Explanations in English

1. Login Functionality

- **Main Files:**
    - login.component.html, login.component.scss, login.component.ts
    - AuthenticationController.cs
- **Description:**
    - **LoginComponent:** This Angular component is responsible for the user interface of the login page. The user enters their username and password in the login form, and upon clicking the "Login" button, the data is sent to the backend.
    - **Flow:** After the user fills out the form and clicks the login button, the loginInput() method in the login.component.ts file is triggered. This method collects the data from the form and sends it to the server using the login() method in AuthenticationService. The backend processes the request through AuthenticationController.cs, checking if the user exists and if the password is correct. If the authentication is successful, the user is redirected to the product list page.

2. Forgot Password Functionality

- **Main Files:**
    - forgot-password.component.html, forgot-password.component.scss, forgot-password.component.ts
    - AuthenticationController.cs
- **Description:**
    - **ForgotPasswordComponent:** This Angular component provides an interface for users who have forgotten their password. The form allows the user to input a new set of credentials: username and new password.
    - **Flow:** After the user enters the new credentials and clicks the reset button, the resetPassword() method in forgot-password.component.ts is triggered. This method checks if the entered passwords match and then sends the request to the server via AuthenticationService. The backend, through AuthenticationController.cs, verifies the existence of the user and updates the password in the database. If the process is successful, the user is notified and redirected to the login page.

3. User Registration Functionality

- **Main Files:**
    - register.component.html, register.component.scss, register.component.ts
    - AuthenticationController.cs
- **Description:**
    - **RegisterComponent:** This component is responsible for the user registration interface. The user needs to enter a username and password to create an account.
    - **Flow:** After completing the registration form and clicking the "Register" button, the registerInput() method in register.component.ts is triggered. This method collects the form data and sends it to the server to create a new user. AuthenticationController.cs receives the request, checks if the username is already in use, and if not, creates a new account and saves the data to the database. After registration, the user is redirected to the login page.

4. Product Management

- **Main Files:**
    - production.component.html, production.component.scss, production.component.ts
    - production-add-modal.component.html, production-add-modal.component.scss, production-add-modal.component.ts
    - production-edit-modal.component.html, production-edit-modal.component.scss, production-edit-modal.component.ts
    - production-delete-modal.component.html, production-delete-modal.component.scss, production-delete-modal.component.ts
    - ProductsController.cs, ProductService.cs, ProductRespository.cs
- **Description:**
    - **ProductionComponent:** This is the main component that manages the display, addition, editing, and deletion of products.
    - **Modals:** The component uses modals to allow users to add, edit, or delete products. Each modal is a separate component: ProductionAddModalComponent, ProductionEditModalComponent, ProductionDeleteModalComponent.
    - **Flow:** Upon opening the product management page, ProductionComponent loads the list of products from the server using the getProducts() method in ProductionService. Users can filter products or select a product to edit or delete. Each operation (add, edit, delete) is managed through a specific modal that sends requests to the server via ProductService. The backend, through

ProductsController.cs, delegates these operations to ProductService, which in turn interacts with the database via ProductRespository.

5. Navigation (Nav Bar)

- **Main Files:**
    - nav-bar.component.html, nav-bar.component.scss, nav-bar.component.ts
- **Description:**
    - **NavBarComponent:** This component provides navigation between the different sections of the application, such as login, product list, and registration. It also manages the login and logout functionalities.
    - **Flow:** The component checks the user's authentication status and displays the appropriate buttons (Login, Logout, Register). When the Logout button is clicked, the logOut() method is triggered, which deletes the authentication cookie and redirects the user to the login page.

6. Asynchronous Loading (Spinner Loading)

- **Main Files:**
    - spinner.component.html, spinner.component.scss, spinner.component.ts
    - loading.interceptor.ts
- **Description:**
    - **SpinnerComponent:** This component displays a loading spinner on the screen during asynchronous operations, such as HTTP requests to the backend.
    - **Flow:** The component is integrated with LoadingInterceptor, which intercepts all HTTP requests and activates the spinner when a request is in progress. The spinner is deactivated once the request is completed, providing visual feedback to the user that the application is processing an operation.

7. Modal Management

- **Main Files:**
    - modal.component.html, modal.component.scss, modal.component.ts
- **Description:**
    - **ModalComponent:** This generalized component is used to create reusable modal interfaces throughout the application. Each modal can have custom titles and actions configured through parameters.
    - **Flow:** Each modal is opened and managed through the open() method in ModalComponent, which loads the specific content and waits for user actions,

such as confirmation or closing the modal. These modals are mainly used for product management (adding, editing, deleting).

8. Database and Application Context

- **Main Files:**
    - AppContext.cs, Product.cs, User.cs
- **Description:**
    - **AppContext:** This class manages the connection and interaction with the database using Entity Framework Core. AppContext defines the entities (User, Product) and how they are mapped to the database tables.
    - **Flow:** For any data-related operation (such as adding a user or a product), AppContext is used to interact with the database. The Product and User classes define the structure of the database table for their respective entities, and AppContext ensures that the data is correctly read and written.

Explanations in Romanian

1. Autentificare (Login)

- **Fișiere principale:**
    - login.component.html, login.component.scss, login.component.ts
    - AuthenticationController.cs
- **Descriere:**
    - **LoginComponent:** Este componenta Angular responsabilă pentru interfața de utilizator a paginii de login. Utilizatorul introduce numele de utilizator și parola în formularul de login, iar la apăsarea butonului de "Login", datele sunt trimise către backend.
    - **Fluxul:** După ce utilizatorul completează formularul și apasă pe butonul de login, metoda loginInput() din componenta login.component.ts este declanșată. Aceasta colectează datele din formular și le trimite către server folosind metoda login() din AuthenticationService. Backend-ul procesează cererea prin AuthenticationController.cs, verificând dacă utilizatorul există și dacă parola este corectă. Dacă autentificarea reușește, utilizatorul este redirecționat către pagina de listă de produse.

2. Resetare Parolă (Forgot Password)

- **Fișiere principale:**
    - forgot-password.component.html, forgot-password.component.scss, forgot-password.component.ts
    - AuthenticationController.cs
- **Descriere:**
    - **ForgotPasswordComponent:** Această componentă Angular oferă o interfață pentru utilizatorii care au uitat parola. Formularul permite introducerea unui nou set de date: numele de utilizator și parola nouă.
    - **Fluxul:** După ce utilizatorul introduce noile date și apasă butonul de resetare, metoda resetPassword() din forgot-password.component.ts este declanșată. Aceasta verifică dacă parolele introduse coincid și apoi trimite cererea către server prin intermediul AuthenticationService. Backend-ul, prin AuthenticationController.cs, verifică existența utilizatorului și actualizează parola în baza de date. Dacă procesul este finalizat cu succes, utilizatorul este informat și redirecționat la pagina de login.

3. Înregistrare Utilizator (Register)

- **Fișiere principale:**
    - register.component.html, register.component.scss, register.component.ts
    - AuthenticationController.cs
- **Descriere:**
    - **RegisterComponent:** Este responsabil pentru interfața de înregistrare a unui nou utilizator. Utilizatorul trebuie să introducă un nume de utilizator și o parolă pentru a crea un cont.
    - **Fluxul:** După completarea formularului de înregistrare și apăsarea butonului de "Register", metoda registerInput() din register.component.ts este declanșată. Aceasta colectează datele din formular și le trimite către server pentru a crea un nou utilizator. AuthenticationController.cs primește cererea, verifică dacă numele de utilizator este deja utilizat și, dacă nu, creează un nou cont și salvează datele în baza de date. După înregistrare, utilizatorul este redirecționat către pagina de login.

4. Gestionare Produse (Product Management)

- **Fișiere principale:**
    - production.component.html, production.component.scss, production.component.ts
    - production-add-modal.component.html, production-add-modal.component.scss, production-add-modal.component.ts
    - production-edit-modal.component.html, production-edit-modal.component.scss, production-edit-modal.component.ts
    - production-delete-modal.component.html, production-delete-modal.component.scss, production-delete-modal.component.ts
    - ProductsController.cs, ProductService.cs, ProductRespository.cs
- **Descriere:**
    - **ProductionComponent:** Aceasta este componenta principală care gestionează afișarea, adăugarea, editarea și ștergerea produselor.
    - **Modale:** Componenta folosește modale pentru a permite utilizatorilor să adauge, editeze sau să șteargă produse. Fiecare modal este o componentă separată: ProductionAddModalComponent, ProductionEditModalComponent, ProductionDeleteModalComponent.
    - **Fluxul:** La deschiderea paginii de gestionare produse, ProductionComponent încarcă lista de produse de la server folosind metoda getProducts() din ProductionService. Utilizatorii pot filtra produsele sau selecta un produs pentru a-l

edita sau șterge. Fiecare operațiune (adăugare, editare, ștergere) este gestionată printr-un modal specific care trimite cererile către server prin ProductService. Backend-ul, prin ProductsController.cs, delegă aceste operațiuni către ProductService care, la rândul său, interacționează cu baza de date prin ProductRespository.

5. Navigare (Nav Bar)

- **Fișiere principale:**
    - nav-bar.component.html, nav-bar.component.scss, nav-bar.component.ts
- **Descriere:**
    - **NavBarComponent:** Aceasta componentă oferă navigarea între diferitele secțiuni ale aplicației, precum autentificare, listă de produse și înregistrare. De asemenea, gestionează funcționalitățile de login și logout.
    - **Fluxul:** Componenta verifică starea de autentificare a utilizatorului și afișează butoanele corespunzătoare (Login, Logout, Register). La apăsarea butonului de Logout, metoda logOut() este declanșată, care șterge cookie-ul de autentificare și redirecționează utilizatorul către pagina de login.

6. Încărcare Asincronă (Spinner Loading)

- **Fișiere principale:**
    - spinner.component.html, spinner.component.scss, spinner.component.ts
    - loading.interceptor.ts
- **Descriere:**
    - **SpinnerComponent:** Aceasta componentă afișează un spinner de încărcare pe ecran în timpul cererilor asincrone, cum ar fi cererile HTTP către backend.
    - **Fluxul:** Componenta este integrată cu LoadingInterceptor, care interceptează toate cererile HTTP și activează spinner-ul când o cerere este în curs de execuție. Spinner-ul este dezactivat odată ce cererea este completată, oferind feedback vizual utilizatorului că aplicația procesează o operațiune.

7. Gestionarea modale (Modal Management)

- **Fișiere principale:**
    - modal.component.html, modal.component.scss, modal.component.ts
- **Descriere:**
    - **ModalComponent:** Această componentă generalizată este folosită pentru a crea interfețe modale reutilizabile în aplicație. Fiecare modal poate avea titluri și acțiuni personalizate prin intermediul configurărilor transmise ca parametri.

- **Fluxul:** Fiecare modal este deschis și gestionat prin metoda open() din ModalComponent, care încarcă conținutul specific și așteaptă acțiuni de la utilizator, cum ar fi confirmarea sau închiderea modalului. Aceste modale sunt utilizate în principal pentru gestionarea produselor (adăugare, editare, ștergere).

8. Baza de date și Contextul aplicației (Database and Application Context)

- **Fișiere principale:**
  - AppContext.cs, Product.cs, User.cs
- **Descriere:**
  - **AppContext:** Este clasa care gestionează conexiunea și interacțiunea cu baza de date, folosind Entity Framework Core. AppContext definește entitățile (User, Product) și cum sunt ele mapate la tabelele din baza de date.
  - **Fluxul:** Pentru orice operațiune legată de date (cum ar fi adăugarea unui utilizator sau a unui produs), AppContext este folosit pentru a interacționa cu baza de date. Clasele Product și User definesc structura tabelului din baza de date pentru entitățile respective, iar AppContext asigură că datele sunt corect citite și scrise.

Aceasta este o detaliere elaborată a fiecărei funcționalități majore din aplicația ta. Aceste explicații pot fi folosite pentru documentarea proiectului pe GitHub sau pentru înțelegerea mai profundă a structurii și logicii aplicației tale. Dacă ai nevoie de alte clarificări sau detalii suplimentare, sunt aici să te ajut!