**Semestrální práce  
Viktoriia Garkusha**

**TWAII**

Aplikace slouží pro správu skladů a nákladu pro letadla. Administrátor (Admin) může přidávat, upravovat a mazat záznamy (sklady, letadla, náklady), zatímco běžný uživatel (User) má k dispozici pouze náhled a nemůže provádět změny. **1. Úvod**

Projekt je postaven na technologii **Blazor Server** (ASP.NET Core), používá **Entity Framework Core** a **Microsoft SQL Server** pro ukládání dat a **ASP.NET Core Identity** pro správu uživatelů a jejich rolí. Smyslem je mít webovou aplikaci, kde se dají spravovat:

* **Sklady (Warehouses)**
* **Letadla (Airplanes)**
* **Náklady (Cargos)**

Hlavní cíle:

1. **CRUD operace** (Create, Read, Update, Delete) nad entitami: sklady, letadla, náklady.
2. **Autentizace a autorizace** uživatelů, aby jen administrátor (Admin) mohl přidávat, mazat či upravovat data. Běžný uživatel (User) má mít pouze možnost prohlížet.

**2. Architektonické principy**

1. **Blazor Server** – místo, aby se celé UI vykreslovalo na klientovi (jako u SPA), běží vše na serveru a prohlížeč komunikuje pomocí SignalR.
2. **ASP.NET Core Identity** – knihovna pro přihlašování uživatelů a správu rolí (Admin, User).
3. **Entity Framework Core** – objektově-relační mapper pro komunikaci s MSSQL databází.
4. **Dvě úrovně ochrany dat**:
   * **UI** (Blazor) skrývá akce jako Edit, Delete a umožní je jen roli Admin.
   * **Server-side** ochrana – pomocí [Authorize(Roles="Admin")] na kontrolérech či stránkách se zabrání i ručnímu volání API těmi, kdo nejsou Admin.

Důvodem této architektury je kombinovat jednoduchou tvorbu uživatelského rozhraní (Blazor), kde není nutné psát JavaScript, s robustním mechanismem pro uživatelské účty (Identity). Tím se dosahuje rychlého vývoje, přehledné správy a lepší bezpečnosti (hlavně díky roli Admin a ověřování na serveru).

**3. Aplikace a datová vrstva**

**3.1 Databázové kontexty**

* **DataContext** spravuje tabulky pro doménové entity (Warehouse, Airplane, Cargo). Zde se staráme o vztahy mezi tabulkami a případné seedovací (testovací) záznamy.
* **AuthDbContext** (IdentityDbContext) je určen pro tabulky jako AspNetUsers, AspNetRoles atd. Umožňuje tak používat vestavěné funkce Identity (registrace, login, role).

Díky tomu lze oddělit čisté doménové modely (sklady, letadla, zásilky) od uživatelů a jejich oprávnění. Při migracích se tak generují různé tabulky pro Identity a pro samotná data.

**3.2 Modely**

Základní modely jsou:

* **Warehouse** (jednoduše WarehouseCode a Location, omezení délky kódu a názvu).
* **Airplane** (kód letadla, maximální nosnost a přidružený sklad, pokud letadlo stojí ve skladu).
* **Cargo** (kód nákladu, popis, váha a příznak, zda je ve skladu nebo v letadle).

Délky kódů, validace (např. max. 5 znaků pro WarehouseCode) a regex na alfanumerické znaky zajišťují čistotu dat.

**4. Služby a logika (Services)**

Každá hlavní entita (Warehouse, Airplane, Cargo) má vlastní **service** vrstvu (např. WarehouseService). Ta nabízí metody jako:

* GetAllAsync() – načítá seznam.
* GetByIdAsync(...) – načítá konkrétní záznam.
* AddAsync(...), UpdateAsync(...), DeleteAsync(...) – základ CRUD.

Ve službách se provádí i **validace** (kontrola délky kódu, povolených znaků, atd.). Pokud vstup nevyhoví, vyhodí se InvalidOperationException, aby uživatel dostal chybovou hlášku a data se neukládala.

**Proč** je vhodné mít tyto služby?

* **Oddělení logiky** od uživatelského rozhraní.
* Možnost stejných služeb volat z kontrolérů (API) i z Blazor komponent.

**5. Uživatelské rozhraní (Blazor Server)**

Blazor Server umožňuje mít .razor soubory pro jednotlivé stránky:

* **Warehouses.razor** – seznam skladů, tlačítko „Add a New Warehouse“, volá WarehouseService.GetAllWarehousesAsync.
* **EditWarehouse.razor** – formulář pro úpravu/přidání skladu. Při odeslání volá WarehouseService.UpdateWarehouseAsync nebo AddWarehouseAsync.

Totéž platí pro letadla (Airplanes.razor, EditAirplane.razor) a zásilky (Cargos.razor, EditCargo.razor).

**Role** Admin vs. User se na úrovni komponent řeší tak, že:

* Skryjeme tlačítka „Edit“, „Delete“, „Add“ pro roli User.
* [Authorize(Roles="Admin")] můžeme dát přímo na stránky typu /edit-warehouse, takže user s rolí „User“ se na stránku vůbec nedostane (403 Forbidden).

Důvodem je **dvojitá ochrana**:

1. V UI se nezobrazí akční tlačítka pro běžné uživatele.
2. Na serveru [Authorize(Roles="Admin")] zajistí, že i když by user zkusil ručně napsat adresu, dostane zákaz.

**6. Autentizace a role**

**6.1 ASP.NET Core Identity**

Aplikace využívá zabudované **Identity**. To umožňuje:

* Snadný login/registraci přes /Identity/Account/Login a /Identity/Account/Register (vestavěný UI nebo sám skaffoldovaný).
* Správu uživatelů v tabulce AspNetUsers v databázi.
* Tvorbu rolí v tabulce AspNetRoles.

V kódu (typicky v Program.cs) se nastaví:

csharp

Copy code

builder.Services.AddDefaultIdentity<IdentityUser>()

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<AuthDbContext>();

Díky tomu je možné volat:

csharp

Copy code

await userManager.AddToRoleAsync(user, "User");

await userManager.AddToRoleAsync(user, "Admin");

a rozlišovat jejich oprávnění.

**6.2 Jak je řešen Admin**

* Při startu se často vytváří dvě role: "Admin" a "User".
* Jeden účet (např. boss@example.com) dostane roli "Admin".
* Ostatní nově registrovaní získají roli "User".

Tím je zaručeno, že jen Admin vidí a volá úpravy a mazání. Běžný uživatel (User) se smí jen dívat.

**7. API Kontroléry (volitelné)**

Pokud aplikace kromě Blazor stránek poskytuje i HTTP rozhraní, existují **Controllers** s [ApiController]. Tam se přidá [Authorize(Roles="Admin")] na metody POST, PUT, DELETE. Tím je zaručeno, že běžný user obdrží **403** při pokusu o zápis.

Hlavní motivace:

* Mít možnost volat API z externích klientů (mobilní aplikace, Postman).
* Oddělit logiku do standardních REST endpointů.

**8. Nasazení a běh**

1. **Connection String** – v appsettings.json se nastaví "DefaultConnection" na správný SQL server.
2. **Migrace** – dotnet ef migrations add ... a dotnet ef database update pro DataContext a AuthDbContext.
3. **Spuštění** – dotnet run (nebo nasazení na IIS/Azure).
4. **Přístup** – v prohlížeči na https://localhost:5001/warehouses (popř. /airplanes, /cargos) a /Identity/Account/Login.

**9. Shrnutí**

Projekt demonstruje následující klíčové myšlenky:

* **Blazor Server** jako rychlý způsob tvorby server-side UI, včetně formulářů a tabulek.
* **Entity Framework Core** pro dotazy do MSSQL, s entitami Warehouse, Airplane, Cargo.
* **ASP.NET Core Identity** k řešení uživatelské přihlášky, uložených hesel, a hlavně **rolí** pro autorizaci:
  + Admin → přístup k úpravám a mazání.
  + User → pouze čtení (omezená tlačítka v UI, a na serveru blokace přes [Authorize(Roles="Admin")]).
* **Kombinace** skrytí akcí v uživatelském rozhraní a server-side kontrola zajišťuje bezpečnost.

Díky této architektuře lze snadno rozšiřovat:

* Přidat další entity (např. objednávky, zákazníky).
* Rozšířit Identity (dvojfaktorová autentizace, e-mailové potvrzení účtu).
* Vyexportovat data (reporty, statistiky) – to vše v rámci stávajícího EF a Blazor UI.