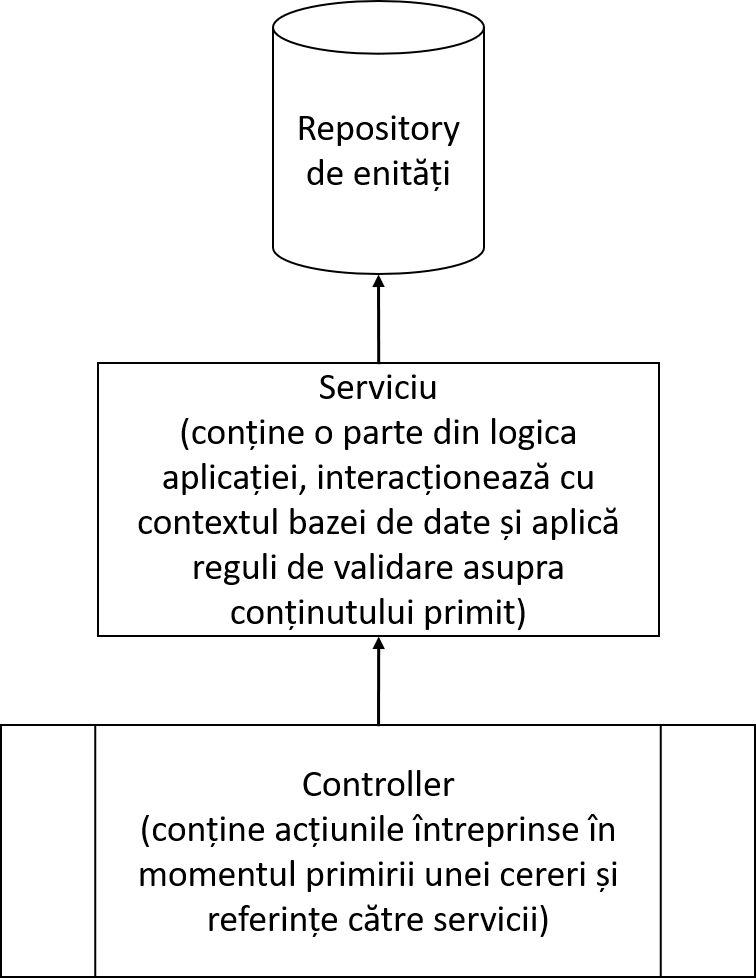
Pe parcursul implementării, a fost necesară luarea unei decizii cu privire la modul de stocare al fișierelor asociate întrebărilor (fișierul de configurare, fișierul producător și fișierul cu gramatica, dacă e cazul). O opțiune este stocarea fișierelor în baza de date. O altă opțiune este stocarea acestora în sistemul de fișiere și referențierea în baza de date. Russell Sears, Catharine van Ingen și Jim Gray (Russell Sears, 2006) au ajuns la concluzia că dacă documentele sunt mai mici de 256KB, stocarea în baza de date este mai eficientă. Dacă documentele au peste un 1MB, stocarea în sistemul de fișiere e mai eficientă. Cum fișierele folosite în cadrul proiectului au mărimi între 0,5 și 6 KB, opțiunea aleasă a fost cea a stocării în cadrul bazei de date.

Implementarea serviciilor

Una din etapele procesului de implementare a fost definirea locului unde se va afla logica internă a aplicației și implementarea funcționalităților propriu-zise. Modelul poate deveni extins pentru a fi capabil să efectueze o cerere către baza de date și să formateze răspunsul în mod corepunzător, ca să fie returnat View-ului. Însă, cum un model poate fi folosit ca răspuns la o cerere GET sau ca încapsulator al datelor unei cereri POST, acest plus de funcționalitate se poate dovedi redundant. În cazul unei cereri POST nu sunt necesare metodele asociate modelului, ci doar câmpurile postate către server (doar informația).

Dacă logica este pusă în controller, atunci acesta va avea responsabilități pe care nu ar trebui să le aibă, precum validarea datelor de intrare, generarea de modele, formatarea răspunsului, etc. De asemenea, acțiunile întreprinse pentru gestionarea cererilor vor fi greu de citit.

În ambele cazuri, nu sunt respectate principiile POO, clasele devenind din ce în ce mai încărcate, adresând diverse probleme. Astfel, este necesară adaugarea unui nou strat în cadrul arhitecturii aplicației, de care să fie dependent controller-ul. Logica aplicației este la rândul ei, dependentă de datele furnizate de mediul de stocare. Stratul care se ocupă cu logica aplicației poate fi intitulat un serviciu, iar în acest caz ierarhia componentelor, arată ca în figura următoare.



[Poziționarea serviciilor în fluxul de lucru al aplicației]

Pentru fiecare cerere, controllerul apelează un serviciu ce efectuează acțiunile necesare, menținându-se complet separat de logica aplicației și de accesul la baza de date. Serviciile conțin logica aplicației, ocupându-se și cu validarea sau accesul la baza de date.

Serviciile dedicate subiectelor, categoriilor și întrebărilor conțin metode ce privesc adăugarea și obținerea acestora de către un administrator. Serviciul de întrebări mai are, adițional, o metodă de generare a întrebării în momentul susținerii unui test de către un student.

Serviciul dedicat testelor conține metode de adăugare a unui test de către un profesor, de obținere a testelor create sau de susținut, în cazul studenților. Acesta mai permite și obținerea detaliilor despre un anumit test precum denumirea acestuia, data, ora, itemii din componență, și în cazul profesorilor, rezultate sumare și statistici referitoare la studenții care au parcurs un anumit test.

Serviciul dedicat rezultatelor conține metode de adăugare a unei intrări în tabela de rezultate, atunci când un student începe un test, de inserare a răspunsurilor la întrebările din test și de actualizare a intrării cu rezultatul final. De asemenea, mai permite și obținerea testelor specifice unui student la care acesta a furnizat răspunsuri pentru a putea fi listate corespunzător în interfață.

Serviciul dedicat utilizatorilor conține metodele necesare pentru accesarea funcționalităților platformei precum cele de înregistrare și autentificare.

Serviciile sunt disponibile în cadrul aplicației prin Dependency Injection. Acestea sunt furnizate componentelor care au nevoie de ele ca parametrii în cadrul constructorilor[[11]](#footnote-11). Constructorul unui controller va primi drept parametrii referințe către serviciile pe care le utilizează. Constructorul unui serviciu utilizează DI pentru a injecta contextul bazei de date folosit pentru orice operație CRUD.

În cadrul fișierului Program.cs este realizată înregistrarea serviciilor pentru dependency injection și a pipeline-ului prin care trec cererile HTTP (middleware). Fiecare serviciu are asociată o singură instanță pe timpul rulării aplicației. Pipeline-ul de gestionare al cererilor este alcătuit din componente denumite middleware care efectuează operații asupra contextului HTTP al acestora și fie invocă următorul middleware fie termină procesarea lor (de exemplu, middleware-ul de autentificare prin JWT) [[12]](#footnote-12).

Configurarea unei aplicații în ASP.NET Core se realizează printr-un furnizor de configurație, care citește datele sub forma cheie-valoare dintr-o sursă precum fișierul appsettings.json.

1. https://ocw.cs.pub.ro/courses/pw/laboratoare/03, accesat la 23.04.2022 [↑](#footnote-ref-1)
2. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 25.04.2022 [↑](#footnote-ref-2)
3. https://blog.e-zest.com/setting-up-hybrid-structure-of-mvc-and-reactjs, accesat la 23.04.2022 [↑](#footnote-ref-3)
4. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio, accesat la 25.04.2022 [↑](#footnote-ref-4)
5. https://reactjs.org/docs/create-a-new-react-app.html, accesat la 14.04.2022 [↑](#footnote-ref-5)
6. https://jonhilton.net/understanding-the-asp-net-react-template/, accesat la 24.04.2022 [↑](#footnote-ref-6)
7. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 24.04.2022 [↑](#footnote-ref-7)
8. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-6.0, accesat la 03.05.2022 [↑](#footnote-ref-8)
9. https://www.thisdot.co/blog/adding-react-to-your-asp-net-mvc-web-app, accesat la 25.04.2022 [↑](#footnote-ref-9)
10. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio, accesat la 27.04.2022 [↑](#footnote-ref-10)
11. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model?view=aspnetcore-6.0&tabs=visual-studio, accesat la 15.05.2022 [↑](#footnote-ref-11)
12. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=windows, accesat la 12.05.2022 [↑](#footnote-ref-12)